FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	RFID communication eavesdropping
Student:	Jan Havránek
Supervisor:	Ing. Jiří Buček, Ph.D.
Study Programme:	Informatics
Study Branch:	Computer Security and Information technology
Department:	Department of Computer Systems
Validity:	Until the end of summer semester 2019/20

Instructions

Study the topic of contactless chip card communication in accordance with ISO / IEC 14443, especially Type A.
Survey existing solutions for eavesdropping of communication between a contactless chip card and its reader.
Study the architecture and functionality of the Proxmark3 device with an emphasis on the parts of its design that implement eavesdropping (sniffing).
Expand the functionality of Proxmark3 so that you can transfer sniffed data to your PC or mobile phone in real time.
Program or expand an existing application to receive and view data from the device.
The application will perform at least basic decoding of the transmitted frames (frame type determination).
Test the solution on a number of different readers and contactless cards and determine possible throughput bottlenecks.

References

Will be provided by the supervisor.

prof. Ing. Pavel Tvrdík, CSc.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 14, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

RFID communication eavesdropping

Jan Havránek

Department of Computer Systems
Supervisor: Ing. Jiří Buček, Ph.D.

May 16, 2019

Acknowledgements

I would like to thank my supervisor, Ing. Jiří Buček, Ph.D., for his valuable advice, guidance and feedback. Next, I want to thank the lecturers of the course BI-DPR for helpful tips regarding typography and the overall thesis structure, and lastly, I want to thank my family and friends for their support during my work on this thesis.

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 that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 16, 2019 ……

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Havránek, Jan. *RFID communication eavesdropping.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstract

This thesis summarizes the principles of ISO/IEC 14443 type A communication. Next, it researches existing devices for eavesdropping and analysis of RFID communication, analyzes the capabilities of the Proxmark3 hardware device and extends its functionality in the area of eavesdropping RFID communication according to the ISO/IEC 14443A standard. The extension allows for real time eavesdropping of the communication along with a mobile application for the Android operating system that enables the user to view the eavesdropped data and its basic analysis.

Keywords mobile application, Proxmark3, Android, RFID, eavesdropping, sniffing, ISO/IEC 14443

Abstrakt

Tato práce se zabývá shrnutím fungování komunikace typu A dle standardu ISO/IEC 14443. Dále se zabývá rešerší existujících zařízení pro odposlouchávání a analýzu RFID komunikace, analýzou fungování hardwarového přípravku Proxmark3 a rozšiřuje jeho funkcionalitu v oblasti odposlouchávání RFID komunikace dle standardu ISO/IEC 14443A. Rozšíření přípravku umožňuje odposlech RFID komunikace v reálném čase. Součástí implementace je mobilní aplikace pro operační systém Android umožňující zobrazit odposlechnutá data a jejich základní analýzu.

Klíčová slova mobilní aplikace, Proxmark3, Android, RFID, odposlech, ISO/IEC 14443

Contents

List of Figures

List of Tables

List of Listings

Introduction

Smart cards, and contactless smart cards in particular, play a significant role in today's modern society. The two most prominent use cases of contactless smart cards are payment and authentication, both of which carry critical importance to the users. Therefore, it is vital that the cards are secure and their security is thoroughly tested.

The importance of payment security is quite obvious as it is a financially attractive target for attackers. In the case of authentication and access control, an attacker can impersonate the user (e.g. an employee), access restricted areas and perform malicious activities on his behalf, potentially causing damage to the company and causing trouble for the user.

Smart cards often utilise proprietary communication protocols although there is also a number of open protocols used. Nonetheless, it is important to question and analyze the security of such protocols. One way of performing such analysis is to eavesdrop the RFID communication and subsequently decode and examine the data in detail, which is the topic that this thesis addresses.

Goal of the analytic part of this thesis is to review the principles of contactless chip card communication in accordance with ISO/IEC 14443A, and to survey existing solutions for eavesdropping of communication between a contactless chip card and the reader. Specifically, the aim is to study the functionality of the Proxmark3 RFID research device with an emphasis on the parts of its design that implement eavesdropping.

Goal of the practical part of this thesis is to expand the eavesdropping functionality of Proxmark3 and develop or expand existing client application to enable user to view the sniffed data in real time. Further aim is to implement basic decoding of the received data in the client application. Last goal of the thesis is to test the implemented solution on a number of different readers and contactless cards and determine possible throughput bottlenecks.

The first chapter, *The ISO/IEC 14443 standard*, is concerned with the details of type A RFID communication per the mentioned standard. It de-

scribes the physical layer of the communication, its modulation sequences, bit and byte formats, and also with the higher layer commands and anticollision procedure. The chapter also briefly summarizes the types of communication frames described in 4th parth of ISO14443.

Chapter *Existing solutions* discusses the already existing solutions, devices and approaches to eavesdropping RFID communication. It mentions how the communication can be "sniffed" using an oscilloscope, mentions specialized NFC analysis devices and lastly examines specialized open source devices for RFID research.

Chapter *Proxmark3 in detail* focuses on one specific open source RFID device, Proxmark3. The chapter describes its architecture, capabilities with emphasis on the parts related to its RFID eavesdropping functionality and, consistently with the thesis' goals, further focuses on the existing Android client applications for Proxmark3.

Design and implementation, as the name implies, deals with designing the implementation goals of this thesis – a Proxmark3 firmware modification for real time RFID sniffing and a companion Android client application for receiving and decoding the data.

Chapter *Testing* proposes a testing approach for the implementation discussed in previous chapter and applies it to properly test the solution.

Chapter 1

The ISO/IEC 14443 standard

This thesis focuses on the part of the ISO/IEC 14443 standard that describes the type A RFID communication with bitrate of $\frac{fc}{128}$ (~106 kbit/s) where fc (carrier frequency) is the frequency of the RF operating field of 13.56 MHz ± 7 kHz as defined in part 2 of the standard [1]. The waveform of the carrier frequency with its period measured is shown in Figure 1.1.

1.1 Signal interface

This section decribes the details of the signal interface (i.e. physical layer) of the ISO14443A communication per part 2 of the standard [1].

1.1.1 Reader to tag communication

The initiating side of the communication is always the reader. It generates a HF (high frequency) alternating magnetic field which delivers power and

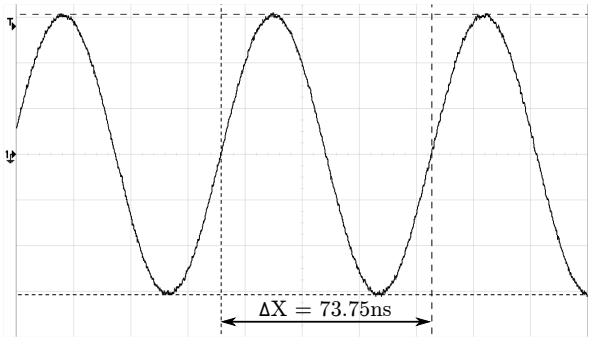

Figure 1.1: Waveform of the carrier frequency ($\frac{1}{\Delta X} \approx 13.56$ MHz)

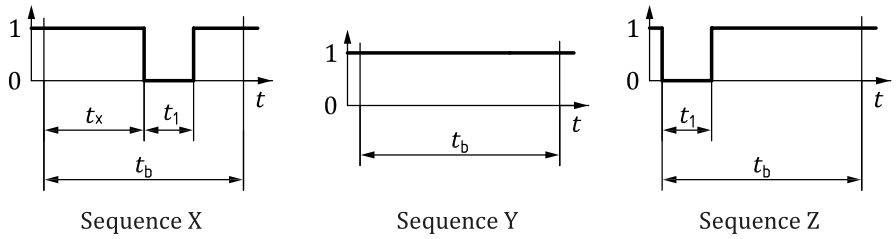

Figure 1.2: Sequences for type A reader to tag communication [1]

clock to the tag through inductive coupling and is subsequently modulated to transfer data. The tag is in most cases entirely powered by the reader's field as the tags are not required to have a power supply of their own.

The data transmission from the reader to the tag is ensured by 100% ASK (amplitude shift keying) modulation of the reader's field with modified Miller coding. The standard defines the following modulation sequences:

- sequence X: a *PauseA* follows after half of a bit duration (t_x);
- sequence Y: no modulation for the full bit duration (t_b);
- sequence Z: a *PauseA* occurs at the beginning of the bit duration (t_b).

PauseA is defined as the reader modulation pause of type A communication. Apart from parameter t_1 (length of *PauseA*), another three timing parameters are defined for the bitrate of $\frac{fc}{128}$: t_2, t_3 and t_4. These parameters determine how the reader's field should behave during the modulation pause (field rise time, etc.) Details of the *PauseA* modulation are not important for the purposes of this thesis and are therefore ommited.

The parameters t_b, t_x and t_1 for bitrate $\frac{fc}{128}$ are defined as follows:

- $t_b = \frac{128}{fc} \approx 9.44\mu s$;
- $t_x = \frac{64}{fc} \approx 4.72\mu s$;
- $\frac{28}{fc} \leq t_1 \leq \frac{40.5}{fc} \approx 2.06\mu s \leq t_1 \leq 2.99\mu s$.

The standard uses sequences shown in Figure 1.2 to code data using modified Miller coding in the following manner:

- start of communication: sequence Z;
- end of communication: logic 0 followed by sequence Y;

- logic 0: sequence Z with an exception in case a contigous sequence of 0s does *not* immediately follow a start of communication, sequence Y is used for the first 0;
- logic 1: sequence X;
- no information: at least two sequences Y.

Example of a reader to tag communication utilizing the modified Miller coding is shown in Figure 1.3.

1.1.2 Tag to reader communication

As mentioned before, the initiating side of the communication is the reader which activates the RFID tag through inductive coupling after the tag is placed in the operating field of the reader. The inductive coupling on the tag's side is typically ensured by a large area antenna coil typically with 3–6 windings of wire [2].

Transmission of data from the tag to the reader is achieved through load modulation and Manchester coding. The carrier frequency is loaded by the tag with a subcarrier frequency, fs, for different halves of the bit duration. The subcarrier frequency for the bitrate of $\frac{fc}{128}$ is defined as $\frac{fc}{16}$ (~848 kHz). The subcarrier is generated by switching a load in the tag. The standard defines the following modulation sequences:

- sequence D: the carrier is modulated for the first half of the bit duration;
- sequence E: the carrier is modulated for the second half of the bit duration;
- sequence F: the carrier is not modulated for one bit duration.

The standard utilizes the mentioned sequences for coding information as follows:

- start of communication: sequence D;
- end of communication: sequence F;
- logic 0: sequence E;
- logic 1: sequence D;
- no information: no subcarrier.

Example of a tag to reader communication utilizing the Manchester coding is shown in Figure 1.3.

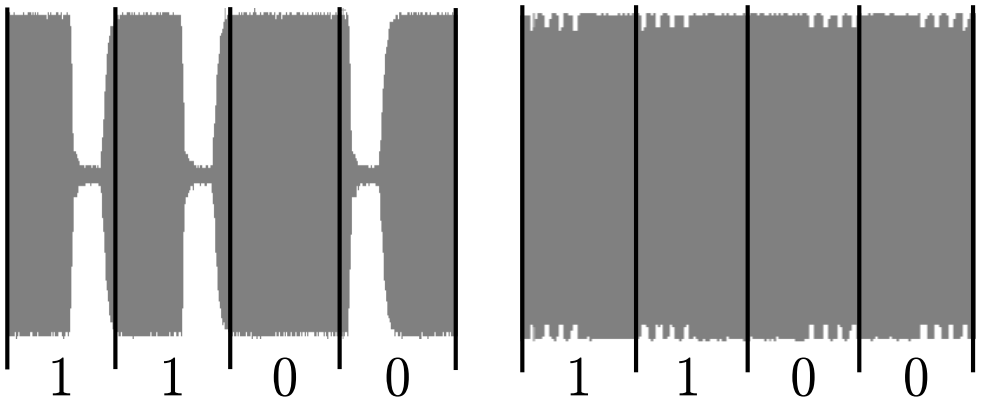

Figure 1.3: Example communication signals – reader to tag (modified Miller, left) and tag to reader (Manchester, right)

1.2 Initialization and anticollision

This section addresses the initialization and anticollision process of ISO14443 type A communication and summarizes the details of bit trasmission, byte format, frame format and basic commands that are important for the initialization and anticollision procedure and further communication per part 3 of the standard [3].

1.2.1 Initialization

The reader is by default in an idle state where it generates the HF operating field and alternates between type A and type B modulation until it detects a tag in the field and starts communicating with it. The selected type is kept for the whole communication session until the reader deactivates the tag or the tag is removed from the operating field.

The initial step in the communication is the exposure of the tag to the reader's operating field. The tag gets powered by the reader's operating field, activates and silently waits for commands from the reader.

1.2.2 Anticollision

The anticollision procedure is used to detect whether there is more than one tag in the reader's operating field and subsequently select only a single one of the present tags to communicate with.

For the purposes of the anticollision procedure and further communication, the standard declares that the communication shall be transferred in

pairs, reader to tag followed by tag to reader. First the reader frame is transferred, after which a reader frame delay time (FDT) occurs, then the tag frame is transferred back to the readers and a tag delay time occurs. For the values of the FDTs see [3].

There are three types of frames defined for the $\frac{fc}{128}$ bitrate:

- short frame;
- standard frame;
- bit oriented anticollision frame.

Short frame consists of a start of communication (S), 7 bits of data LSB (least significant bit) first and an end of communication (E). There is no parity bit in short frame.

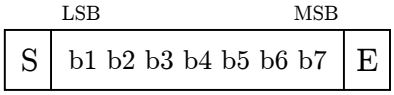

Figure 1.4: Short frame, adapted from [1]

Standard frame consists of a start of communication, n data bytes (8 bits LSB first) each followed by an odd parity bit (P) and an end of communication. The odd parity bit is set to a value such that the number of ones in the data byte along with the parity bit (9 bits total) is odd. For $\frac{fc}{128}$ bitrate the format of standard frame is identical for both reader and tag. There is an exception for tag's standard frame in case the frame is transmitted with a bitrate higher than the one mentioned. In that case, the last parity bit is inverted.

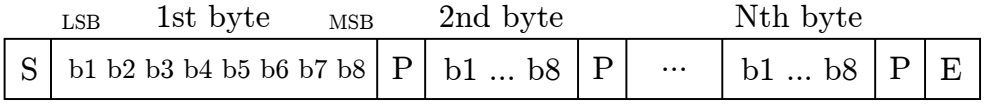

Figure 1.5: Standard frame, adapted from [1]

Bit oriented anticollision frame is only used during the anticollision loop. Structure of the frame is the same as the structure of a standard frame with the difference that it can be split on the bit level into two parts in case there is a collision.

A collision occurs when two or more tags transmit different bit values at the same time and the field is load modulated for the whole bit duration which is detected by the reader.

During the anticollision process, the reader utilizes the following commands to communicate information with the tag:

- *REQA*, *WUPA* (short frame) – type A request, type A wakeup
- *ANTICOLLISION* (bit oriented anticollision frame)
- *SELECT* (standard frame)
- *HLTA* (standard frame) – type A halt

In the beginning of the anticollision procedure, the reader probes its operating field for type A tags by repeatedly sending the *REQA* request command. If the tag is presented in the reader's field it activates and responds with an *ATQA* command. In *ATQA*, the tag signals the size of its UID (unique identifier). The UID can consist of four, seven or ten bytes, called single, double or triple UID respectively. Size of the UID determines the needed level of cascade loop (CLn) that needs to be executed for the reader to obtain the whole UID of the tag.

During the first cascade level (CL1), the reader requests the UID of the tag using an *ANTICOLLISION* command. The tag responds with 4 bytes of its UID and a BCC (block check character) byte. The BCC byte is calculated as bitwise exclusive or (XOR) of the preceding 4 bytes. If a collision occurs, the reader sends another *ANTICOLLISION* command in which it repeats the bits of the UID that it received before the collision occured and appends a single chosen bit (value depends on the reader implementation). Now only the tags whose UID begins with the given bit sequence respond. This process is repeated until no collision occurs. At that point, the reader received part of the UID for CL1 and issues the *SELECT* command. The tag ackowledges the selection by a *SAK* command and signals if the complete UID of the tag was transmitted. The anticollision loop for single size UID tags ends here and the tag signals that the UID si complete in *SAK*. In case the tag has double or triple UID size, the same anticollision loop continues for CL2 and CL3 respectively until the tag's full UID is communicated to the reader.

The remaining commands – *HLTA* and *WUPA* – are used for suspending and waking up the tag, respectively. In case the reader needs to communicate with another tag, it can halt the currently selected tag and eventually wake it up later with these two commands.

1.3 ISO/IEC 14443-4 block types

The last message in the anticollision procecure is *SAK*, which apart from the UID completion also denotes if the given tag is compliant with 4th part of the ISO14443 standard which describes a public transmission protocol for the

communication. Some tags are not compliant with it and can further operate using a proprietary protocol.

In case the tag is compliant, the anticollision procedure is followed by the tag receiving the *RATS* (request for answer to select) command from the reader and responding with *ATS* (answer to select) command in which it denotes various parameters for further communication. Next, the communication itself begins. For the purposes of this thesis, only the types of blocks that are further transmitted are important, therefore other details of the communication are ommited.

Type of the transmitted block is determined by the two most significant bits of the first byte field, the protocol control byte. There are three types of blocks defined in the standard:

- I-block: used to transfer information for the application layer (e.g. an application protocol data unit – APDU – per ISO/IEC 7816);
- R-block: contains positive or negative acknowledgement messages that relate to last received block;
- S-block: used to exchange control information between the reader and the tag.

Chapter 2

Existing solutions

This chapter discusses existing solutions and approaches for eavesdropping RFID communication. This includes a basic analysis with an oscilloscope and a proper probe, dedicated NFC analyzers and lastly specialized open source devices.

2.1 Oscilloscope

First of the researched methods of RFID eavesdropping is the usage of an oscilloscope. This method is very dependent on the technical parameters of given device. Oscilloscopes without any RFID analysis capabilities are suitable for analysis of physical characteristics of the radio communication (field strength, timings, etc.) but unfit for intercepting and decoding larger amounts of data that is sent between the card and the reader. That is caused by the limited size of the scope's memory and because the oscilloscope captures a large amount of samples. This fact enables the user to examine the waveform of the reader's RF field in great detail but limits the length of the communication that can be captured.

While working on this thesis, I had access to Agilent MSO6104A [4] oscilloscope and the RF-R 400-1 probe [5]. Although the scope does not have any NFC related functionality, I was able to capture a small part of communication between the reader (ACR122 [6]) and the tag (ISIC card) using the pulse width triggering with the pause length set to the value of parameter t_1 (see section 1.1.1). Thanks to the beginning sequence (sequence Z) which includes a pause, this triggering method was sufficient. The oscilloscope setup is shown in figure 2.1.

The captured communication is shown in Figures 2.2 and 2.3. Figure 2.2 shows the initial request from the reader (*REQA*) and the corresponding response from the tag (*ATQA*). The value of the tag's response (4403 in hexadecimal) corresponds with the *ATQA* for Mifare Desfire EV1 [7], the type

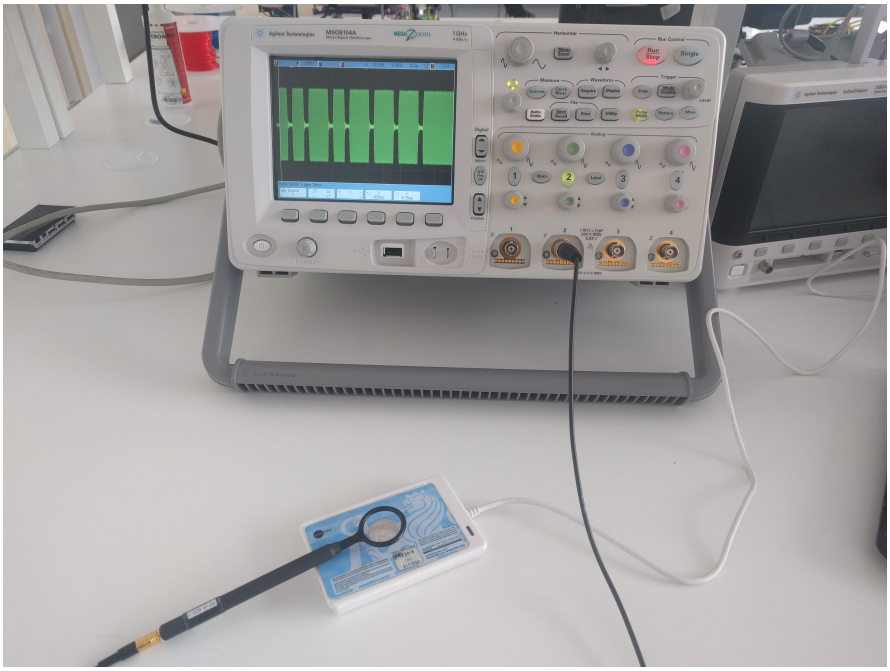

Figure 2.1: Oscilloscope setup

of card used at CTU (Czech Technical University in Prague). Decoded bits are in the LSB first order as defined in [3]. Letters in the figures have the following meaning:

S start of communication

E end of communication

P odd parity bit

Example of a device with RFID analysis functionality is the Keysight's InfiniiVision 4000 X-Series oscilloscope. Specifically, the oscilloscope is capable of utilizing software for NFC analysis and automated tests. According to Keysight's website [8], the device software's capabilities are NFC triggering and automated testing of NFC-A among others. Demonstration of the NFC analysis functionality can be seen in videos [9] and [10].

Disadvantage of the oscilloscope eavesdropping method is its costliness as oscilloscopes are expensive devices in comparison to some single purpose devices intended to be used specifically for NFC or RFID analysis.

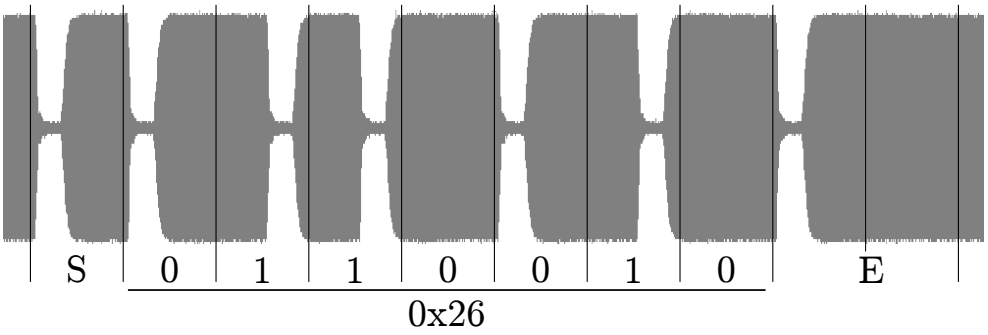

Figure 2.2: Example of oscilloscope eavesdropped communication from reader to tag (short frame, LSB first, *REQA*)

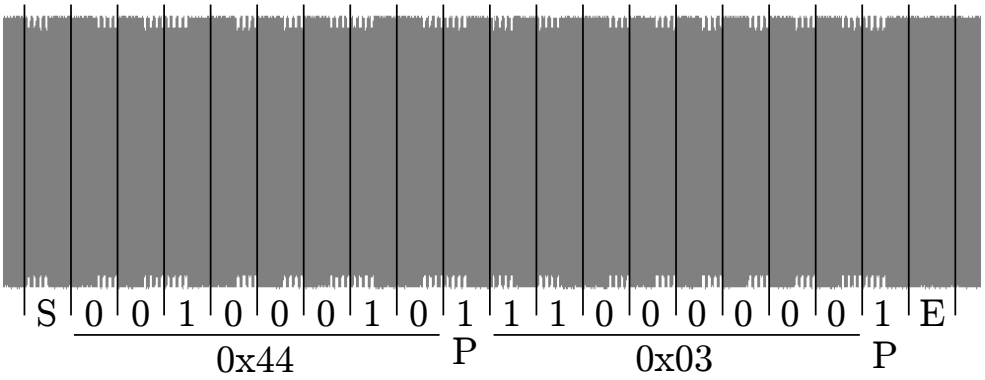

Figure 2.3: Example of oscilloscope eavesdropped communication from tag to reader (standard frame, LSB first, *ATQA*)

2.2 NFC protocol analyzers

Another option for intercepting RFID communication is the usage of specialized devices for analyzing such communication. Example of one such device is the ComProbe® NFC Protocol Analyzer [11] which is a proprietary single purpose device for capturing and analyzing NFC traffic. The vendor includes a companion software for the device which allows for live capture and decoding of NFC communication [12].

As I did not find any official price while working on the thesis, the following is only a speculation and my opinion on the matter. Although the analyzers might be more affordable than high end oscilloscopes, they might be also quite

costly mainly due to the added support and services from the vendors.

Apart from the price, their another big disadvantage is the fact that they are proprietary and therefore not easily expandable.

2.3 Software defined radio

Software defined radio (SDR) is a radio communication system where traditionally hardware components (e.g. filters, amplifiers, etc.) are instead implemented in software on a computer or embedded system [13]. The signal is therefore converted from analog to digital as soon as possible and further processing is done in software. This allows for the usage of SDR in RFID sniffing.

Cheap USB dongles meant for reception of multiple ranges of radio signals – mostly DVB-T (digital video broadcasting — terrestrial), DAB (digital audio broadcasting) and FM (frequency modulation) radio – can be often used as SDR. This is possible thanks to the manufacturers using a software tunable chip (e.g. RTL2832U).

The lower tunable range of such DVB-T dongles often ends at 20 MHz, therefore they are not able to tune down to the NFC frequency of 13.56 MHz.

The author of [14] used one such DVB-T dongle and managed to overcome the frequency limitation by applying a software patch to the SDR driver. As mentioned in the blog post, a second solution to overcoming the limitation would be to tune the dongle to the second or third harmonic frequencies of NFC (respectively 27,12 MHz; 40,68 MHz).

Subsequently, the author was able to demodulate and decode the reader's side of communication in the GNURadio software and therefore eavesdrop one way of the NFC communication using an SDR.

The reader's side of communication is easier to eavesdrop due to the big differences in field strength in ASK modulation. On the other hand, the communication from the tag is ensured by load modulation of the reader's operating field which is harder to capture. The author of aforementioned post did not continue his research on whether he could sniff the tag's side of communication and as of making of this thesis, I did not find any other research on this topic.

2.4 Open source devices

Last researched option for RFID eavesdropping are three specialized open source devices. Both their hardware and software are open source. Open source hardware (OSH) enables anyone with proper equipment and hardware skills to build the device and use it while the open source software (OSS) offers the users an option to alter the source code and amend the devices' functionality to their liking. The openness of the devices also allows anyone

to build and sell the devices. That leads to many options when looking to buy such a device in case the user does not have the time, skills or resources to build the device himself. On the other hand, the openness may also lead to fragmentation and confusion between different hardware revisions, clones, etc. which can be viewed as a slight disadvantage.

2.4.1 ChameleonMini

ChameleonMini [15] is a device focused on work with the high frequency (HF, 13.56 MHz) RFID. It can read as well as emulate a wide range of tags and is also capable of sniffing both ways of the communication. For list of supported tags and procotols, see [15]. ChameleonMini was previously able to only sniff one way of the communication – from the reader to tag. As of today, ChameleonMini supposedly can sniff even the tag's side of communication as well [16]. Unfortunately, I did not have access to the device, therefore I could not test the device's capabilities myself.

2.4.2 HydraNFC

HydraNFC [17] is an expansion shield hardware for high frequency RFID research. It is originally designed to be used as an expansion of HydraBus [18] – an another piece of open source hardware for embedded research – but due to its open source nature and public specifications, anyone can create a piece of hardware that supports this shield.

Similarly to ChameleonMini, HydraNFC in conjunction with HydraBus can read UIDs from various tags, emulate them and also eavesdrop on both sides of ISO14443A communication. The sniffing feature works in real time and overcomes the transmission speed challenges by utilizing two USB ports of the user's computer – one for the configuration of the sniffer and the other for transmission of the sniffed data to the computer.

2.4.3 Proxmark3

Proxmark3 [19] is a very versatile tool specifically designed for RFID research. In contrast with the aforementioned open source devices, it is capable of working with both low frequency (LF, 125 kHz) and high frequency (HF, 13.56 MHz) tags and among other things can read most available RFID tags.

Reason I chose this device to work on is the fact that it is open source, relatively easily expandable and it was already available to me during work on this thesis as opposed to the other devices.

There is also a new revision of the device available, called Proxmark3 RDV4 [20], based on a fork of the original design.

Chapter 3 examines the architecture and capabilitites of Proxmark3 and tools surrounding it in more detail.

Chapter 3

Proxmark3 in detail

Proxmark3 is a powerful general purpose RFID tool, originally developed by Jonathan Westhues and released under the terms of the GPL license in 2009 [21]. It it further maintained and improved by a community of RFID researchers and enthusiasts. This thesis deals with the original hardware revision of the device and current version of software corresponding with the project's GitHub repository [22].

3.1 Architecture

This chapter discusses details of the Proxmark3 device. It describes its overall architecture and operation.

3.1.1 Hardware

The six main parts of the Proxmark3 hardware are:

- antenna connector;
- analog antenna circuits;
- analog to digital converter (ADC) – Texas Instruments TLC5540;
- field-programmable gate array (FPGA) – Xilinx Spartan-II XC2S30;
- microcontroller (MCU) – Atmel AT91SAM7S512;
- mini USB port.

The antenna connector allows for easy exchangeability of antennas. It utilizes a single 4-pin Hirose connector where two of the four pins are used for the LF antenna and two for the HF antenna. An alternative for homemade antennas is soldering the their wires directly onto the test pads placed around

the connector on the Proxmark's PCB (printer circuit board). The fact that each antenna uses different pair of pins or pads allows for connecting both of the antennas (LF and HF) at the same time.

The device's PCB includes two separate analog antenna circuits, one for LF and one for HF signal, that can be used independently. Output of these circuits is routed into the ADC which digitizes the analog signal to an 8-bit value that represents voltage on the antenna. The ADC output is connected to eight parallel input pins of the FPGA.

The FPGA preprocesses the raw digitized signal and sends it to the microcontroller which performs the higher level logic on the signal depending on current mode of operation.

Proxmark3 connects to the user's computer or mobile phone using a single USB interface. It presents itself as a single serial device (e.g. `/dev/ttyACM0` in Linux) that can be opened, read from and written to. This allows for easy integration with alternative clients, such as the Python client or Android mobile application implemented in the practical part of this thesis.

The PCB includes a single mini USB port connected directly to the Atmel microcontroller which includes a USB controller right on the chip. Thanks to this, Proxmark3 does not need any additional controller hardware for the USB communication.

For development purposes, there is also a single JTAG connector included on the PCB. This connector is mainly used for debugging and for flashing the microcontroller's firmware for the first time. It is also needed in case the bootrom (discussed in the next section) gets corrupted and cannot be used for flashing through USB anymore.

3.1.2 Firmware

Firmware of Proxmark3 has two parts – a bootrom and the control software itself. The bootrom's main purpose is to allow for easy flashing of the microcontroller through USB without the need to use the aforementioned JTAG connector.

When powered on, the microcontroller performs two key operations:

1. depending on the current mode (LF or HF), it flashes the FPGA with the corresponding hardware description image (the HF image is loaded is by default);

2. begins listening for USB commands in an endless loop (more on USB commands in section 3.1.3).

Both the microcontroller firmware and the client are written in the C programming language. The hardware configuration of the FPGA is described in Verilog, a hardware description language.

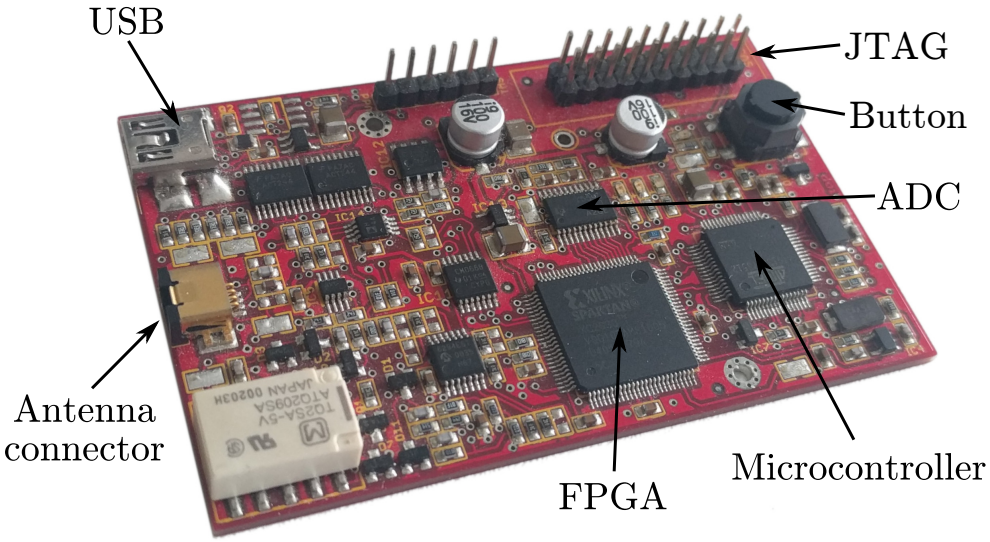

Figure 3.1: Proxmark3

3.1.3 Command structure

The operation of Proxmark3 is controlled from the user's device using a single USB port. The communication is ensured by sending a `UsbCommand` structure back and forth. The structure consists of four 8 byte fields – the type of command and three command arguments – and one 512 byte array for data of the command. Total size of this structure is 544 bytes and it is sent every time. Therefore, both sides (computer/phone, Proxmark3) always check the communication channel and if there is any data available, they start receiving until they receive full 544 bytes of data before they take any action based on the received data.

Illustration of the command structure can be seen in Table 3.1. This information is important later in the thesis as the practical part deals with the limitations that this design of communication introduces.

Field	Size
Command type	8 bytes
Parameter 1	8 bytes
Parameter 2	8 bytes
Parameter 3	8 bytes
Data	512 bytes

Table 3.1: `UsbCommand` structure

3.1.4 Default PC client

The default C client that is included in the Proxmark3 repository is a simple command line interface for interacting with the device. It includes a help text for each command with explanation of the command's parameters and features a history of recently used commands.

Similarly to the main loop of the microcontroller the PC client includes a loop that is checking the USB communication channel and is ready to receive a `UsbCommand` structure. Communication from Proxmark3 to the client is mainly utilized when the client wants to download data from the device or Proxmark3 itself sends debug messages to the client. This loop is implemented in a background thread while the main thread of the application is handling the user's input for the command line interface.

The client also allows for scriping in Lua which means that if a user wants to expand or implement a new functionality, he is not necessarily required to directly modify the C source code of the client and Proxmark but can utilize and combine existing already implemented functionalities. The official repository currently includes 27 Lua scripts. An example of such script is `mifare_autopwn.lua` which aims to automate attacks on Mifare Classic HF tags.

3.2 Capabilities

As mentioned, Proxmark3 is a very versatile device, that can be used for a wide range of reasearch tasks regarding RFID, including both LF and HF. It can work in a total of three modes – reader, tag and sniffer.

In the reader mode, Proxmark3 can read the UIDs of most available RFID tags. On top of that, it can read more information from the tag, such as the memory contents of Mifare Classic tags. The following list shows some of the tags that Proxmark3 can work with:

- Low frequency tags:
  - EM410x – read only
  - EM4x05 – read/write
- High frequency tags:
  - Mifare Classic
  - Mifare Ultralight
  - NTAG 203, 213, 215, 216

For tags that Proxmark3 does not directly support in current version, there is an option to send raw arbitrary bytes to an ISO14443 tag. This adds an

effective support for any tag and allows for free experimentation with standard and non-standard commands.

As a tag, the device can simulate a number of different RFID tag types. Concrete examples are EM410x (very often used instead of keys for opening apartment building doors in Czech Republic), Jablotron and Visa2000 tags from the LF group and Mifare and iClass from the HF group.

In the sniffer mode, Proxmark3 can eavesdrop on a few different types of RFID communication:

- ISO14443 – both type A and type B (HF)
- iClass (HF)
- Hitag (LF)

This thesis is only concerned about the ISO14443A eavesdropping feature, discussed in section 3.3 and details about the other sniffing commands are therefore not mentioned in this work.

3.3 ISO14443A eavesdropping implementation

Proxmark3 already has the eavesdropping capability for ISO14443A communication implemented. This section focuses on and describes this functionality, its limitations and disadvantages.

As already mentioned in section 3.1.1, the whole data flow of the eavesdropping process – similarly to other tasks – begins with the device's antenna. The signal from the antenna travels through the HF peak detection circuit, is converted by the ADC to an 8-bit value which is then received by the FPGA. The FPGA preprocesses the signal and transforms it into a single bit stream of samples which represent state of the reader's field (modulated/not modulated). This bit stream is then sent to the microcontroller through its synchronous serial port (SSP).

The main function in the MCU's firmware which deals with the eavesdropping is called `SnoopIso14443a`. In the beginning of its operation, it sets up a circullar direct memory access (DMA) buffer, where the bits that are received through the synchronous serial port (SSP) are saved automatically without any action taken by the MCU. The buffer's size in current implementation is 128 bytes. The function then runs in a loop and calls functions `MillerDecoding` and `ManchesterDecoding`. These two functions, as their names imply, are crucial for finding the modulation patterns of modified Miller and Manchester coding and extracting the encoded data bits.

After the functions mentioned above decode a whole data frame either from the reader or from the tag, the `LogTrace` function gets called. This functions saves the decoded frame (also called trace) into a trace buffer. Data

from this buffer can later be extracted by the client application by issuing a corresponding command. Diagram of the described data flow is shown in Figure 4.1.

The eavesdropping loop is stopped by the user by pressing the Proxmark3 button.

Both the DMA buffer and the trace buffer are stored in a 40 kilobyte memory chunk called `BigBuf`. This buffer is mainly used for saving the traces (at low addresses) and also for allocating memory (at high addresses) for the needs of Proxmark3 operation, utilizing the `BigBuf_malloc` function.

Main limitation of the current implementation is the inability of the user to review the eavesdropped traffic in real time. It requires the user to start the process using the Proxmark3 command shell, perform the communication he wishes to examine, and finally use the command shell again to download the traffic from the device and review it. This presents a hassle in case there are errors in the sniffing process that the user cannot immediately see. Improvement of this limit is presented in the practical part of this thesis.

3.4 Existing Android clients

Apart from the option of performing RFID eavesdropping in real time, this thesis' motivation is also to simplify the usage of the eavesdropping functionality without the need to carry a laptop. Therefore, only Android client applications are further discussed. According to [23], for the use of Proxmark3 with a mobile device, there already exist four Android client applications:

- Proxdroid;
- angelsl;
- Walrus;
- AndProx.

Proxdroid and angelsl's client utilize a cross-compiled Proxmark3 client with modifications that allow them to be built as a standalone Android applications using Android NDK (native development kit). Unfortunately, these two applications require direct access to protected files of the Android operating system, namely the serial device of Proxmark3 in the `/dev` directory. To access this device, the user has to apply modifications to the phone's operating system and either "root" the OS – i.e. allow root (linux superuser) access to the system – or at the very least, modify the access permissions of the system files. On top of that, the applications are no longer maintained and therefore are not further discussed in this thesis.

On the other hand, AndProx and Walrus utilize the Android USB API and do not require any modifications to the operating system itself. The following sections introduce the applications and compare them.

3.4.1 Walrus

Walrus [24] is an application designed for physical security assessments. It provides the user with a unified interface for various RFID card cloning devices and allows for reading and saving the cards' UIDs along with the GPS location where the card was acquired.

Along with Proxmark3, it currently supports ChameleonMini and according to the project's website, the developers plan to add support for more devices in the future.

As mentioned, this application is focused on the card cloning aspect of Proxmark3 and therefore cannot utilize other funtionalities of the device, e.g. eavesdropping of the RFID communication.

3.4.2 AndProx

In contrast with Walrus, AndProx [25] aims to provide a fully featured command shell for Proxmark3 with the same functionality as the default PC client.

Similarly to Proxdroid, it utilizes the existing C Proxmark3 client, but does not require any elevated access to the Android system files and devices. It employs the Android USB API to connect to Proxmark3 and wraps the Java serial port so that the original Proxmark C client can utilize it. This allows the application to effectively reuse the existing C client implementation without significant changes and avoids the mentioned OS modifications.

Chapter 4

Design and implementation

This section describes the practical part of this thesis. It discusses its design and individual parts.

Goal of the implementation is to allow Proxmark3 to sniff RFID communication and transfer the eavesdropped data to a mobile application in real time. The solution consists of a modification to the existing Proxmark3 firmware and an implementation of a new Android application.

4.1 Real time eavesdropping functionality design

Important part of the implementation design was a certain level of experimentation with different approaches to transferring the eavesdropped data and testing if the reception speed is sufficient so that the Proxmark's internal DMA buffer does not overflow. While designing the solution, I utilized a simple proof of concept Python client, that helped me test the different approaches and their effectiveness.

The first and most naive approach tried was to send the data as part of already implemented debug messages where each eavesdropped frame was sent in one message. These messages respect the usb command architecture mentioned in section 3.1.3. Because of this, the data of the traces utilized only a part of the `UsbCommand` data array and the remaining bytes were filled with zero bytes. This inefficiency is most obvious in the case of short frames (e.g. *REQA*) where only the command type, first parameter (message length) and one byte of the data field in `UsbCommand` structure were used and the rest (527 bytes) wasted.

To respect the usb command architecture while not wasting throughput by transferring zero bytes, the potential solution would have to wait until the data field of `UsbCommand` structure was filled and send it then. This would lead to a state where the short messages and data exchanges that do not exceed the

512 byte boundary would not be sent immediately and the transfer effectively would not be performed in real time.

Due to the reasons mentioned before, I decided not to adhere to the usb command protocol in place. The only field conserved from the `UsbCommand` is the command type field to determine which type of message is sent to the client.

Initial approach was to just prepend the eavesdropped data with the command type field. This turned out as too slow because the client had to read each field (such as timestamp, duration, etc. discussed in the next section) one by one which introduced large delay times and caused the DMA buffer to overflow in the meantime. Solution to this problem was to include an another 2 byte field which denotes the total length of the whole trace message. This allowed the client to read only the first two fields one by one and then read the rest of the message at once and parse it later which significantly reduced the reception delay, therefore this approach was chosen to be used in conjunction with the application discussed later.

4.2 Proxmark3 firmware modifications

The modifications made to the Proxmark3 firmware are quite minimal and aim to achieve the goal without significant changes to the existing implementation.

Firstly, the size of the DMA buffer had to be increased as the access to the serial device through operating system APIs in Android is much slower than direct access to the serial device (e.g. `/dev/ttyACM0`) using the aforementioned Python client. The DMA buffer was enlarged to 16 kilobytes. Normally, this would not be desired because the memory space for saving of traces would be severely reduced but in case of the real time snoop function, the traces are not saved in memory, therefore it is not a complication.

The `SnoopIso14443a` function has a one single byte parameter in which two of the eight bits are used for existing flag parameters. The real time sniffing mode uses the third bit as a flag declaring if the eavesdropping should be performed in real time. Depending on the parameter, a logging function is selected using a function pointer (either the default function or the real time logging function). Further, if the real time flag is set, the function sends a 2 byte status message to signalize its end to the real time client application.

To make the existing and the new real time logging functions interchangeable, I avoided modifying the current function – `LogTrace` – and instead added a new fuction – `LogTraceRealtime`. The new real time function is very similar to the original `LogTrace` function but instead of saving the traces to the internal trace buffer, it sends the data over USB to the client application directly and omits saving the traces to the memory.

Format of the data sent over USB is described in Table 4.1. The newly added command type is called `CMD_ISO_14443A_REALTIME_TRACE`.

Field	Size
Command type	2 bytes
Total following trace length	2 bytes
Timestamp	4 bytes
Duration of the frame	2 bytes
Length of data	2 bytes
Data	n bytes
Parity data (one byte per 8 bytes of data)	m bytes

Table 4.1: Format of real time trace data

Figure 4.1 shows the data flow of the eavesdropping functionality described in section 3.3 along with the new firmware additions marked by gray color.

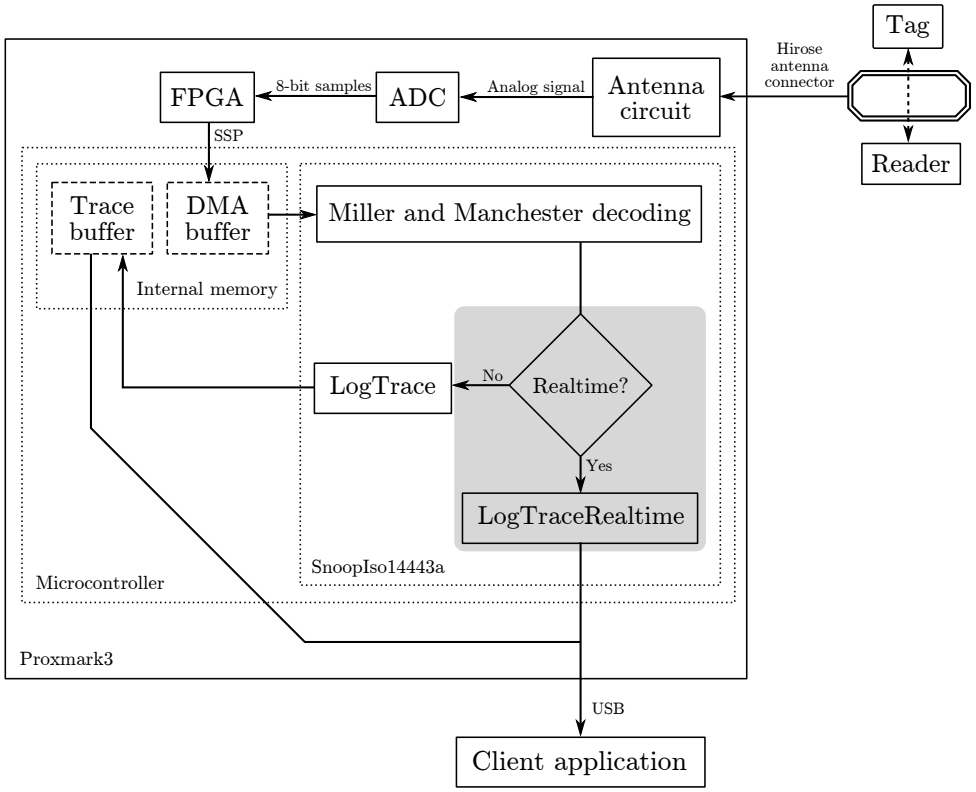

Figure 4.1: Proxmark3 eavesdropping data flow diagram

The data is received, further processed and shown to the user in the Android application discussed in the next section.

4.3 Android client application

The Android client application, called *RT Client* (i.e. RealTime Client), acts as a simple frontend to the newly implemented real time eavesdropping functionality. It receives the sniffed data, annotates the individual frames in accordance with ISO14443 and shows the data to the user in a list. The list interface aims to be clear and easily readable. It distinguishes between the reader and tag frames by utilizing two different colors along with a small icon at the beginning of every row of the list. It also clearly marks frames where parity errors has occured by and exclamation mark and a red warning sign.

Appearance of the application is shown in Figure 4.2.

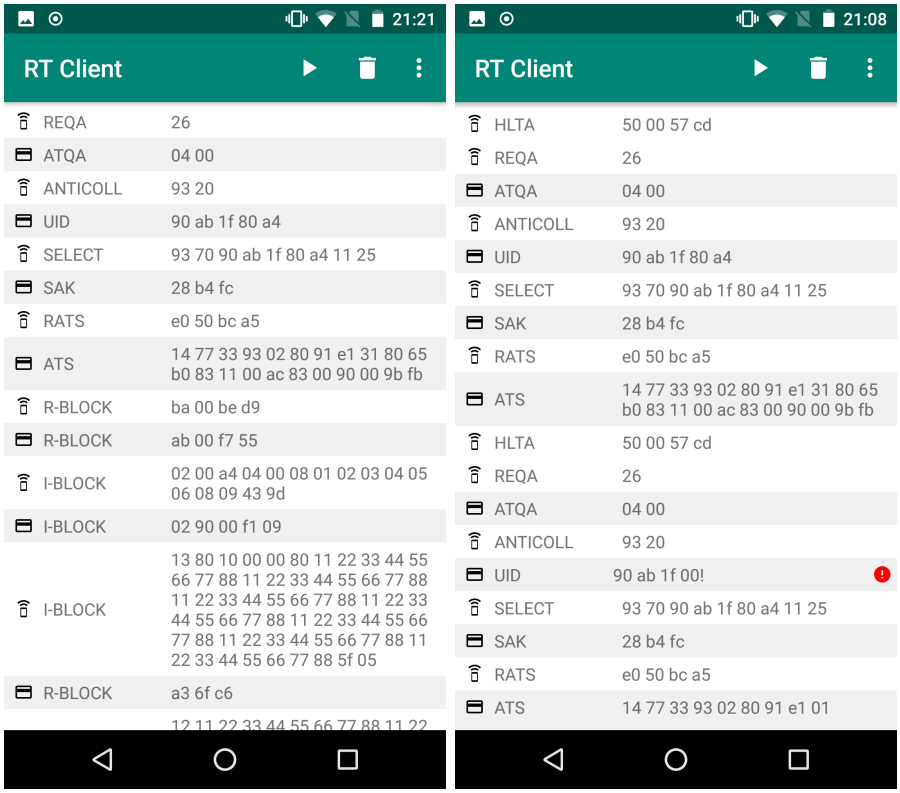

Figure 4.2: Android client application with eavesdropped anticollision and ISO14443-4 communication (left) and example of parity error (right)

The application consists of four main parts:

- Proxmark3 connection handled by main activity;
- data receiver thread;
- parser thread;
- trace list.

Main activity connects to Proxmark3, passes the connection to data receiver thread which downloads the traces from the device and further sends it to the parser thread. The parser thread processes the data and sends it back to the main activity. The data is added to the dataset of the adapter and shown in the list with proper annotations in accordance with ISO14443. Simplified data flow of the application is shown in Figure 4.3.

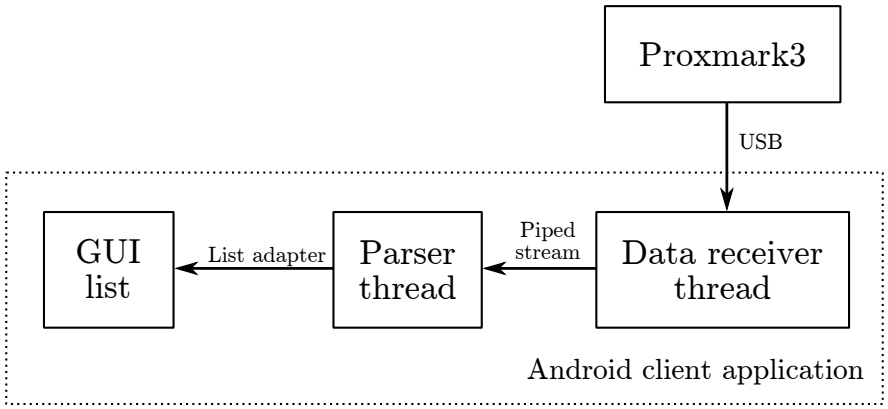

Figure 4.3: Application's data flow diagram

The individual components of the application are discussed in more detail in the following chapters.

4.3.1 Proxmark3 connection

The physical connection of Proxmark3 to the mobile phone is realized by one additional piece of hardware – a USB On-The-Go (OTG) adapter – which allows for connecting full size USB connector to the phone.

An existing OSS library called `usb-serial-for-android` [26] is employed for realizing the USB connection part. It simplifies work with USB serial hardware on Android, such as Arduino or, for the purposes of this thesis, Proxmark3. In contrast with some other mentioned Proxmark client applications, the library utilizes the Android USB host API and therefore does not require any Android OS modifications.

After the user starts the sniffing by pressing the start button, the main activity of the application searches the available USB devices using `UsbManager` system service. If it finds the Proxmark3 device using its USB identifiers (vendor and product ID), it checks if the user has granted the application a permission to use the device. If he did not, the user is asked to provide the permission. After obtaining it, the application passes a reference to the device to the serial library mentioned above and opens its serial port.

The reference to the opened serial port is further passed to the data receiver thread, discussed in the next section.

4.3.2 Data receiver thread

It is vital for the real time client to receive the eavesdropped data from Proxmark3 as fast as possible to avoid overflowing the circular DMA buffer where the device holds the incoming preprocessed data from the FPGA.

As the name suggests, the `DataReceiver` thread has exactly this purpose. Its goals are:

1. send `UsbCommand` with properly set parameter (mentioned in section 4.2) that signals Proxmark3 to start the eavesdropping loop;
2. read the data from Proxmark3 as fast as possible – to achieve that, the thread has its priority set to a maximum value;
3. send the received data to the parser thread.

The thread holds a reference to a `UsbSerialPort` object (provided by the `usb-serial-for-android` library) which represents the serial port of Proxmark3. It reads bytes from the serial port and immediately sends them to parser thread using a `PipedOutputStream`.

4.3.3 Parser thread

The parser thread receives the bytes from the receiver thread by reading from a `PipedInputStream` connected to a `PipedOutputStream` of the data receiver thread.

The thread recognizes a total of three types of incoming messages:

- debug messages with statistics sent by Proxmark3 at the end of the snoop function;
- end status message which signals an end of the sniffing loop to the application;
- real time trace message which contains the actual eavesdropped communication frames.

The debug messages are simply read and shown to the user as an Android toast (a small popup message).

End status message signals that the user ended the sniffing by the Proxmark3 button to the application which can subsequently stop both of the threads and update the inner state of the application accordingly.

The real time trace message is read and its bytes are passed to the constructor of `Trace` class (a wrapper class for one data frame). The constructor parses

the individual byte fields, described in section 4.2, using Java's `ByteBuffer` class. The thread then takes the newly created `Trace` object and sends it back to the main activity which adds it to the existing dataset of the trace list adapter.

4.3.4 Trace list

The list of traces (frames) is the main part of the application's graphical interface. The instance of `TraceListAdapter` class holds a dataset (`ArrayList`) of `Trace` objects and handles how the individual rows are filled with data.

Depending on the direction of communication a given frame was transmitted (either reader to tag or tag to reader), it sets the background color and icon of the list row.

Then a specialized function handles the basic annotation of frames. It can annotate frames that are part of the anticollision procedure and frames that are compliant with the 4th part of ISO14443 standard.

For reader to tag anticollision communication, the annotation function checks the value of frame bytes and selects appropriate annotation text (e.g. *REQA*, *SELECT*, etc.). For tag to reader anticollision communication, it checks the last reader frame and, depending on its type, identifies the type of the tag frame.

ISO14443-4 frames are annotated using a bit mask which checks the two most significant bits of the first byte in a given frame and determines the frame type (*I/R/S-block*) based on the masked result.

The adapter then fills the text field with hexadecimal representation of the eavesdropped data and finally it checks if there are parity errors in the frame data and marks the row with the aforementioned warning sign if there are any.

Chapter 5

Testing

This chapter discusses how the solution implemented in the practical part of this thesis was tested.

5.1 Equipment and approach

Multiple different tags were used for the testing:

- Mifare Classic 1K;
- Mifare Desfire EV1;
- GemCombiXpresso R4 72K;
- NXP NTAG213.

For reading the tags, multiple NFC readers were used:

- ACR122
  - Plug and play USB smartcard reader
  - CCID and PC/SC compliant
  - NXP PN532 NFC controller
- Nexus 5 (N5)
  - Released in November 2013
  - 2 GB RAM
  - Qualcomm MSM8974 Snapdragon 800 (Quad-core 2.3 GHz) [27]
  - Android 7.1.2 (LineageOS 14.1 – a custom Android ROM)
  - Broadcom BCM20793M NFC controller [28]

- Nexus 5X (N5X)
  - Released in October 2015
  - 2 GB RAM
  - Qualcomm MSM8992 Snapdragon 808 (Hexa-core 4x1.4 GHz Cortex-A53 and 2x1.8 GHz Cortex-A57) [29]
  - Android 8.1.0 (LineageOS 15.1)
  - NXP PN548 NFC Controller [30]
- Motorola E4 Plus (ME4P)
  - Released in June 2017
  - 3 GB RAM
  - Mediatek MT6737 (Quad-core 1.3 GHz Cortex-A53) [31]
  - Android 7.1.1 (OEM)
  - Probably NXP NFC controller (supports Mifare Classic)

As the main stress test, the GemCombiXpresso smart card with an "echo" application (so called applet) was used. This applet as its name implies does a single thing – receives data sent by the reader and echoes it back. This applet allowed me to send very large amounts of data very fast and test if the DMA buffer does not overflow with increasing amount of data transferred.

I primarily tracked the utilization of the DMA buffer as it tells me if the data transfer can keep up with the speed that the eavesdropped data is coming to the device. The Python script in Listing 5.1 utilizes the `pyscard` Python library to interface with the ACR122 reader, selects the applet by its ID and then repeatedly sends 128 bytes of data to the card in a loop with 500 iterations. On its output, the status words signaling result of the data exchange are shown.

As secondary tests, the phones' NFC capabilities were utilized for reading the various types of tags using NFC TagInfo application (v4.24.4) by NXP, the manufacturer of Mifare tags. For full functionality, the application requires an NXP NFC controller. Apart from performing the anticollision and reading the tag's UID, the application supports more advanced operations – reading whole Mifare Classic 1K memory, listing applications present on Mifare Desfire EV1 tag and reading NTAG213 memory.

This communication was then eavesdropped with Proxmark3 in conjunction with the real time Android client application to test eavesdropping on various types of traffic.

Unfortunately, the Nexus 5's NFC reader is unusable for the testing purposes as a reader, because its NFC field is very weak. The phone has difficulties regularly reading a tag placed directly on its back and with the Proxmark's

```
#!/usr/bin/env python3

from smartcard.System import readers
from smartcard.util import toHexString

AID = [0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x08, 0x09]
SELECT = [0x00, 0xA4, 0x04, 0x00, 0x08] + AID
ECHO = [0x80, 0x10, 0x00, 0x00, 0x80] + \
        [0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88] * 16

r = readers()
print(r)
conn = r[0].createConnection()
conn.connect()

data, sw1, sw2, = conn.transmit(SELECT)
print("SELECT: {:02x} {:02x}".format(sw1, sw2))

for i in range(500):
    data, sw1, sw2, = conn.transmit(ECHO)
    print("ECHO: {:2}: {:02x} {:02x}".format(i, sw1, sw2))
```

Listing 5.1: Python testing script for GemCombiXpresso

antenna placed between the two, the phone cannot read the tag at all. Secondly, it does not support Mifare Classic tags due to its Broadcomm NFC controller. Therefore the Nexus 5 was only used for testing of the Android real time client application.

While working on the thesis, I figured out that the eavesdropping is most reliable and performing without errors when the Proxmark's antenna is placed directly on the reader and the tag is circa half a centimeter away from the reader. I found that a small piece of cardboard (shown in Figure 5.1) works well for this purpose.

Utilizing the mentioned testing setup with a spacer, the Proxmark3 eavesdropping functionality itself worked reliably without any reception errors (parity errors, etc.).

Nexus 5 and Motorola E4P worked with the OTG adapter alone but Nexus 5X needed to be used in conjunction with a powered USB hub due to the phone probably not delivering enough power for Proxmark3 to function.

The connection to Proxmark3 through Android's USB API and subsequent real time eavesdropping worked without any problems with all three phones.

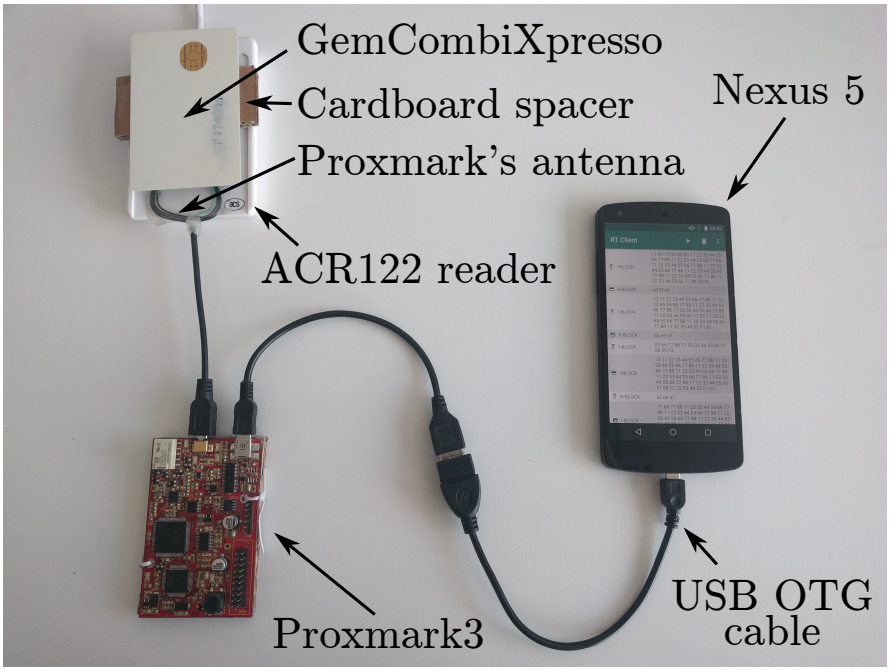

Figure 5.1: Testing setup with ACR122 as reader, Nexus 5 as client and GemCombiXpresso tag

5.2 Results

This section presents the results of previously proposed tests.

5.2.1 Stress test

As mentioned before, the main stress test of the implementation was the continuous communication between the echo applet installed on the GemCombiXpresso card and the ACR122 reader. The total number of messages sent by the reader is 500. The reader sends the 128 bytes message to the tag and the tag simply echoes the message back without any changes. This generates very large and fast traffic between the reader and the tag which allows for proper testing of robustness of the implementation.

The phones – N5, N5X and ME4P – are, in the scope of this test, used as client devices for Proxmark3 and the newly implemented Android application is utilized and tested as well.

One run of the test takes a total of is 73.5 seconds and the total amount of data generated is approximately 146.5 kB. Therefore the average communication speed is $\frac{146.5}{73.5} \approx 2$ kB/s. The test was run 10 times with each phone.

The Nexus 5X and Motorola E4P completed the test without a problem. The statistics of DMA buffer utilization are shown in Table 5.1.

Client phone	Min	Max	Avg
Nexus 5X	1581 B ≈ 10 %	3634 B ≈ 22 %	2170 B ≈ 13 %
Motorola E4P	4625 B ≈ 28 %	10339 B ≈ 63 %	7992 B ≈ 49 %

Table 5.1: Nexus 5X and Motorola E4P DMA buffer utilization statistics

Nexus 5X had the best buffer utilization out of the three phones. Motorola had overall higher utilization of the DMA buffer than N5X, but the maximum was 63 % so there is still a solid reserve.

Unfortunately, Nexus 5 on the other hand, failed 8 of the 10 tests by overflowing the DMA buffer. Only two of the tests were completed until the end. Table 5.2 shows statistics of the 8 failed attempts of Nexus 5.

	Min	Max	Avg
Time	12.2 s	37 s	23.1 s
Bytes sniffed	22564 B	72929 B	44489 B
Percent of data sniffed	15 %	50 %	30 %

Table 5.2: Nexus 5 failed test runs statistics

In the long test run, Nexus 5 could not keep up with the transmission speed and the buffer overflowed. Earliest buffer overflow occured 12.2 seconds after start of the test. Altough, it is very early in the test run, usage out of scope of this test probably would not require sniffing continuous communication for more than 5 seconds.

The N5's failed test runs are probably caused by the phone being quite outdated and not as powerful. There is probably also a number of other factors such as the Android operating system thread scheduling, number of processes running in the background, etc.

5.2.2 Secondary tests

The secondary test data exchanges that were performed using the NFC enabled phones and various tags are:

1. Mifare Classic 1K read with NXP TagInfo on Nexus 5X;
2. Mifare Classic 1K read with NXP TagInfo on Motorola E4 Plus;
3. Mifare Desfire EV1 read with NXP TagInfo on Nexus 5X;
4. Mifare Desfire EV1 read with NXP TagInfo on Motorola E4 Plus;
5. NTAG 213 read with NXP TagInfo on Nexus 5X;

6. NTAG 213 read with NXP TagInfo on Motorola E4 Plus.

Tests are further referenced by the numbers listed above. Parameters of each data exchange are summarized in Table 5.3 below. It shows that Motorola E4P reads the tags much slower than Nexus 5X, sometimes even twice as long.

#	Data transferred	Time	Transfer peed
1	2.7 kB	2.5 s	1.1 kB/s
2	2.7 kB	7.5 s	0.4 kB/s
3	2.2 kB	2.4 s	0.9 kB/s
4	2.2 kB	6.1 s	0.4 kB/s
5	1.3 kB	1.9 s	0.7 kB/s
6	1.3 kB	3.5 s	0.4 kB/s

Table 5.3: Characteristics of test data exchanges

Table 5.4 shows results of the individual tests, numbered from 1 to 6 as mentioned before. Each row represents a single client phone where the real time client application was tested. The values in the table represent maximum percentage of DMA buffer utilization (% out of the 16 kB) during the sniffing. Each test was performed 10 times and the DMA buffer utilization averaged.

Client	1	2	3	4	5	6
Nexus 5	9 %	75 %	14 %	35 %	7 %	64 %
Nexus 5X	—	14 %	—	37 %	—	16 %
Motorola E4P	10 %	—	15 %	—	11 %	—

Table 5.4: Percentages of DMA buffer utilization

The DMA buffer utilization in the other tests behaves contrary to expectations. It is lowest during the fastest transactions on the least powerful phone. My current guess is that it is dependent on the Android's scheduling mechanisms and thread priority management and that the data exchange is done before the data receiver thread has its priority lowered or is slowed down somehow else. Please note that this is just a speculation and I currently do not have any evidence to back it up and do not have enough time to test this claim further.

During the testing, I also found out that timestamps of the received data frames are sometimes incorrect. Unfortunately, due to not having enough time to investigate this issue, I was not able to debug and fix it. This finding fortunately is not a big problem for the thesis' goal as the application's current purpose revolves around showing only the received bytes of data (which are

not affected by this issue) and the timestamps are not utilized and not shown to the user.

5.3 Summary

The testing showed that the reception speed of the real time eavesdropped data is directly dependent on the phone's performance. Older phones, such as Nexus 5, might have occasional problems with the DMA buffer overflowing but not while eavesdropping short data exchanges (e.g. anticollision, single memory read, etc.). Newer and more powerful phones, such as Nexus 5X and Motorola E4 Plus, do not have any problems receiving the data continuously for a long time. In conclusion, the implemented solution for real time eavesdropping with Proxmark3 and a companion Android mobile device is functioning well.

Conclusion

The goals of the thesis were to research the workings of type A RFID communication in accordance with the ISO/IEC 14443 standard, review the principles of eavesdropping of such communication and explore existing solutions, devices and approaches that enable users to do so. Subsequently, aim of the practical part was to improve the eavesdropping capability of Proxmark3 and implement a companion Android client application to allow users to eavesdrop RFID communication in real time with immediate feedback on results of the process and basic decoding of the traffic without the need to carry a laptop.

The thesis explores the ISO/IEC 14443 standard and reviews the workings of type A RFID communication. It reviews the current state of existing solutions and devices that allow for eavesdropping of the RFID communication. The explored eavesdropping options include the usage of an oscilloscope, dedicated RFID or NFC protocol analyzers and open source hardware devices, mainly Proxmark3, which is subsequently the focus of practical part of the thesis.

The practical part successfully builds upon and improves the eavesdropping capabilities of the Proxmark3 device by adding an option to transfer the data to user's device in real time. Further, it includes an implementation of a new Android client application which acts as a client to the real time eavesdropping process, downloads the data from the Proxmark3 device in real time and performs basic decoding.

The goals outlined in the beginning were fulfilled and therefore the assignment as well. The thesis lays ground work for further development of real time eavesdropping Proxmark3 clients.

Bibliography

1. *BS ISO/IEC 14443-2:2016. Identification cards — Contactless integrated circuit cards — Proximity cards: Radio frequency power and signal interface*. The British Standards Institution, 2016. ISBN 978-0-580-86386-8.
2. FINKENZELLER, Klaus. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication*. Third Edition. Trans. by MÜLLER, Dörte. John Wiley & Sons, Ltd., 2010. ISBN 978-0-470-69506-7.
3. *BS ISO/IEC 14443-3:2016+A1:2016. Identification cards — Contactless integrated circuit cards — Proximity cards: Initialization and anticollision*. The British Standards Institution, 2016. ISBN 978-0-580-94402-4.
4. *MSO6104A Mixed Signal Oscilloscope: 1 GHz, 4 Analog Plus 16 Digital Channels [Discontinued]* [online]. Keysight Technologies, 2019 [visited on 2019-05-15]. Available from: `https://www.keysight.com/en/pdx-x202252-pn-MSO6104A/mixed-signal-oscilloscope-1-ghz-4-analog-plus-16-digital-channels`.
5. *RF-R 400-1: H-Field Probe 30 MHz up to 3 GHz* [online]. Langer EMV-Technik, 2019 [visited on 2019-05-05]. Available from: `https://www.langer-emv.de/en/product/rf-passive-30-mhz-3-ghz/35/rf-r-400-1-h-field-probe-30-mhz-up-to-3-ghz/13`.
6. *ACR122: USB NFC Reader* [online]. Advanced Card Systems Ltd., 2016 [visited on 2019-05-06]. Available from: `http://www.acr122.com/`.
7. *MIFARE DESFire EV1* [online]. NXP Semiconductors Austria GmbH Styria, 2019 [visited on 2019-05-06]. Available from: `https://www.mifare.net/en/products/chip-card-ics/mifare-desfire/mifare-desfire-ev1/`.

8. *DSOX4NFC NFC Automated PC-based Test Software and NFC Triggering for 4000 X-Series Oscilloscopes* [online]. Keysight Technologies, 2019 [visited on 2019-05-05]. Available from: https://www.keysight.com/en/pd-2712765-pn-DSOX4NFC/nfc-automated-pc-based-test-software-and-nfc-triggering-for-4000-x-series-oscilloscopes.
9. *NFC Testing Using an Oscilloscope Part 1: Benchtop R&D Measurements* [online]. Keysight Labs, 2016 [visited on 2019-04-23]. Available from: https://www.youtube.com/watch?v=RAw4i9aWl-o.
10. *NFC Testing Using an Oscilloscope Part 2: Automated Measurements* [online]. Keysight Labs, 2016 [visited on 2019-04-23]. Available from: https://www.youtube.com/watch?v=dCa6y6Mp-64.
11. *ComProbe® NFC Protocol Analyzer: NFC-A, NFC-B and NFC-F* [online]. 2019 [visited on 2019-04-24]. Available from: http://www.fte.com/products/NFC.aspx.
12. *NFC-A, NFC-B and NFC-F Protocol Analyzer datasheet* [online]. 2017 [visited on 2019-04-24]. Available from: http://www.fte.com/docs/datasheet_nfc.pdf.
13. DILLINGER, Markus; MADANI, Kambiz; ALONISTIOTI, Nancy. *Software Defined Radio: Architectures, Systems and Functions*. Wiley & Sons, 2003. ISBN 0-470-85164-3.
14. RONA, Jean-Christophe. *Sniffing and decoding NFC with a DVB-T stick (RTL-SDR) and GNURadio* [online]. 2017 [visited on 2019-05-05]. Available from: http://blog.rona.fr/post/2017/10/15/Sniffing-and-decoding-NFC-with-a-DVB-T-stick-rtl-sdr-and-GNURadio.
15. *ChameleonMini official repository* [online]. 2019 [visited on 2019-04-21]. Available from: https://github.com/emsec/ChameleonMini.
16. *Sniff Both Way 14443 by gypsophlia - Pull Request #180* [online]. 2018 [visited on 2019-05-06]. Available from: https://github.com/emsec/ChameleonMini/pull/180.
17. *HydraNFC 1.0 Specifications* [online]. 2019 [visited on 2019-05-07]. Available from: https://hydrabus.com/hydranfc-1-0-specifications/.
18. *HydraBus 1.0 Specifications* [online]. 2019 [visited on 2019-05-07]. Available from: https://hydrabus.com/hydrabus-1-0-specifications/.
19. *PROXMARK.org - radio frequency identification tool* [online]. 2016 [visited on 2019-05-07]. Available from: http://www.proxmark.org/.
20. *Proxmark3 RDV4.0 Dedicated Github* [online]. 2019 [visited on 2019-05-07]. Available from: https://github.com/RfidResearchGroup/proxmark3.
21. *A Test Instrument for HF/LF RFID* [online]. 2009 [visited on 2019-05-07]. Available from: http://cq.cx/proxmark3.pl.

22. *Proxmark3 official repository* [online]. 2019 [visited on 2019-04-21]. Available from: https://github.com/Proxmark/proxmark3.

23. *Proxmark3: Android* [online]. 2019 [visited on 2019-05-08]. Available from: https://github.com/Proxmark/proxmark3/wiki/android.

24. *Walrus: Make the most of your card cloning devices.* [online]. 2019 [visited on 2019-05-08]. Available from: https://walrus.app/.

25. *AndProx repository* [online]. 2019 [visited on 2019-05-08]. Available from: https://github.com/AndProx/AndProx.

26. *usb-serial-for-android repository* [online]. 2016 [visited on 2019-05-09]. Available from: https://github.com/mik3y/usb-serial-for-android.

27. *LG Nexus 5: Full phone specifications* [online]. GSMArena.com, 2013 [visited on 2019-05-13]. Available from: https://www.gsmarena.com/lg_nexus_5-5705.php.

28. *Nexus 5 Teardown* [online]. 2013 [visited on 2019-05-13]. Available from: https://www.ifixit.com/Teardown/Nexus+5+Teardown/19016.

29. *LG Nexus 5X: Full phone specifications* [online]. GSMArena.com, 2015 [visited on 2019-05-13]. Available from: https://www.gsmarena.com/lg_nexus_5x-7556.php.

30. *Nexus 5X Teardown* [online]. 2015 [visited on 2019-05-13]. Available from: https://www.ifixit.com/Teardown/Nexus+5X+Teardown/51318.

31. *Motorola Moto E4 Plus: Full phone specifications* [online]. GSMArena.com, 2017 [visited on 2019-05-13]. Available from: https://www.gsmarena.com/motorola_moto_e4_plus-8722.php.

Appendix A

Acronyms

ADC analog to digital converter

ASK amplitude shift keying

ATQA answer to request, type A

fc carrier frequency (13.56 MHz)

CTU Czech Technical University in Prague

DAB digital audio broadcasting

DVB-T digital video broadcasting — terrestrial

EOC end of communication

FDT frame delay time

FM frequency modulation

FPGA field-programmable gate array

GPL GNU General Public License

GPS global positioning system

HF high frequency

IEC International Electrotechnical Commission

ISO International Organization for Standardization

JNI Java native interface

LF low frequency

MCU microcontroller unit

ME4P Motorola E4 Plus

MSB most significant bit

N5 Nexus 5

N5X Nexus 5X

NFC near-field communication

OSH open-source hardware

OSS open-source software

OTG On-The-Go

PCB printed circuit board

PCD proximity coupling device

PICC proximity integrated circuit card

REQA request command, type A

RF radio frequency

RFID radio frequency identification

SOC start of communication

SSP synchronous serial port

UID unique identifier

Appendix B

Contents of enclosed CD

- `readme.txt` ..... CD contents description
- `apk`
  - `rtclient.apk` ..... built APK of rtclient Android application
- `src` ..... source files of the implementation
  - `proxmark3` ..... modified Proxmark3 firmware
  - `rtclient` ..... Android real time client application
  - `rtclient.py` ..... Python real time client application
  - `stress-test.py` ..... implementation stress test
- `text` ..... the thesis text directory
  - `BP_Havranek_Jan_2019.pdf` ..... the thesis text in PDF format
  - `BP_Havranek_Jan_2019.tex` ..... LaTeX source code of the thesis
  - `FITthesis.cls` ..... thesis LaTeX template
  - `materials` ..... directory with materials used in this thesis