FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Contactless card communication in Android
Student:	Nikola Karlíková
Supervisor:	Ing. Jiří Buček, Ph.D.
Study Programme:	Informatics
Study Branch:	Information Systems and Management
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2019/20

Instructions

Analyze the communication of contactless cards in the Android OS. Perform a search of existing software for communication tunneling (NFCProxy etc.).

Create your own application that will relay the communication between a contactless reader and card using two mobile phones. The phones will be connected using TCP/IP.

- Choose a suitable methodology for application development in Android.
- Create a component model of your system and document it using UML.
- Create sequence diagrams of interaction between the contactless card and the reader both in the normal case (without tunneling) and in the case of tunneling the communication using mobile phones.

The application will record the communication together with corresponding timestamps. Test your work using two mobile phones with a NFC interface. Evaluate delay added by relaying the communication. Based on your findings, evaluate costs and the impact of tunneling using mobile phones on contactless payment.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

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

Prague October 18, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Contactless card communication in Android

Nikola Karlíková

Department of Software Engineering
Supervisor: Ing. Jiří Buček, Ph.D

January 9, 2019

Acknowledgements

I want to thank my supervisor, Ing. Jiří Buček, for providing me with the sources necessary to complete this book and for expertise and valuable guidance and encouragement. I also want to thank the Department of Information Security for providing me with all the needed equipment.

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 accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on January 9, 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Karlíková, Nikola. *Contactless card communication in Android.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

V bakalářské práci jsem se soustředila na prozkoumání a implementaci přepojovacího útoku na bezkontaktní platby pomocí NFC technologie, internetového připojení a mobilních telefonů používající systém Android™. Úspěšně jsem implementovala řešení s použitím mobilních telefonů umožňujícím NFC připojení a služeb a rozhraní poskytnutých systémem Android bez použití dalších jiných služeb. Během testování jsem dosáhla 100% úspěšných útoků. Změřená zpoždění způsobená přenosem dat nebyla natolik významná, aby ovlivnila transakce. Na základě těchto výsledků mohu říci, že řešení je i přes jeho jednoduchost efektivní a může být použito a dále studováno jinými studenty a osobami se zájmem o téma útoků na bezkontaktní platby. CD příloha je připojena na konci této práce. Příloha obsahuje snímky obrazovek z mobilních telefonů a záznamy a skripty z testování.

Klíčová slova mobilní aplikace, Android, přepojovací útok, síťová komunikace, bezpečnost bezkontaktních karet, NFC technologie, emulace bezkontaktní platební karty, TCP/IP protokol, NFCProxy

Abstract

In the thesis, I focused on the examination and implementation of the solution for relay attack on contactless transactions using the NFC technology and public network with Android mobile devices. I successfully implemented the solution using the NFC enabled Android devices and services and interfaces provided by Android without other specific services. During the testing part, I accomplished 100% of successful attack runs. The measured delay caused by the data relay was not significant and did not affect the transaction. Based on this result, I can deduce, that this solution, despite its simplicity, is efficient and can be used and studied by any user interested in the topic of contactless payment attacks. Attachments are provided at the end of this thesis. CD attachment contain screenshots from the mobile devices and testing logs and scripts.

Keywords mobile application, Android, relay attack, network communication, security of contactless cards, NFC technology, emulation of contactless payment card, TCP/IP protocol, NFCProxy

Contents

Introduction **1**

1 Analysis **3**
1.1 Contactless smart cards 3
1.2 Smart cards communication 5
1.3 EMV specification 9
1.4 Android options 11
1.5 Attacks on contactless cards 12
1.6 TCP/IP Protocol 14
1.7 Pyscard 14
1.8 Chapter summary 15

2 Design and implementation **17**
2.1 Design of the implementation 17
2.2 Workflow 18
2.3 Preparing projects 19
2.4 Leech application 20
2.5 Ghost application 22

3 Testing **25**
3.1 Simulation 26
3.2 Real time testing 27
3.3 Delays 30

Conclusion **33**

Bibliography **35**

A Abbreviations **37**

B Content of the attached CD **39**

List of Figures

1.1 Sequence diagram card-terminal communication 5
1.2 Response status bytes example 7
1.3 Smart cards files structure 8

2.1 Design of the implementation 18
2.2 Component digram 19
2.3 Sequence digram 21

3.1 Ticket vending machine 28
3.2 Leech application screenshot 29

List of Tables

1.1 Command APDU structure 6
1.2 Command APDU CLA byte coding 7
1.3 Response APDU structure 7
1.4 SELECT command structure 9

3.1 APDU commands in simulation 27
3.2 Python script running results 27
3.3 Time delays in normal mode 31
3.4 Time delays in relay mode 32

Introduction

In recent decades, smart cards became integral part of our everyday lives. There are not so many people, who would not carry any smart card either it is payment or access card. By smart cards, I mean not only physical ones but also virtual cards uploaded in any mobile device. In these days for every operation system, there are quite many mobile applications that can simulate payment cards and allow user to pay without carrying the real card.

Smart cards working on the principle of NFC technology can be prone to intended abuse. Problem parts need to be captured so that cards are secured and less dangerous for their users. I will deal with one of the problem parts and try to implement real relay attack on the payment transactions that allows me to extend the data exchange distance and eavesdrop the communication between the card and the terminal. There are already implementations that deal with the relay attack that I will also discuss and describe.

I will introduce the way, how to relay the communication between card reader and smart card using two mobile devices, both using NFC technology. I will focus on the way, how to transfer the data between the mobile devices and I will also focus on how to track the communication protocol.

The information transfer can be divided into three parts. First is about communication between a card reader and first mobile device. This device will simulate the card and it will receive commands from the reader. The second part is redirecting the collected information to second device using TCP/IP network protocol. In the third part, the second device will play role of a card reader and will send the command to the connected smart card. The card will generate response. The response will be then redirected back to card reader the same way in reverted order. In this way, I can transfer the information from the card reader to the card and vice versa without touching the card to the reader physically. This solution should enable distance communication. The distance should be as long as area of the used local network. I will implement basic structure of relay attack for payment smart cards. I will provide the UML diagram and sequential diagram of the communication. During the

testing part, I will try the implementation in real payment transactions. I will also measure the delays caused by extension of the communication and observe the communication protocol.

I chose the topic because of its interesting impact on society. This topic and its problems can apply to anyone who uses the NFC technology in common life including me. The NFC technology is very interesting topic I wanted to interpenetrate into and discover its options. The other reason why I choose this topic was working with Android system. I like the Android platform and I previously developed applications for Android OS.

Result of my thesis can help users who are concerned about NFC technologies and contactless payment cards and want to resolve its critical parts. The result can also help students of security to know more about the NFC technologies and communication protocol between card and reader.

Chapter 1

Analysis

In this chapter, I will introduce the basic information about smart cards and communication protocols for data exchange between card readers and smart cards. Next, I will describe the NFC technology and Android options for using the NFC technology in mobile devices. In the end, I will describe already existing solutions for contactless card communication, data transfer via network protocol and I will briefly describe the Pyscard Python library that I will use in the testing part.

1.1 Contactless smart cards

Contactless cards are able to transmit power and data without any required electrical connection between the card and the card terminal. The method for this kind of data transfer is known as RFID (radio-frequency identification) systems. Contactless smart cards are in most of the cases operated passively. This means that all power comes from the terminal and cards do not have any source of power. The power transmission is based on the principle of a loosely coupled transformer working mostly on 135 kHz frequency. If a card appears in the area of terminal's electromagnetic field, part of the field goes through the coil of the card that causes generating of voltage. [1]

The aim of the standardization for the contactless cards was to generate standards that would enable the integration of several technologies for contactless power and data transmission, so contactless cards may also have other functional components. Nowadays there are three standards for three different reading ranges. These are ISO/IEC 10536, 14443 and 15693. Terminals in order to achieve interoperability should support all of these standards.

Applications of smart cards are very diverse and they are expanding. As an example, we can see smart cards in form of payment cards, personal IDs, health insurance cards or cards for public transport.

1.1.1 NFC technology

On September 2002, Sony and Philips announced that they would develop new near-filed communication technology, shortly NFC technology. NFC is a wireless connectivity based on RFID (Radio Frequency Identification) technology that enables two-way communication between electronic devices with a single tap. The aim for building NFC compliant devices was to connect smart cards and smartcard reader functions and provide comfortable way of communication method just by holding devices close to each other. NFC technology is suitable to Philip's Mifare and Sony's FeliCa, already existing contactless smart cards technologies. All NFC standards are based on RFID standards including ISO/IEC 14443. The specification for NFC is made by NFC Forum. [2]

NFC technology operates on 13.56 MHz frequency and data exchange rate is up to 424 kilobits/s. As it requires short range, typically 4 cm or less, it can prevent unintentional data leaks. The communication is half-duplex and any NFC device can be master (initiator) or slave (recipient), but after link establishment, the roles cannot reverse.

The main NFC unit is NFC tag. The tag is passive device powered by an NFC field. Simple tags offer just read and write operations. Tags that are more complex can perform math operations and can have cryptographic hardware to authenticate the access. Every tag has its own ID. This ID is low-level serial number and is used for identification and for preventing collisions. ID can be stable or generated randomly after every discovering. [3]

NFC technology operates in three modes [4]. Every mode implies different requirements on NFC devices.

- **Reader/writer mode** allows devices to read and write passive NFC tags. Smart posters are example of this mode.
- **P2P mode** allows NFC devices to exchange data with other NFC peers (Android Beam).
- In **card emulation mode** the NFC device itself acts as an NFC Card. The external reader can then access the emulated NFC card. The NFC device can also act as a reader for other contactless cards.

I am going to use the third option, card emulation. It is going to use it in both ways as a smart card simulator and as a contactless card reader.

The prerequisite for mobile payment is presence of the security element in the mobile device. There are three different options used to integrate security element. First option is to use smart card microcontroller permanently integrated in the mobile equipment. Second option is to use embedded security chip in external secure memory card. User can easily transfer the payment application to different device by swapping the memory card. Third option

uses the SIM or USIM. Payment application can be then hosted directly in SIM or USIM or in supplementary secure chip. [5]

It is necessary to have solution for loading and personalization of the tag providing payment application. This process must be supplied by trustworthy entity certified for payment systems. Sony and NXP (Phillips) developed different contactless card systems. While the FeliCa system provided by Sony is widely used in Japan and Asia, the Phillips's Mifare is used worldwide. The Mifare system is compatible with ISO/IEC 14443 Type A.

1.2 Smart cards communication

Interaction between the smart card and the terminal is based on half-duplex communication. The terminal always initiates communication. The card never sends data first, but it always sends data as a response to a command received from terminal. Specific response answers to specific command. This kind of relationship is called master-slave. After the smart card finishes its part, it goes to the low-power sleep mode and can be again awaken by another command from the terminal. [6]

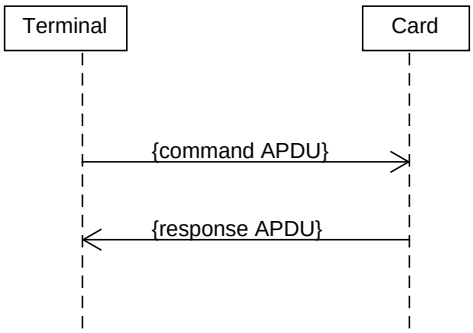

Figure 1.1: Sequence diagram showing communication between the terminal and the smart card

1.2.1 Logical channels

It is possible to access applications in smart cards via logical channels. When using logical channel, up to 19 applications in single card can exchange data with terminal. Two bits in the class byte of the APDU command can be used to address logical channels. Next, there is a special class byte with bit 8 set to zero and bit 7 set to one whose bits 1 to 4 address logical channels. More

sessions with associated application can run in parallel so each logical channel essentially represents a separate smart card.

1.2.2 Message structure: APDUs

APDU stands for application protocol data unit and it is used to exchange data between the card and the terminal. There are two kinds of APDU that can be distinguished. These are command APDUs and response APDUs. APDUs are transferred transparently without any modifications. APDUs based on ISO/IEC 7816-4 standard are independent of the transmission protocol, so the content of an APDU remains unchanged even if different transmission protocol is used.

1.2.3 Command APDU structure

As depicted in table 1.1, a command APDU consists of a header and a body. The header is mandatory and is composed of four bytes: the class byte CLA, the instruction byte INS that codes the actual command and the two parameters P1 and P2, used to provide more information for the command. The body is optional and may have variable length. The number of bytes in the data field is specified in L_c parameter and the expected length of response is specified in L_e parameter. If the value of L_e is 00, terminal expects maximum length of response.

Code	Name	Lenght	Description
CLA	Class byte	1	Instruction class
INS	Instruction byte	1	Code of instruction
P1	Parameter 1	1	Instruction parameter 1
P2	Parameter 2	1	Instruction parameter 2
L_c	Lenght	1 or 3	Number of data bytes
Data field	Data	$=L_c$	Data bytes
L_e	Lenght	≤ 3	Maximum of expected response bytes

Table 1.1: Command APDU structure (ISO/IEC 7816-4, 1997)

The class byte CLA is used to indicate if the command and the response comply with the specific part of ISO/IEC 7816 and when it is applicable, the format of messaging and the logical channel number. CLA byte according to ISO/IEC 7816 has 0X structure. For coding X see table 1.2. For the purpose of this thesis, I will not need any other CLA byte then 0x00.

xx - -	Secure messaging
0x - -	
00 - -	No SM
01 - -	Individual format of SM
1x - -	
10 - -	Command header not authenticated
11 - -	Command header authenticated
- - xx	Logical number channel

Table 1.2: Command APDU CLA byte coding (ISO/IEC 7816-4, 1997)

1.2.4 Response APDU structure

As depicted in table 1.3, APDU response consists of optional body and two mandatory bytes SW1 and SW2. These ending bytes represents the state of the receiving entity (card) after finishing its part (receiving the command and sending the response). For example, if the command is rejected with a response where SW1 = 6C, the SW2 byte contains the value expected in L_e byte. The return code 9000 means that the command was executed completely and successfully. Other examples of response APDU are showed in figure 1.2.

Body	Return code	Return code
Data field	SW1	SW2

Table 1.3: Response APDU structure (ISO/IEC 7816-4, 1997)

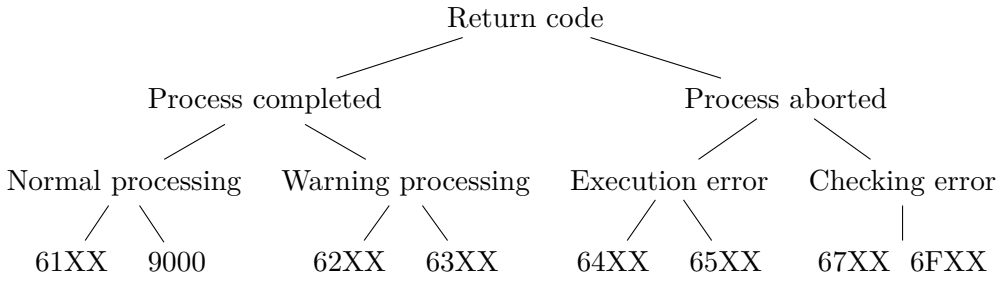

Figure 1.2: Response status bytes example (ISO/IEC 7816-4, 1997)

1.2.5 Application files

Smart cards are data storages, so the data are grouped in specific file structure. To select any file, card necessarily needs to receive the ACTIVATE FILE

command first. This command enables file selection. The opposite command is the DEACTIVATE FILE command that blocks the file. The file is then activated and can be accessed externally using the READ BINARY command. File can be deleted by the DELETE FILE command.

ISO standard 7816-4 supports two categories of files. The first one is called dedicated file (DF) and the second elementary file (EF). DFs act like a directory holding other DFs and EFs. The DF in the root is called master file (MF) and it is mandatory. The root directory is selected implicitly after the smart card is reset. [1] Figure 1.3 illustrates the logical file organization.

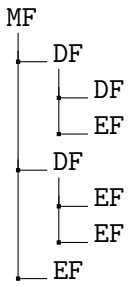

Figure 1.3: Smart cards file structure

The UICC specification introduces another special type of DF called application dedicated file (ADF) that is not located under MF. By accessing the ADF it is possible to access specific card application. Every application has its own identified called Application identifier (AID). ADF can be selected using the SELECT command with an AID. It is possible to add a new application by creating the appropriate DF because files holding data for particular application are grouped together under one specific DF.

1.2.6 SELECT command

I will introduce the structure of the most used SELECT command. This command sets the current active file through logical channel so other commands can refer to this file through this logical channel.

Table 3.1 shows the structure of the SELECT command. In my case, the parameters P1 and P2 will be set to 04 00. It means that I want to select DF directly based on its name, in our case AID, and return the first record of FCI. FCI means the file control information. It is the string of data bytes in response to the SELECT command. If the L_e byte is set to zero, all bytes corresponding to the command should be returned. [6]

As response to the SELECT command, some specific warning conditions can occur. In the case the SW1 = 62 with SW2=82 the selected file is invalid. With SW2=84 the FCI is not supported. Also error status bytes returned can be returned. For example, if SW1=6A and SW2=82, the file was not found.

CLA	As defined in table 1.2 - in our case '00'
INS	'A4'
P1	Selection controll
P2	Selection controll
L_c	Empty or lenght of subsequent data
Data	According to P1 and P2:
	File identifier
	Path from the MF
	Path from the current DF
	DF name
L_e	Empty or maximum length of data expected in response

Table 1.4: SELECT command structure (ISO/IEC 7816-4, 1997)

1.3 EMV specification

EMV is a family of multiple complex protocols with many variants. It is an international standard used by most of the payment smartcards all over the world. EMV is called after its three initiators: **Europay**, **Mastercard** and **Visa**. EMV has been success in reducing skimming fraud. Large part of EMV specification is public, but some parts are proprietary. Proper description of the whole EMV specification is beyond a scale of this thesis. The issue is its size. The specifications for EMV contactless cards have over 700 pages not including many additional options and parameters. It is hard to keep an overview of the way how it works. In this section, I will introduce the essentials of this specification and I will mostly focus on contactless smart cards.

EMV specifies commands and data formats designed for financial transactions between smart cards and terminals. It is built on other standards like ISO/IEC 7816 and ISO/IEC 14443. EMV transactions rely on symmetric cryptography, typically on 3DES or AES. EMV has many variants working upon several smart cards applications for example applications for purposes like withdrawals or internet banking. The backward compatibility between different protocol variants has been source of problems for example, the compatibility of contact cards with magnetic stripe with terminals using contactless communication protocol.

Three books in EMV contactless specifications are common for all contactless EMV cards. These are [7, 8, 10]. In the following paragraphs, I will describe the so-called MasterCard and Visa kernels. I will only describe these two kernels because they are commonly used in the Europe and I have access to these cards.

EMV contactless specification describes two modes of operations: **EMV mode** and **mag-stripe mode**. The mag-stripe mode is provided to support older infrastructures.

Contactless transactions are in many ways similar to contact transactions but there are two important differences.

- Online contactless transactions typically involve only one cryptogram as card response instead of two. The check of the cryptogram then takes place after the card is removed from the range of the terminal. This practice reduces the risk of the interruptions caused by removing the card too soon.

- In case the transaction is provided by NFC phones the additional method for verification of card holder identity was added.

To start the transaction, the reader has to select an application on the card. Smartcards contain a list of AIDs of all applications. This list is specified in the file 2PAY.SYS.DDF01 identified as PPSE (Visa Proximity Payment System Environment). The terminal selects the application with the highest priority and then it continues with the matching kernel specification.

1.3.1 MasterCard PayPass

PayPass is contactless implementation for MasterCard and is specified in Book C[9]. It supports two types of transactions: EMV mode and mag-stripe mode. Terminals are configured to support only one of the mentioned modes. Smartcards however can have combination of these supports. Cards labeled as MasterCard must support Mag-Stripe Mode and may support EMV Mode. Maestro-labeled cards must support EMV Mode and must not support Mag-Stripe Mode.

To initialize the transaction, the terminal starts by reading the PPSE and the card responds with the list of application with matching priorities. Reader choose the application with highest priority and sends the GET PROCESSING OPTIONS command. As response, the card sends avalible AIDs and the terminal then can use it to determine the transaction type and AFL (application file locator) containing all files the terminal can read. If both the card and the terminal support EMV mode, this transaction mode is used.

In next part of the transaction, after the initialization, card should be authenticated and cardholder verified. The terminal risk manager determines whether the transaction should be performed online or offline. During the cardholder verification, terminal sends the GENERATE AC command. Card then knows that the so-called On-Device Cardholder Verification must be performed and will perform the verification. The verification also contains PIN entering.

If the On-Device Cardholder Verification is not supported by the card or the terminal, either online PIN or signature verification must be performed. It is also indicated with the GENERATE AC command. When signature verification is needed, the receipt is printed and it should contain a signature line. As parameter with the GENERATE AC command at least one of the following need to be sent: amount, currency, country or date.[9]

The card responds with the cryptogram. There exists three kinds of cryptogram.

- Transaction Certificate (TC) represents the proof to the terminal that the transaction took place and was performed offline. Terminal sends it later to an issuer.
- Authorization Request Cryptogram (ARQC) is sent to an issuer for an online approval optionally with the PIN.
- Application Authentication Cryptogram (AAC) is provided instead of TC and ARQC in case the card or terminal refuse to complete the transaction.

1.4 Android options

"It can be expected that NFC devices will partially or fully supplant contactless cards in some applications. NFC devices such as mobile telephones can also be regarded as contactless smart cards with a different form factor. This is possible, because contactless terminals are not dependent on the format of ID-1 cards, but instead can exchange data with any device that supports ISO/IEC 14443 interface and is located within range of the terminal" (Wolfgang Rankl, 2010)

Android supports NFC with two packages. These are *android.nfc* and *android.nfc.tech*. One of the main classes, which I will use in realization part, is **NfcAdapter**. This class uses the helper method *getDefaultAdapter(Context)* to get the default NFC adapter. It works as NFC agent to initiate the communication.

Next class is Tag. **Tag** class is an immutable object that represents the state of a NFC tag in time of discovery. It can be directly queried for its ID via *getId()* method or set of technologies it contains via *getTechList()* method. The tag dispatch system in Android device is responsible for reading tags and their data and for starting the interested application. When the application wants to handle the scanned tag it must declare a filter for an intent. When the device discovers the tag, the tag dispatch system sends the intent to the most appropriate application that filters for it without asking the user. The tag object can be created and passed to a single activity as extra object in intent labeled as *NfcAdapter.EXTRA_TAG*.

Another class I use is **IsoDep**. Every tag supports NFC technologies for communication. **IsoDep** class provides access to ISO-DEP (ISO 14443-4) properties and I/O operations. The primary operation is *transceiver(byte[])* method. This method sends raw data to the tag and receives the response.[11]

1.4.1 Host-based card emulation (HCE)

NFC module usually consists of two parts. These are NFC Controller, which is responsible for communication, and secure element (SE). Secure element is responsible for encrypting and decrypting data. Secure element is embedded in SIM cards, SD cards or NFC chips.

When card emulation uses secure element, NFC controller routes the data directly to the secure element. The secure element performs the communication with the terminal, so the application is not involved.

Android 4.4 introduced method called Host-based card emulation (HCE) that does not involve secure elements. This allows applications to talk directly to the reader. The data is routed to the device CPU on which the application is running. In Card emulation mode, NFC simulates an RFID integrated circuit card with security module. Android class for card emulation is **HostApduService**. It can be extended in order to emulate NFC card inside Android application. Android 4.4 supports several protocols that are supported by any NFC devices. HCE provides cards emulation based on the ISO-DEP protocol and supports processing command APDU defined in ISO/IEC 7816-4.

ISO/IEC 7816 standard also defines the way, how to select applications based on the AID. For that reason, HCE needs to register one or several AIDs that the application wants to work with. AIDs can be split into separated groups and these groups can be associated with categories. Android 4.4 supports two categories: payment and other.

It can happen that more **HostApduService** components are installed on the Android device with same AID registered. Android has its own policy to resolve the conflicts depending on the AID group category. For category like payment the user can select default service in the device settings. Android 4.4 has setting called tap&pay where user set the default app for payment purposes. For category other, the device may always ask the user what service may be invoked.[5]

1.5 Attacks on contactless cards

The nature of the contactless smartcards prone to possible attacks. One of that attacks is the relay attack. During the relay attack, the attacker eavesdrops on the communication or secretly manipulate with the card. There is the possibility to use the NFC enabled phones to perform the relay attack.

The implementation of relay attack was firstly described and implemented in [12]. The author used only off-the-shelf NFC mobile phones Nokia 6131 and BlackBerry 9900. Non of the phones is running Android OS.

To use the card without cardholder knowing it, attacker needs to deliver enough power to the card to power it. For this, attacker is likely to use read-available hardware such as NFC mobile phones. What can appear as problem is timing. However, the time-out for terminals are often very long, so the relay does not need to be that fast to terminal not to time-out. Because there is no need to insert card directly to the terminal during the contactless payment, it is easier for attacker to interact with the transaction. There are two different attack scenarios:

- passive eavesdropping on the interaction between the card and the reader, and
- active interaction with the card without cardholder realizing.

One of the most notable relay attack implementation is NFCProxy. NFCProxy software is an open-source Android application. It enables users to proxy the transaction between the card and the reader. This solution extend the distance between the card and the card reader on the same principle as I am going to implement in my solution. In addition, it saves the transactions and can replay them. This software uses two NFC enabled mobile devices. One works in proxy mode and the second in relay mode. NFCProxy also uses same communication initialization. The device that reads the card behaves as server. It opens a port and it waits for connection from other side.

The drawback of this software is that the device in proxy mode needs have a special version of CyanogenMod intalled in order to replay credit cards[13]. CyanogenMod is open-source operation system for mobile devices. In these days, it has been discontinued and mostly replaced be new fork called Lineage OS[14]. In my implementation, I will not use any other operation system than official Android OS. If my implementation will be successful, it would mean that the solution is more prone to abuse because there is no need of knowledge of how to modify the phone firmware. I will be also able to find out if the current Android OS does not offer any resistence against relay attacks.

The NFCProxy software is user-friendly application. NFCProxy enables user to save the transactions, go throw the previous ones and see other details about cards. In my solution, I will focus mainly on the functionality and I will just write the communication on the screen.

There are also other solutions to explore the RFID technology. At the Shmoocon hacker conference, hacker Kristin Paget aimed to prove that RFID credit cards are easy to clone[16]. One of the efficient solutions, how to explore the RFID technology from the reader side is for example python solution called RFIDIOT[15].

There are some ideas how to prevent the relay attacks. The relaying of the messages takes some time. It would be possible to recognize the relay by the time. One solution would be to change the timeout for the reader. Because the first card EMV protocol responses are the same, they can be sent directly from the mobile phone that simulates the card. The timeout would not be efficient here. The only commands that need to be sent directly to the card are the crypto messages where card must send the encrypted data. Problem is that these steps require cards to do the crypto and can vary in time consumption. The time gaps caused by relay can differ in hundreds of milliseconds, same as average time consumption of the relay.

There are some ways, how to protect the card against attacks on contactless cards. Firstly, everyone should protect his or her own card. This means to ensure that the card is not kept in accessible pockets or bags where the thieves can reach them. Another method involves protecting cards by special shield. Aluminium foil used as food wrapping foil is enough to block almost any signal from reaching the card. There are also shields on the market that contain special metal alloys, which do the same job.

1.6 TCP/IP Protocol

TCP/IP protocol suite allows computers communicate with each other. It provides variety of networking services. TCP/IP is a combination of different protocols operating on four different layers.

The link layers includes the device driver and corresponding network interface. The network layer handles the movement of packets around the network using IP, ICMP and IGMP protocols. The third transport layer is responsible of the data flow between two hosts. There are two different protocols TCP and UDP. I will use the TCP protocol, because it is reliable and there is no need the ensure reliability in any other layer. The last application layer adds the details for the particular solution.[17]

1.7 Pyscard

For testing purposes, I will use the Python library Pyscard. Pyscard is a free software under the GNU Lesser General Public License. It is a Python module adding smart cards support to Python. I am going to use it to make a script to prepare set of commands for the card reader. In the testing section, I will show the components of Pyscard library and how I use it in python script to send commands to smart cards.[18]

1.8 Chapter summary

In the chapter I introduced the brief summary of all aspects that I will use further in the thesis. I discussed the basics of the smart cards including the content of standards and contactless cards communication along with the EMV standards. In this section, I also discussed the NFC technology and its use in the field of contactless card operations.

In the other section, I wrote about the similar existing solution called NFCProxy. I captured its disadvantages and how I will avoid them in my solution.

In the last sections, I described the Android options for smart cards emulation with all its main items. There was also brief description of TCP/IP protocol and how I will implement it using Java programming language. I provided brief introduction to Pyscard Python library, because I will use it in the testing chapter.

Chapter 2

Design and implementation

In this chapter, I will design and propose the way how to implement previously mentioned solution. I will provide UML diagrams that describes the component structure of the implementation.

2.1 Design of the implementation

My goal is to implement communication bridge between RFID smart cards and readers realized with two NFC enabled devices with standard Android operation system. The solution enables users to extend the length between the card and the reader. The connection between devices is based on TCP/IP protocol. Whole communication is captured so the protocols can be later observed and studied. The aim is to impement the project as simple and transparent as possible, so other students can understand it and use the parts of the code in their projects.

The communication starts and ends at the card reader side. The data from the reader are transferred to the card and back with two mobile devices through network. After the data from the reader are sent to the card and processed, the card sends the transferred response back to the reader the same way in opposite direction.

I split the implementation into two independent Android applications. I call the first one Ghost and the second one Leech. The so-called ghost-and-leech attacks are special class of attacks containing the relay attack.[19]

Through the Ghost application, I simulate the contactless card. For this, I use Android's Host Card Emulation service (HCE). This application also represents the client side of TCP/IP protocol. The second, Leech application waits for data from client. It represents the card reader and it is communicating with the real smart card.

The solution can be described in five steps. In the first one, the first mobile device acts like a smart card. It uses first application called Ghost. Ghost reads the data that were sent from the reader. In the second step, Ghost

Figure 2.1: Design of the implementation

application acts as a client in network communication. It connects to the second device that represents server and send the data to it using TCP/IP protocol. Second device uses other application called Leech and acts like a card reader. In the third part, the Leech application, receives data from the Ghost, client application. It forwards the received data to smart card using NFC connection and waits for the response. Once it receives the card response, the fourth part begins. The Leech device sends the response back to Ghost device that is already waiting for the server response. In the last step, the Ghost device, still acting like the card, sends the data back to card reader.

In the figure 2.2, I show the structure of the components. The diagram has two main components representing two main applications.

2.2 Workflow

The first step is to prepare both mobile devices so the communication and connection can be initiated properly. Users need to make sure that both phones are on same Wi-Fi or other LAN. When the Leech application is running, user can place the smart card close to the NFC chip in device. The card should be loaded and connected and application should announce it on its screen along with the IP address and port on which the application is waiting. User needs to put this IP address to Ghost applications input field so it can later connect to the Leech application. After all prerequisites are fullfilled, user can start the transaction by putting the device with the Ghost application to the terminal. The communication should be transferred properly and logged on the screen of both devices without any other interacting from the user.

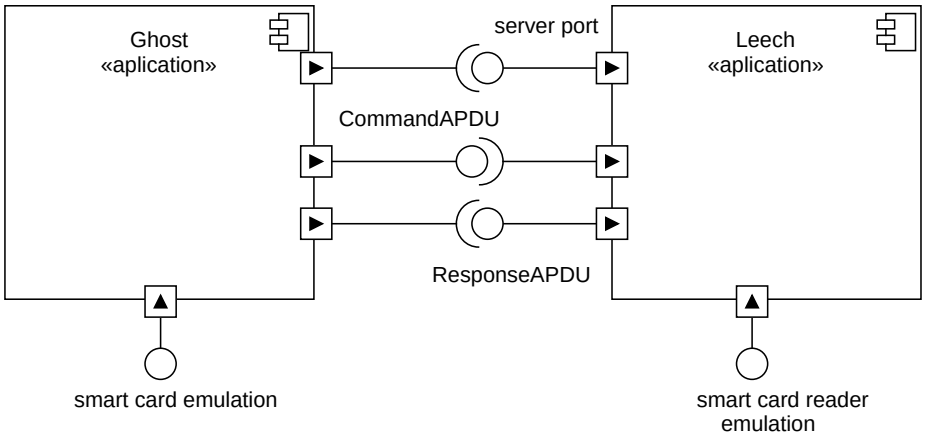

Figure 2.2: Component digram

2.3 Preparing projects

Because the Leech and the Ghost applications are using NFC technology, I need to set up few things. First, I need to manage Android Manifest. The manifest file describes basic information about the application required by Android build tools and operation system. In order to use NFC in the device I need to add NFC permissions and define that the application will use NFC features such as NFC adapter. Application's communication is realized by network, so I also need to add permissions for using internet and Wi-Fi.

```
<!-- HCE service declaration for Client application-->
<uses-feature android:name="android.hardware.nfc.hce"
    android:required="true"/>
<!-- NFC permissions -->
<uses-permission android:name="android.permission.NFC"/>
<uses-feature android:name="android.hardware.nfc"/>
<!-- Internet permissions -->
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
```

Both applications use same tools for handling the APDUs and logging. For handling APDUs, I created tools class called **Utils**. I am using this class for converting the APDU byte arrays to strings and vice versa. The commands from the card reader that need to be displayed on the mobile devices screen are in form of bytes array so I wanted to have full control over what is displayed and I wanted guarantee good readability everytime.

Methods for logging are same for both applications. The difference is just in the way how they are called. For writing logs on the screen, I need to edit the main activity layout so it contains **TextView** element. This user interface element displays text on the screen. Both these elements in applications are the same. I made them scrollable so user can see all data. Every time the method for logging is invoked with specific message to be logged, it writes the message both in the Logcat and on the screen. Along with the message, I am putting also a time to the log for further testing purposes. For screen logging I call *append()* method on the existing object representing the TextView element in the code. If I would try to call previous method from the running thread or other class then the activity we want to call it from, it would not work. In this case I need to call *MainActivity.this.runOnUiThread()* method. As parameter I pass the Runnable object with overridden abstract *run()* method. This practice helps me to do actions on the UI thread of the main class.

```
MainActivity.this.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    logInfo.append(msg + "\n");
                }
            });
```

I split the implementation into two independent applications as I describe it in design section. Realization follows same labels and names as design. I called the applications Leech and Ghost. I will describe the applications and their parts in the following sections. All the workflow and interaction of the applications is depicted in figure 2.3. All messages are synchronous. It means that all wait for response. Card and the terminal can be represented as unknown source in this case because we do not need to know their internal behavior. In this case, it is enough to use them as invocation source.

2.4 Leech application

The Leech application represents the card reader and the server side of the communication. The device with Leech application ensures that the data are transfered to the smart card. Whole implementation is provided in the Android's main class called **MainActivity**. I am using only standard Java and Android services and classes. First thing I need to do is to edit the application manifest so it filters for the NFC intents. When I want my application to filter for the ACTION_TECH_DISCOVERED intent I must create and XML resource file to specify the technologies that I want my application to support. Technologies are listed within a tech-list set. I add the file also in AndroidManifest file in the metadata element.

In MainActivity, I override the *onNewIntent()* method. Every time the device detects the smart card, this method is invoked and the intent and its

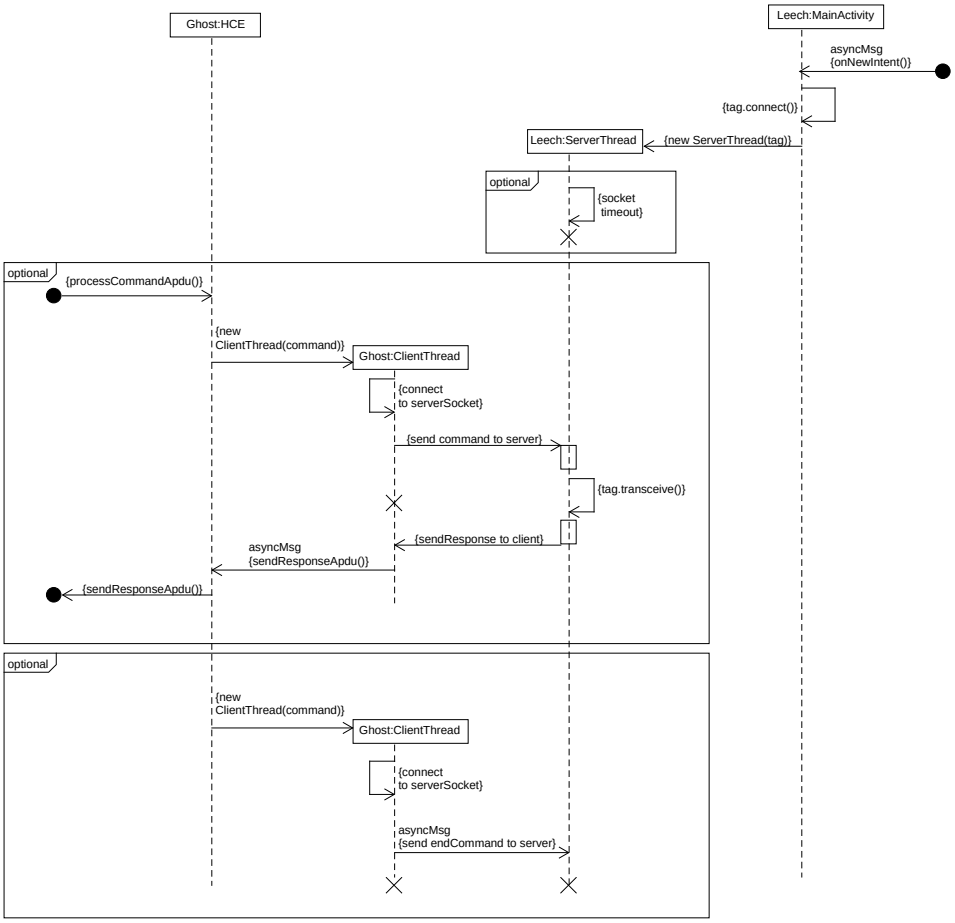

Figure 2.3: Sequence digram

```
<intent-filter>
    <action android:name="android.nfc.action.TECH_DISCOVERED" />
</intent-filter>
<meta-data
    android:name="android.nfc.action.TECH_DISCOVERED"
    android:resource="@xml/nfc_tech_filter" />
```

data is passed as an attribute. Intent is a describtion of an operation that is pass on to the application and that is to be performed. The intent is usually paired with extra data. I check if the new intent has EXTRA_TAG data. This data is defined in NcfAdapter class. I use this extra data to initialize the **Tag** object and start new thread for network communication using this tag. It can happened that the card is read more times so I declared boolean

flag to indicate whether the connection between card and device is already established or not.

New thread is created using nested class called **ServerThread** that represents the runnable object. When I create the new thread and start it, I use the tag to connect the device with the card using IsoDep technology. After successful connection, I create the TCP/IP server socket and start the main client-server communication that is running in a while loop. In this loop, the server socket is waiting for client socket to be accepted. Next, I initialize the java objects for reading from input stream and writing to output stream of the client socket. For reading, I am using **BufferedReader** java object created from clients input stream. Writing is provided by **PrintWriter** object that prints formatted data to output stream. In this moment, I am ready to read the client's message. I need to check if it is not null or if it is not the ending message. I created the ending message used by both sides to indicate the end of the communication so the server knows when to close the connection. If the message equals the end command, I break the loop and close the server socket. Otherwise I convert the message to a byte array using the custom tools described in previous section and I send it to the card using IsoDep method *transceive()*. The card sends back the response and I can catch it as a return value of the IsoDep method. I immediately send the response back to client. In this time, the loop reaches its end and it waits for another client socket to accept and read its message again.

```
Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
IsoDep isoDep = IsoDep.get(tag);
isoDep.connect();
byte[] response =
    isoDep.transceive(Utils.hexStringToByteArray(clientMsg));
```

2.5 Ghost application

Ghost application represents the client side of TCP/IP communication. While the Leech application is waiting on the other side to connect, this side needs to get the right IP address of the server. For that reason, a created input field so user must first enter the server IP address in order to successfully connect and start communication. After IP address is inserted, it is validated before properly saved. This all happens in the **MainActivity** class.

For card simulation in Android, I am using the Host-based Card emulation. I need to declare the HCE supported service that I will use in **AndroidManifest.xml** file. Basic service declaration is used as usual and additional pieces are added. With permission parameter, I set the permission the application must have in order to launch the service. Along with the exported parameter set to true, the parameters mean that only external applications that hold the specific declared permission can bind to my service.

Intent-filter tag specifies the type of intent the component can respond and declare the component capabilities. In this case, I use the BIND_NFC_SERVICE in the mandatory element action to tell the platform that it is this service that is implementing the **HostApduService** interface.

```
<service android:name=".CardEmulationService"
    android:exported="true"
    android:permission="android.permission.BIND_NFC_SERVICE">
    <!-- wher reader is trying to read a card, it fires the
        HOST_APDU_SERVICE action -->
    <intent-filter>
        <action
            android:name="android.nfc.cardemulation.action.HOST_APDU_SERVICE"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
    <!-- to know which service to call based on AID the reader is
        trying to communicate with -->
    <meta-data
        android:name="android.nfc.cardemulation.host_apdu_service"
        android:resource="@xml/aid_list"/>
</service>
```

I used also meta-data tag to register group of requested AIDs. The tag is pointing to the XML resource file **aid_list.xml**. The *requireDeviceUnlock* parameter is set to false, so the device does not need to be unlocked before the service is invoked. In this file, I introduced only one aid group with category set to payment. In this group, I declared three AIDs, one belongs to my testing card and other two for Visa and MasterCard smart cards.

```
<host-apdu-service
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:description="@string/service_name"
    android:requireDeviceUnlock="false">
    <aid-group
        android:category="payment"
        android:description="@string/aid">
        <!-- test card-->
        <aid-filter android:name="A0000000018434D" />
        <!-- visa and mastercard-->
        <aid-filter android:name="A0000000031010" />
        <aid-filter android:name="A0000000041010" />
    </aid-group>
</host-apdu-service>
```

First thing to do to start using the **HostApduService** is to extend it.[5] For this, I introduced the class called **CardEmulationService**. HostApduService has two abstract method that needs to be overridden. *Proccess-*

CommandApdu() method is invoked every time the reader sends an APDU. I can send the response by returning the bytes directly from the same method or I can use the separate *sendResponseApdu()* method. I use the second option because I cannot ensure the response will be computed on the same main thread.

Android will keep sending commands from the reader to my service until either the NFC reader sends another SELECT command that would resolve in calling the service again, or the link between reader and my device is broken. Both cases invoke the second abstract method *onDeactivate()* with the reason as parameter.

In the *processCommandApdu()* method, I am catching the command APDUs sent from the reader as an array of bytes with every invoke. Then I use my tools to convert the received bytes array to string. I use the string form of command APDU as user-friendly form to be shown in UI. After the command is converted, I call the *startCommunication()* method to initialize the communication between both applications. Ghosts side represents the client part of the TCP/IP protocol communication. The connection is implemented in two nested threads. In the first, I create the socket and connect it to the server port. I get the server IP address from MainActivity where I got it from user input. The port is hard-coded at the beginning of the service. When the client is connected, I create and call the second thread for data exchange. In this thread, I initialize the java objects for reading and writing same way as I described it in previous chapter. This means that I am using **BufferedReader** object for reading and **PrintWriter** object for writing. I send the received command with the output object to server and then wait until the server sends back the response. If the response is valid I convert it to an array of bytes and send it back to the reader with help of the HostApduService method called *sendResponseApdu()*. After response is successfully sent I close the client socket and I close both threads.

In case the connection between the card and the reader is disconnected, the second abstract method *onDeactivate()* is called. I am using this method to notify the second device about the end so it can end the connection and close the server socket. To do this, I declared special ending message and send it to the server. There is no need to make other steps in order to end the communication because the service methods are invoked by external events such as receiving the reader command or disconnecting from the reader.

Chapter 3

Testing

In this chapter, I will describe how I tested the solution described and implemented in Realization chapter. I used two ways of how to test the applications. In the first, I simulated the card reader using Python script. In the second, I used real card terminals and tried paying using the implemented relay attack. I will show the results from both testing parts and I will measure the delays coused by the relay. Real time testing enabled me to study the communication protocol between the card and the terminal and I will describe it in this section too.

For testing purposes, I used two mobile devices of type Motorola E4 Plus. Both devices need to be connected on the same LAN. Where it was possible, I used local Wi-Fi. Otherwise, I used third device to connect on data internet and created hotspot. On the device with Ghost application, I needed to set this application as default for payment. This was done in settings of the device in tap&pay section. Otherwise, the device would keep looking for default application called Google pay.

Other equipment I needed were a card reader and a smart card. I was using USB card reader of type SCL 3711 from SCM Microsystems Inc. To use it I inserted it into USB port of my computer. The programs and scripts for card reader simulation are automatically connected. For testing, I could use any smart cards with at least one payment application. During the testing, I used my own real MasterCard debit card to show the use of the solution in real payment transactions.

During the whole testing, when it was possible I counted time delays caused by longest distance the commands and responses must reach. I also counted the number of successful and unsuccessful attempts so I could later measure the success rate of my solution.

3.1 Simulation

First way how to test the implemented solution is to simulate the card reader. For this, I created Python script using Pyscard library.

This framework is for building smart card aware applications in Python. These applications connect with card through the smart card reader and then interacts with the card by sending set of APDUs. It first selects a smart card reader and then connects with the card. In the code below, I show the basics of the testing script.

```
from smartcard.CardType import AnyCardType
from smartcard.CardRequest import CardRequest

cardrequest = CardRequest( timeout=1, cardType=AnyCardType() )
cardservice = cardrequest.waitforcard()
cardservice.connection.connect()
data, sw1, sw2 = cardservice.connection.transmit(SELECT_COMMAND)
cardservice.connection.getReader()
```

This part of the code creates a request for specific card and then waits until the card matching the request is inserted. I do not make difference between cards so I set the card type to *AnyCardType*. For reading the card, I use time-out of five seconds. Function *waitforcard()* returns the card service or it expires because of the set time out. Once a matching card is introduced, I am able to send commands to the card with *transmit()* function. This function returns the card response that I parse into separate variables to differ the data bytes and status bytes. In case I want to see the name of the reader, I call function *getReader()*.

During testing, I used functional MasterCard debit card. First, I run the script using just the smart card. Then I used the devices to try the implemented relay. Because I know the AID of MasterCard cards, I was able to build the SELECT command successfully. Using the script, I created also other APDU commands. I did not expect any concrete responses except response to the SELECT command. Because I created the SELECT command upon real and matching AID, I expected successful transmission so the response should have contained OK status 9000. In other cases, I said that transmission would be successful if the responses would match the responses I get from the communication with the card without the relay attack. All commands created can be seen in Table 3.1. As result of this testing, I did not expect any differences between the results. The implemented solution should not modify the commands and responses. The devices are just forwarding the data without touching it.

In table 3.2, there are results of the script running first with card without any other device and then using the relay. Both are the same so we can see that the implementation do not modify neither the commands nor the responses.

SELECT	[0x00, 0xA4, 0x04, 0x00, 0x07, 0xA0, 0x00, 0x00, 0x00, 0x04, 0x10, 0x10]
GET PROCESSSING OPTIONS	[0x80, 0xA8, 0x00, 0x00, 0x01, 0x83, 0x00]
INTERNAL AUTENTICATE	[0x00, 0x88, 0x00, 0x00, 0x01, 0x01, 0x00]
VELOCITY CHECKING ATC	[0x80, 0xCA, 0x9F, 0x36, 0x00]
VELOCITY CHECKING	[0x80, 0xCA, 0x9F, 0x13, 0x00]

Table 3.1: APDU commands used in simulation

data: 6F 4A 84 07 A0 00 00 00 04 10 10 A5 3f 50 ...	data: 6F 4A 84 07 A0 00 00 00 04 10 10 A5 3f 50 ...
status bytes: 90 0	status bytes: 90 00
data: 70 75 9F 6C 02 00 01 9F 62 06 00 00 00 00 ...	data: 70 75 9F 6C 02 00 01 9F 62 06 00 00 00 00 ...
status bytes: 90 0	status bytes: 90 00
data: 77 16 82 02 19 80 94 10 08 01 01 00 10 01 ...	data: 77 16 82 02 19 80 94 10 08 01 01 00 10 01 ...
status bytes: 90 0	status bytes: 90 00
data: - status bytes: 6A 81	data: - status bytes: 6A 81
data: - status bytes: 69 85	data: - status bytes: 69 85
data: - status bytes: 6A 88	data: - status bytes: 6A 88
data: - status bytes: 6A 88	data: - status bytes: 6A 88

Table 3.2: Python script simulation results

This testing method has many benefits. First, I am able to run the script repeatedly and see the results from the reader side. It allows me to run the script separately with just the card without any other impact on the communication and then using the devices to extend the communication.

The disadvantage of this method is that I am not able to simulate the real communication protocol between the card reader and the payment card. It would require deeper study of the EMV communication protocol and it is not in the scope of this thesis.

3.2 Real time testing

Because I made the relay attack solution to eavesdrop and extend the payment transaction, it was necessary to test it in the real-time payment transactions. Nowadays, it is possible to pay with contactless payment card almost everywhere. For my testing purpose, I chose first embedded payment terminals for

example in vending machines and then I tried the implementation on other places.

Figure 3.1: Ticket vending machine used during the testing

3.2.1 Testing process

As first, I tested the implementation on the vending machines and ticket vending machines with embedded contactless payment terminals because the terminal is set up for the payment automatically and I am able to connect both phones to my computer in order to correct possible errors during the whole transaction. After that, I tried other payment terminals in shops etc.

From the first attempts, only 20% of approximately ten transactions were successful. The most common errors were *transaction* errors or *unsupported card* errors and all appeared on the terminal side. The communication was not even initialized so the Ghost application was not invoked. Because of that, I did not have any logs and I could not observe the transaction. During one transaction, the Ghost application was called and terminal sent one command to the card. Card responded with error status code 6A82 and the transaction was aborted with same error as in previous cases. Error code 6A82 means that the file, which terminal wanted to access, was not found.

The bug was in the Ghost application. In the file that specifies the group of AIDs the application was simulating were just two AIDs for Visa and MasterCard types of cards. The mobile phone is than able to connect to the terminal as Visa or MasterCard smart card without the terminal knowing it is not real card. The problem was that the terminal did not know beforehand what type of card had got into its area so it tried various kinds of AIDs with the SELECT command and waited for OK response. In my cases, most of the

terminals started with AID that was not specified in the previously mentioned file and because the implementation did not supported that types of cards, the terminal aborted the transaction with error message. In the successful case, terminal started with MasterCard AID, so the card responded with OK status code (9000) and the transaction then continued normally. In other case, the terminal sent the SELECT command for Visa. Card could not find file with Visa identifier, so it returned error code.

The solution was to add PPSE AID (325041592E5359532E4444463031) to group of AIDs. The vendor of this AID is Visa International. In response to this command, card sends list of all availible AIDs it has. Terminal then can choose from this list the most appropriate AID and access it.

After this correction, 100% of seven attempt were successful. No other problems occurred during further testing transactions.

3.2.2 EMV protocol observation

During the testing, I encountered two very similar protocols. They differ only in the first command and the way they discover the proper AID. In the figure 3.2, I show the example of the Leech application screenshot. The figure shows, how the application reads the card, initializes the server port and how it forwards the messages to connected card.

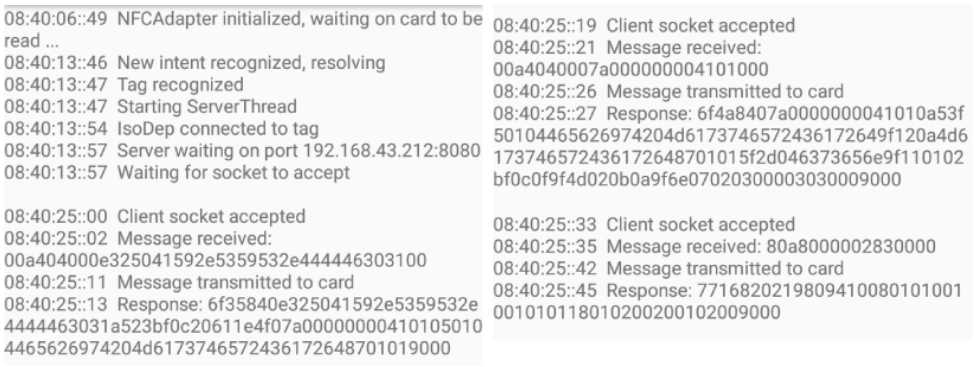

Figure 3.2: Example of the Leech application screenshot

The first discovered payment protocol sents as first the SELECT command with MasterCard AID. Because my card responded with OK status (9000) the terminal knew it is communicating with MasterCard type of card and started the transaction. It is possible that if I would use different card than MasterCard the protocol would continue with another SELECT command with other supported AIDs until it would find the appropriate one.

The difference in the first command in the second protocol is just in the AID. The first command is also the SELECT command but it is using PPSE AID. As response, card sent a list of AIDs of all availible payment applications.

In my case, the response contained just MasterCard AID. The terminal then could choose from the supported AIDs the one with the highest priority.

Other commands in both protocols are the same. As next, the terminal sent the SELECT command with appropriate AID it choose in previous step. It means that the first protocol sends the same command twice. First to select the appropriate AID and then to activate the corresponding file. After successful SELECT command, the application file is activated and accessible for the reader. Third command was the GET PROCESSING OPTIONS command to initiate the transaction process. This command sends the data for processing the transaction. This data could include the transaction amount, for example. As a response, the card sent data containing AIP, which describes the functions supported by the smart card, and AFL, application file locator.

Next five commands were all READ RECORDS commands. They represent the authentication process. In my case, they differed only in combination of the parameters P1 and P2. First parameter (P1) specifies record number and second parameter (P2) controls the reference. As the last, terminal sent GENERATE AC command requiring Application Cryptograms (ACs) from the card.

There are many variants of EMV protocol present on the market. It is very possible that there are some cases where my solution may fail and the success rate may be less than 100%.

3.3 Delays

Because the relay is time consuming, it is reasonable to measure the time delays. My implementation allows to measure the delays because every log contains time stamp. I am then able to measure the delays by comparing the time when the message was sent from the first device and time when the message was received by the second device.

Second way of how to measure the delays is to compare times of single operation. During the testing with simulated reader, I captured the times of single operations and I compared these times of transaction between the card and the terminal with transaction using the relay. Because now I know the exact communication protocol from previous testing, I am able to reproduce it using own reader. Despite the fact that time length of the real payment transaction can differ from the simulated one, the relay logic and delays are the same because it is not dependent on the used reader and card but just on the network. Therefore, total delay will not differ and it is a good practice to capture the time delays. Tables 3.3 and 3.4 below show examples of the measurment. The table 3.3 shows the transaction without the relay and table 3.4 shows transaction with relay attack using data network. Numbers mean the seconds of the timestamp of the action.

First, I measured a few transactions without the relay. The average time gap between sending a command and receiving a response was 68.776 ms. Maximum gap was 318.530 ms long. Average length of the whole transaction was 632.697 ms.

command	command timestamp (s)	response timestamp (s)	time gap (s)
SELECT	38.5511	38.5941	0.04300
SELECT	38.5961	38.6361	0.0400
GET PROCESSING OPTIONS	38.6371	38.6991	0.0620
READ RECORDS	38.7001	38.7370	0.0369
READ RECORDS	38.7390	38.7910	0.0520
READ RECORDS	38.7940	38.8090	0.0150
READ RECORDS	38.8090	38.8210	0.0120
READ RECORDS	38.8220	38.8610	0.0381
GENERATE AC	38.8630	39.1798	0.3168
		Min gap	0.0120 ms
		Max gap	0.3168 ms
		Avegare gap	0.0684 ms
		Total lenght	0.6287 ms

Table 3.3: Delays captured during simulation of the transaction protocol without using the relay attack

Next, I measured transactions using relay on two different networks. First network was public Wi-Fi. The average length of the transaction was 6.470 s with maximum gap of 4.566 s and average gap of 717.692 ms. As second, I used internet data connection using regular mobile phone and sharing the network via hotspot. This solution was much faster than the previous one. The average length of the transaction was 2.629 s. Maximum gap was just 723.138 ms and average gap was 290.933 ms.

As I can see, the amount of the total delay time depends on the network speed. I am able to reach satisfying results using fast connection. The time difference between data exchange without the relay is up to 300 ms. This time can be covered by the fast relay so no one would be able to track it. The difference between my testing runs are around 2 seconds and using even faster connection can reduce this difference. Payment terminals have long timeouts so even these differences should not be problem.

command	command timestamp (s)	response timestamp (s)	time gap (s)
SELECT	18.4960	18.8519	0.3559
SELECT	18.8530	19.1733	0.3203
GET PROCESSING OPTIONS	19.1743	19.4805	0.3062
READ RECORDS	19.4815	19.7052	0.2237
READ RECORDS	19.7072	20.0424	0.3352
READ RECORDS	20.0440	20.2556	0.2116
READ RECORDS	20.2556	20.3662	0.1106
READ RECORDS	20.3672	20.5605	0.1933
GENERATE AC	20.5635	21.1394	0.5759
		Min gap	0.1106 ms
		Max gap	0.5759 ms
		Avegare gap	0.2925 ms
		Total lenght	2.6434 ms

Table 3.4: Delays captured during simulation of the transaction protocol using the relay attack

Conclusion

The aim of the thesis was to study the problem of contactless smart cards in the contactless payment transactions and to design and implement working solution for relay attack using two NFC enabled mobile devices.

In the analytic section of the thesis, I studied the main parts of contactless smart cards and payment communication protocols and described them in details. It was difficult to learn all the main parts in the topic of contactless payment transactions because all implementations use several standards and the specification is extensive. I also described all technical details of the contactless communication. I pointed to problem parts of the contactless transactions and I showed how the contactless smart cards are prone to attacks.

Based on my own design, I implemented functional solution for relaying the payment transaction. The solution is transparent and can be well read and it allows users to successfully perform the relay attack and observe the communication protocol between the terminal and the smart card. I was able to try the proccess of the application development including analytic part and design with UML diagrams.

The final solution worked successfully in real time transactions without any problems. I tried to pay using the implemented relay attack in several places, including vending machines and regular payment terminals in various kinds of shops. I performed successful tests and based on the results I measured time delays caused by the relay. I found out that even though the total time of the transaction went up in the worst case by few seconds, the transaction was not interupted and finished successfully.

Af far as I tried the solution, I found some impacts of the relay attack on real time transactions. Because the implementation and use of the relay attack is not so difficult, it can be used by thieves who can perform it on the card of different person without anyone knowing it. It is enough to get close to the card so it is possible to connect to it with phone and ensure the communication with the second phone by any transaction channel. The

chance of the abuse is reduced by the contactless transaction money amount limit for entering the PIN. The cardholder can protect his cards with special shields or just by ensuring the card is not kept in accessible places.

In the future, the implementation can be extended to include more additional user-friendly features like saving and repeating the transaction or automatic delays capturing.

Bibliography

[1] RANKL, Wolfgang a Wolfgang EFFING. Smart Card Handbook. 4. Chichester, West Sussex, U.K: Wiley, 2010. ISBN 978-0-470-74367-6.

[2] Sony: Press releases. Sony: Press releases [online]. Netherlands/-Japan, 2002, 2002 [cit. 2019-01-06]. Avalible at: `https://www.sony.net/SonyInfo/News/Press_Archive/200209/02-0905E/`

[3] NFC Forum: What is NFC? About the technology. NFC Forum [online]. 2019 [cit. 2019-01-06]. Avalible at: `https://nfc-forum.org/`

[4] Nick Pelly and Jeff Hamilton. How to NFC Presentation presented at: Google I/O 2011.; 2011 May 10-11; USA (California) [online]. Avalible at: `https://www.youtube.com/watch?v=49L7z3rxz4Q`

[5] Android Developers: Host-based card emulation overview. Android Developers: Host-based card emulation overview [online]. 2018, 2018 [cit. 2019-01-06]. Availible at: `https://developer.android.com/guide/topics/connectivity/nfc/hce`

[6] SN ISO/IEC 7816-4. Informaní technologie - Identifikaní karty = Karta s integrovanými obvody a kontakty - ást 4: Mezioborové píkazy pro výmnu. 2. eský normalizaní institut, 1997.

[7] EMVCo. Book A: Architecture and General Requirements v2.6. March

[8] EMVCo. Book B: Entry Point Specifications v2.5. EMV Contactless Specifications for Payment Systems. March 2015.

[9] EMVCo. Book C-2 Kernel 2 Specification v2.5. EMV Contactless Specifications for Payment Systems. March 2015.

[10] EMVCo. Book D: Contactless Communication Protocol v2.6. EMV Contactless Specifications for Payment Systems. March 2016.

[11] Android Developers: Near field communication. Android Developers: Near field communication [online]. 2018, 2018 [cit. 2019-01-06]. Availible at: https://developer.android.com/guide/topics/connectivity/nfc/

[12] FRANCIS, Lishoy, Gerhard HANCKE, Keith MAYES a Konstantinos MARKANTONAKIS. Practical Relay Attack on Contactless Transactions by Using NFC Mobile Phones [online]. United Kingdom, 2005 [cit. 2019-01-06]. Availible at: https://eprint.iacr.org/2011/618.pdf.RoyalHollowayUniversityofLondon.

[13] SourceForge: NFCProxy. SourceForge: NFCProxy [online]. 2013, 2013 [cit. 2019-01-06]. Availible at: https://sourceforge.net/projects/nfcproxy/

[14] LINEAGEOS ROM. LINEAGEOS ROM [online]. [cit. 2019-01-06]. Availible at: https://www.cyanogenmods.org/

[15] Github: RFIDIOT. Github: RFIDIOT [online]. 2018 [cit. 2019-01-06]. Availible at: https://github.com/AdamLaurie/RFIDIOt

[16] Kristin Paget. Credit Card Fraud: The Contactless Generation Presentation presented at: Shmoocon; 2011 January 27-29; USA (Washington D.C.) [online]. Avalible at: https://www.youtube.com/watch?v=HRXb-FZ6WFM

[17] STEVENS, W. Richard. TCP/IP Illustrated, Vol. 1: The Protocols. Addison-Wesley Professional, 1994. ISBN 978-0201633467.

[18] AUSSEL, Jean-Daniel and Ludovic ROUSSEAU. Pyscard - Python for smart cards. Githubpyscard - Python for smart cards [online]. 2014 [cit. 2019-01-06]. Avalible at: https://pyscard.sourceforge.io/index.html

[19] CZEKIS, A., K. KOSCHER, J. SMITH, T. KOHNO. RFIDs and Secret Handshakes: Defending Against Ghost-and-Leech Attacks and Unauthorized Reads with Context-Aware Communications. In: ACM Conference on Computer and Communications Security (2008)

Appendix A

Abbreviations

AAC Application Authentication Cryptogram

ADF Application Dedicated File

AFL Application File Locator

AID Application Identifier

APDU Application Protocol Data Unit

ARQC Authorisation Request Cryptogram

CLA Class Byte in command APDU

CPU Central Proccessing Unit

DF Dedicated File

EF Elementary File

EMV Europay Mastercard Visa

FCI File Control Information

HCE Host-based Card Emulation

I/O Input/Output

ICMP Internet Control Message Protocol

IGMP Internet Group Management Protocol

INS Instruction Byte in command APDU

IP Internet Protocol

ISO/IEC International Organization for Standardization/International Electrotechnical Commission

LAN Local Area Network

MF Master File

NFC Near Field Communication

NXP Next Experience (formerly Philips Semiconductors)

PPSE Visa Proximity Payment System Environment

RFID Radio Frequency IDentification

TC Transaction Certificate

TCP Transmission Control Protocol

UDP User Datagram Protocol

UI User Interface

Appendix B

Content of the attached CD

- `apks` ..... directory containing apks of both applications
  - `ghost-debug.apk`
  - `leech-debug.apk`
- `diagrams` ..... directory containing diagrams
- `pyscard scripts` ..... directory containing Python scripts for testing
- `screens`
  - `ghostRealTime.png` ..... screenshot from Ghost application
  - `ghostRealTimeWrongType.png` .... screenshot from Ghost application
  - `leechRealTime.png` ..... screenshot from Leech application
  - `readerSimulation.png` ..... log from testing simulation
- `source codes` ..... folder containing applications source codes
- `thesis`
  - `thesis.pdf` ..... thesis in PDF format
  - `thesis.tex` ..... thesis in LaTeX format
- `README.txt` ... content of the CD describtion and installation instructions