# Security Analysis of Mobile Payment Systems

**Atul Kumar**

EIT ICT Labs Master School

University of Twente

Under Supervision of:

Prof. Dr. F.E. Kargl

The Services, Cybersecurity and Safety Research Group

University of Twente, The Netherlands

Jordi van den Breekel

Eleonora Petridou

Information Protection Services

KPMG Netherlands

A thesis submitted for the degree of

*Master of Science in Security and Privacy*

July 2015

I would like to dedicate this thesis to *my parents*

# Acknowledgements

First and foremost I offer my sincerest gratitude to my supervisor, Prof. Dr. F.E. Kargl, who introduced me to the world of Near Field Communication (NFC) during one of the lectures on the Security and Privacy of Mobile Systems course, which was part of my Master study program at the University of Twente, The Netherlands. His support and immense knowledge greatly helped me to develop the different concepts discussed in this thesis. His constructive feedback was very helpful in identifying areas of improvement. One simply could not wish for a better or friendlier supervisor.

Next, I am very thankful to Jordi van den Breekel, my supervisor at KPMG. He has supported me throughout my thesis with his patience and knowledge whilst allowing me room to work in my own way. I attribute the level of my Masters thesis to his encouragement and effort and without him this thesis would not have been completed or written. He always motivated me to perform structured research which was very crucial in achieving the results in a timely manner. He was one of my biggest sources of motivation and inspiration.

I would also like to thank Eleonora Petridou, my second supervisor at KPMG. She always encouraged and helped me in improving the quality of my thesis. I would also like to thank her for introducing me to KPMG during one of the guest lectures at the University of Twente.

I also express my gratitude towards all my colleagues at KPMG, Amstelveen. Writing my thesis at the Information Protection Services

division at KPMG was one of the best experiences of my life. In my daily work I have been blessed with a friendly and cheerful group of fellow colleagues.

Nils Rodday, my friend as well as my classmate, has been an invaluable help through out my thesis. He was always available to have discussions and provide his useful opinions and was always willing to help me.

I am also indebted to the many countless contributors to the "Open Source" programming community for providing the numerous tools and systems I have used to produce both my results and this thesis. The entirety of my thesis has been completed using such technologies.

I am also thankful to Dr. Andreas Peter, a member of the graduation committee, for all his help and support in offering internships with different companies and various topics to choose from.

I also place on record my sense of gratitude to everyone who directly or indirectly lent their support in this thesis.

Finally, I thank my parents for supporting me throughout my studies. I also like to thank my brother, my sister-in-law and their little daughter (our little angel) for supporting and encouraging me to perform well in my studies. Lastly, I am grateful to the God for the good health and well-being that were necessary to complete this thesis.

# Abstract

Mobile payments have evolved from mobile banking to contactless payment which uses radio communication technology. NFC has enabled mobile devices to emulate contactless cards either by using hardware-based Secure Element (SE) or software-based i.e. Host Card Emulation (HCE). We provide a detailed comparison between the different forms of SE. We provide an analysis of HCE and a security mechanism implemented in Android 4.4 and above, which turns off the NFC controller and application controller when the device display screen is disabled, to prevent device skimming. We present a flaw in the design of the implementation of this security mechanism and provide a proof-of-concept for the same. In addition, we present different attack vectors like man-in-the-middle attack and denial-of-service attack for HCE-based applications. We also provide an analysis of the Vodafone NFC SIM card payment solution and describe different components involved. We also present different attack vectors like spoofing and relay attack on the Vodafone NFC SIM card payment solution. We also propose two countermeasures for relay attacks which are based on challenge response protocol.

# Contents

**Contents** **v**

**List of Figures** **ix**

Acronyms . . . . . xii

**1 Introduction** **1**

1.1 Mobile Payments . . . . . 1

1.2 Security Requirements . . . . . 2

1.3 Motivation of the Thesis . . . . . 2

1.4 Research Questions . . . . . 5

1.5 Methodology . . . . . 5

1.6 Scope . . . . . 6

1.7 Organization of the Thesis . . . . . 6

**2 Background** **8**

2.1 Standard mobile payment system model . . . . . 9

2.2 Security Mechanisms . . . . . 10

2.2.1 Device and owner identification . . . . . 11

2.2.2 Tokenization . . . . . 11

2.2.3 Trusted Execution Environment . . . . . 18

2.2.4 Secure Element . . . . . 20

2.2.5 Cloud based approach . . . . . 22

2.2.6 Code Obfuscation . . . . . 23

2.2.7 White-box Cryptography . . . . . 24

2.3 Near Field Communication . . . . . 24

2.3.1 Modes of communication . . . . . 25
2.4 NFC layers and standards . . . . . 25
2.5 Card Emulation . . . . . 28
2.5.1 Host-based Card Emulation . . . . . 29
2.5.2 Different flavors of HCE-based Mobile Payment System . . 30

**3 Secure Element and its different flavors 32**
3.1 Secure Element . . . . . 32
3.1.1 UICC . . . . . 33
3.1.1.1 Trusted Security Manager . . . . . 34
3.1.1.2 Over-the-Air provisioning . . . . . 35
3.1.1.3 Security features and available interface . . . . . 35
3.1.2 Embedded Secure Element . . . . . 37
3.1.3 Micro Secure Digital (Micro-SD)-based SE . . . . . 38
3.2 SE in the Cloud . . . . . 39
3.3 Comparison of the different forms of SE . . . . . 41

**4 Host-based Card Emulation 44**
4.1 Host Card Emulation . . . . . 45
4.2 Security Mechanisms . . . . . 45
4.3 Relay Attack . . . . . 46
4.3.1 Relay Attack on HCE-based Mobile Payment System . . . 48
4.3.1.1 Hardware-based Relay Attack . . . . . 48
4.3.1.2 Attack Scenario . . . . . 49
4.3.1.3 Software-based Relay Attack . . . . . 51
4.3.1.4 Attack Scenario . . . . . 51
4.4 Man-in-the-Middle Attack . . . . . 53
4.5 Denial-of-Service Attack . . . . . 55
4.5.1 Corrupt Routing Table . . . . . 56
4.5.2 Application Identifier (AID) conflict . . . . . 56
4.5.3 Quick Battery Discharge . . . . . 58
4.5.4 Malicious POS terminals . . . . . 58
4.6 Summary . . . . . 60

**5 NFC SIM Card** **62**
5.1 Vodafone NFC SIM card . . . 63
5.1.1 Vodafone Wallet . . . 63
5.1.2 Vodafone SmartPass . . . 63
5.2 Spoofing Attack . . . 64
5.2.1 Attack Scenario . . . 65
5.2.2 Assumptions . . . 65
5.2.3 Goal of this attack . . . 67
5.2.4 Limitation . . . 67
5.3 Relay Attack . . . 67
5.3.1 Attack Scenario . . . 67
5.3.2 Assumptions . . . 68
5.3.3 Goal of this attack . . . 69
5.3.4 Limitation . . . 69

**6 Countermeasures for Relay attack** **70**
6.1 Challenge Response Protocol . . . 70
6.1.1 Replay Attack . . . 71
6.1.2 Challenge response protocol for protection against relay attack . . . 72
6.2 Manual challenge response protocol . . . 72
6.3 Side channel challenge response protocol . . . 74

**7 Proof of Concept** **77**
7.1 Relay Attack on HCE . . . 77
7.2 The Experiment . . . 78
7.2.1 First Phase . . . 79
7.2.1.1 Arduino Uno Board . . . 80
7.2.1.2 Seeed Studio NFC Shield v2.0 . . . 80
7.2.1.3 Samsung Galaxy S4 . . . 80
7.2.1.4 Arduino IDE . . . 80
7.2.1.5 A Laptop . . . 81
7.2.1.6 HCE Sample Android Application . . . 81

7.2.1.7 Seeed Studio PN532 library implementing HCE . 82
7.2.2 The Experiment Setup . . . . . 82
7.2.2.1 Result . . . . . 84
7.2.2.2 Experiment with Android 4.4.2 on Nexus 7 . . . 86
7.2.2.3 Experiment with Android 5.1.1 on Nexus 5 . . . 87
7.2.3 Second Phase . . . . . 88
7.2.4 The Experiment Setup . . . . . 88
7.2.4.1 Result . . . . . 89
7.2.5 Countermeasures . . . . . 92
7.2.6 Experiments for verifying device lock feature . . . . . 93
7.2.6.1 The Experiment Setup . . . . . 93
7.2.6.2 The Experiment . . . . . 93
7.2.6.3 Results . . . . . 96

**8 Future Work and Conclusions 98**
8.1 Overview . . . . . 98
8.2 Conclusion . . . . . 100
8.3 Future Work . . . . . 102
8.3.1 Test the new attack vectors proposed . . . . . 102
8.3.2 Attacks on NFC Subscriber Identity Module (SIM) card . 102
8.3.3 Other possibilities of hardware-based relay attacks . . . . . 103
8.3.4 Countermeasures to relay attacks . . . . . 103

**Appendix A 104**

**Appendix B 108**

**References 111**

# List of Figures


2.1 An overview of Chapter 2 . . . . . 8
2.2 The transaction flow and key parties in mobile payment model. Adapted from [15] . . . . . 10
2.3 Account Data [20] . . . . . 12
2.4 The flow and key elements in token-based payment mechanism [9] 17
2.5 Typical Track2 Data vs LoopPay Tokenized Track 2 data [33] . . 18
2.6 Generate TPV [33] . . . . . 19
2.7 Sample token format as defined by PCI DSS [32] . . . . . 19
2.8 The normal and secure world environment [7] . . . . . 20
2.9 TEE Architecture defined by GlobalPlatform [12] . . . . . 21
2.10 Software based relay attack [6] . . . . . 22
2.11 Comparison between NFC layer and OSI layer [16] . . . . . 27
2.12 ISO standards for contact and contactless cards [14] . . . . . 28
2.13 Android HCE protocol stack [1] . . . . . 28
2.14 Android operating with both SE and HCE Adapted from [1] . . . 30

3.1 An overview of Chapter 3 . . . . . 33
3.2 Key capabilities of TSM [21] . . . . . 34
3.3 UICC-based SE [26] . . . . . 36
3.4 ISD and SSD hierarchy as suggested by Global Platform [5] . . . . 37
3.5 Embedded SE [26] . . . . . 38
3.6 MicroSD based SE [26] . . . . . 39
3.7 Comparison of different forms of SE [3] . . . . . 43

4.1 An overview of Chapter 4 . . . . . 44

4.2 Standard architecture of relay attack [28] . . . . . 47
4.3 Wireless Charger . . . . . 49
4.4 Wireless Charger as a mole for relay attack . . . . . 50
4.5 Software-based relay attack . . . . . 52
4.6 Man-in-the-middle attack . . . . . 54
4.7 Malicious POS terminal causing DoS attack . . . . . 59

5.1 An overview of Chapter 5 . . . . . 62
5.2 Spoofing attack Scenario for NFC SIM card . . . . . 66
5.3 Wireless charger as a mole . . . . . 68

6.1 An overview of Chapter 6 . . . . . 71
6.2 Manual Challenge Response protocol . . . . . 74
6.3 Side Channel Challenge Response protocol . . . . . 75

7.1 An overview of Chapter 7 . . . . . 78
7.2 Samsung Galaxy S4 with Android 4.4.2 . . . . . 81
7.3 Initial Setup . . . . . 83
7.4 Arduino IDE . . . . . 83
7.5 The overall setup . . . . . 84
7.6 Serial monitor output . . . . . 85
7.7 Serial monitor output showing communication with android HCE app . . . . . 85
7.8 Source code of the sample HCE app. Refer Appendix A . . . . . 86
7.9 Experiment with Android 5.1.1 . . . . . 87
7.10 Google Nexus 5 with Android 5.1.1 . . . . . 89
7.11 Initial setup with Google Nexus 5 and Wireless Charger . . . . . 90
7.12 Google Nexus 5 is kept on the Wireless Charger along with the NFC antenna . . . . . 91
7.13 Google Nexus 5 kept on the Wireless Charger(powered OFF) along with NFC antenna . . . . . 92
7.14 hce_aids file showing requireDeviceUnlock attribute is true in Google Wallet . . . . . 93

7.15 Google Wallet communicates with the NFC reader when the device is locked . . . . . 94
7.16 Use of apktool and SignApk . . . . . 95
7.17 Sample HCE application with requireDeviceUnlock set to true and Nexus 7 screen is on and locked . . . . . 95
7.18 requireDeviceUnlock attribute is set to false in the sample HCE application and true in Google Wallet . . . . . 96
7.19 Serial monitor shows that NFC reader is able to communicate to both applications . . . . . 97

# Acronyms

| | |
|---|---|
| AID | Application Identifier |
| APDU | Application Protocol Data Unit |
| API | Application Programming Interface |
| CPU | Central Processing Unit |
| CSC | Card Security Code |
| EMV | Europay, MasterCard, and Visa |
| eSE | Embedded Secure Element |
| GSM | Global System for Mobile Communications |
| HCE | Host Card Emulation |
| ID&V | Identification and Verification |
| IIN | Issuer Identification Number |
| IPC | Inter Process Communication |
| ISD | Issuer Security Domain |
| ISO | International Organization for Standardization |
| Micro-SD | Micro Secure Digital |
| MMU | Memory Management Unit |
| MNOs | Mobile Network Operators |

| | |
|---|---|
| MST | Magnetic Secure Technology |
| NFC | Near Field Communication |
| NFC-WI | Near Field Communication Wired Interface |
| OSI | Open Systems Interconnection |
| OTA | Over-the-air |
| PAN | Primary Account Number |
| PCI DSS | Payment Card Industry Data Security Standard |
| PIN | Personal Identification Number |
| POS | Point of Sale |
| QR | Quick Response |
| RF | Radio Frequency |
| RFID | Radio-frequency Identification |
| SE | Secure Element |
| SIM | Subscriber Identity Module |
| SMSC | Short Message Service Centre |
| SSD | Supplementary Security Domain |
| SSL/TLS | Secure Socket Layer/Transport Layer Security |
| SWP | Single Wire Protocol |
| TEE | Trusted Execution Environment |
| TMI | Token Mode Indicator |

| | |
|---|---|
| TPV | Token Part Value |
| TSM | Trusted Services Manager |
| UICC | Universal Integrated Circuit Card |
| UMTS | Universal Mobile Telecommunications System |

# Chapter 1

# Introduction

This chapter provides an introduction to the mobile payment system and its security requirements. It discusses the motivation behind this thesis. It provides the research questions which this thesis will try to solve. It discusses the methodologies which will be used to provide solutions to the research questions. It also defines the scope and the overall organization of the thesis.

## 1.1 Mobile Payments

Payment in general is an interaction between engaging parties regarding money transfer. In the past, one of the modes of payment available was cash, i.e. paying physical money in the form of notes or coins. As the world evolved after the invention of the Internet, virtual money came into the picture and was increasingly used. Physical cards like credit or debit cards started replacing cash. This allowed people to make payment using these cards at physical stores as well as online electronic commerce stores. People started carrying multiple bank cards (credit or debit) in their wallet as different banks provide different benefits to their customers. With the increase in the usage of these bank cards, online and card-based frauds started increasing. Fraudsters began to steal this card data by different means such as using malware to sniff the card data or using phishing to acquire bank credentials by tricking users. We need to replace these cards with something which might help in mitigating these frauds. Mobile phones developed

additional features by including multiple types of sensors and hardware components, which led to the creation of a wide number of application areas. One of them was a mobile-based payment system whose usage has doubled between 2012 and 2013 to $1 billion, and which is predicted to reach $57 billion by 2017[22]. Some of the key requirements for such a system are usability and security. As cards are being replaced by mobile phones, it will become more comfortable for the user since there is no need to carry credit or debit cards in their wallet. However, comfort should not lead to the compromise of security. It is necessary to implement security mechanisms to protect confidential data, i.e. card holder data. The security mechanisms should cover the confidentiality, integrity as well as availability of confidential data.

## 1.2 Security Requirements

Security is one of the key requirements for mobile-based payment systems. It needs to assure confidentiality and integrity of data involved in the transaction. It also requires proper authentication mechanisms so that each party involved in the transaction can be assured that they are talking to the right entity. The last requirement is non-repudiation which ensures that there is adequate proof of the message being sent by the intended party such as payment application or service provider. Some of these requirements are fulfilled by using cryptographic mechanisms like encryption, digital signatures, Personal Identification Number (PIN) codes, One time passwords or bio-metrics.

## 1.3 Motivation of the Thesis

The current mobile payment environment is drastically changing with the introduction of technologies like Near Field Communication (NFC), Secure Element (SE), Host Card Emulation (HCE), Quick Response (QR) Code and Magnetic Secure Technology (MST)[1]. Out of these, NFC, SE and HCE have received wider acceptance. Most of the mobile payment applications currently available utilizes

[1] `http://www.google.com/patents/US8628012`

NFC. Therefore, this thesis will focus only on NFC-based mobile payment.

In the case of hardware-based SE[1], the deployment and management of mobile payment solutions is dependent on the SE manufactures. Access to the SE is managed by SE manufacturers. Therefore, the service providers like financial institutions have to rely on the SE manufacturers. For example, if a service provider uses Universal Integrated Circuit Card (UICC)-based SE to deploy the payment application, the access is managed by Mobile Network Operators (MNOs) which adds additional cost for the service provider and also restricts the usage of the application to the users who have the UICC from the same MNOs. With the introduction of HCE in Android KitKat 4.4, it is possible to emulate a payment card without any need for hardware-based SE. This change has allowed banks and service providers to develop mobile payment solutions which do not have any dependency on hardware-based SE. Many mobile devices with NFC capabilities and running Android 4.4 or above can emulate a payment card. However, removing hardware-based SE will impact the security of these solutions as there will be a lack of local secure storage which might be needed by mobile payment solutions. Therefore, an analysis is required as to whether introduction of HCE actually lowers the security level (and how much) compared to SE without taking into account the requirement to enhance security.

A relay attack is considered as one of the prevalent attack vectors because of its simplicity, its effects and its ability to circumvent the cryptographic protections. The countermeasures such as time or distance-bounding protocols which are suggested or implemented for relay attack have some limitations. This can be illustrated by an example. So-called relay attacks forward all requests from a Point of Sale (POS) terminal to a (typically infected) legitimate smartphone where data is stored in the SE. One of the proposed countermeasures for mitigation is time-bounding protocol. The time bounding protocol measures the round-trip-time between Point of Sale (POS) terminal and the smartcard; if this time is too long, an alarm would be triggered and the transaction is refused. However, it is challenging to fix the maximum round-trip time required, which

[1]Hardware-based SE refers to UICC, embedded SE and Micro-SD-based SE

limits the overall efficiency of this protocol. When introducing HCE, data may be stored in a cloud-based SE somewhere in the Internet. Communication with this cloud-based SE obviously introduces unpredictable additional delay and may render the time-bounding protocol ineffective. One of the security mechanisms implemented in HCE is to disable the NFC which in-turn stops the HCE service, if the device screen is inactive. Does this protect the mobile device from interacting with an unintended device to hinder a relay attack? It is these kinds of consequences of HCE introduction that this thesis investigates. Our goal is to give qualified recommendations about adoption of HCE and also propose security enhancements.

With an NFC SIM card, an NFC chip is embedded in the same UICC which might allow the SE to directly communicate with the NFC reader. The mobile device (base-band processor) has a limited interface to communicate with the UICC. This interface does not allow any application running in the application processor to communicate with the UICC. With the introduction of an NFC SIM card an additional interface is provided for communication. This additional interface might allow applications running in the application processor to communicate with the NFC SIM card. As mentioned earlier, inclusion of an NFC chip might allow an NFC reader to directly communicate with the data stored in the SE[1] and this might assist in performing a relay attack if there is no additional protection in place. This thesis will investigate such scenarios and propose countermeasures for relay attack.

[1]In vodafone NFC SIM card, the POS terminal can communicate with SE when the battery of the mobile device is dead; this suggest that direct communication between the NFC SIM card and the POS terminal is possible `https://vodafonesmartpass.com/reg/static/html/nl/faq.html#3-9`

## 1.4 Research Questions

The research questions which need to be answered are:

1. In terms of security, re-usability and standardization, which form of SE i.e. UICC, Micro-SD, Embedded Secure Element (eSE) and cloud-based SE is better?

2. What are the different attack vectors and security controls in mobile payment systems based on HCE?

3. What are the different attack vectors and security controls in an NFC SIM card-based mobile payment systems?

4. Are existing security controls against relay attacks adequate or, if not, can we propose enhanced ones?

## 1.5 Methodology

We will provide a comparison between the different NFC-based mobile payment systems using different forms of SE i.e. UICC, Micro-SD, embedded and cloud-based approach, by performing a literature research based on the scientific papers, journals and white papers published. We will discuss the benefits and drawbacks of each of them.

To analyze the mobile payment solution based on HCE we will study the specifications provided by Android by referring to the documentation available on the android developer page. We will analyze the security measures suggested which are specific to the HCE on the Android documentation. We will then identify if there are potential ways to bypass some of the security measures specific to the HCE. As there are limited specifications available such as Android documentation, one approach will be to utilize reverse engineering techniques to understand the working of some of the currently available HCE-based solutions.

To analyze the mobile payment solution based on NFC SIM cards, we will study the specifications available like Open Mobile Application Programming Interface (API). One of the NFC SIM card-based mobile payment solutions was recently launched in the Netherlands by Vodafone[1]. We will analyze this mobile payment solution to identify potential risks and vulnerabilities by using techniques like reverse engineering. We will de-compile the apk[2] files of the Vodafone applications and perform an analysis to identify potential risks.

To answer research question 4 we will analyze the different countermeasures for relay attack and limitations in their implementation. We will suggest new countermeasures which will try to make it a little more difficult for an attacker to perform a relay attack. Most of the mobile devices are equipped with multiple sensors. We will perform research on the use of different sensors in the mobile device like microphone. We will explore the potential use of some of the sensors available in mobile phones to propose feasible solutions.

## 1.6 Scope

The scope of this thesis is NFC-based mobile payment systems with more emphasis on the HCE-based system and NFC SIM cards. The thesis scope also includes NFC-based mobile payment system based on SE in the UICC, SE in the cloud, Micro-SD and eSE in the mobile device.

## 1.7 Organization of the Thesis

The remainder of this thesis is structured as follows:

**Chapter 2** discusses the standard mobile payment system and the different security mechanisms applicable for the same. It provides details regarding NFC and its different modes of operation.

[1] `http://goo.gl/TPQJp2`
[2] `https://www.androidpit.com/android-for-beginners-what-is-an-apk-file`

**Chapter 3** discusses the different forms of secure storage used in the NFC-based mobile payment environment. It provides a comparison of different forms of SE. It also analyzes the benefits and limitations of each of them.

**Chapter 4** discusses different security mechanisms and attack vectors which we have identified, that are applicable for the HCE-based mobile payment system.

**Chapter 5** discusses the Vodafone NFC SIM card which enables Vodafone customers to make NFC-based mobile payment. It discusses two attack vectors we have identified that are applicable for an NFC SIM card-based payment solution.

**Chapter 6** discusses the two countermeasures we have proposed for protection against relay attack.

**Chapter 7** provides the details about the proof-of-concept.

**Chapter 8** provides the conclusion and discusses the possibilities of future work.

# Chapter 2

# Background

This chapter discusses the standard mobile payment system and the different security mechanisms applicable for the same. It discusses Near Field Communication and its different modes of operation. It discusses the foundational knowledge needed to understand the rest of this thesis. Figure 2.1 provides an overview of this chapter.

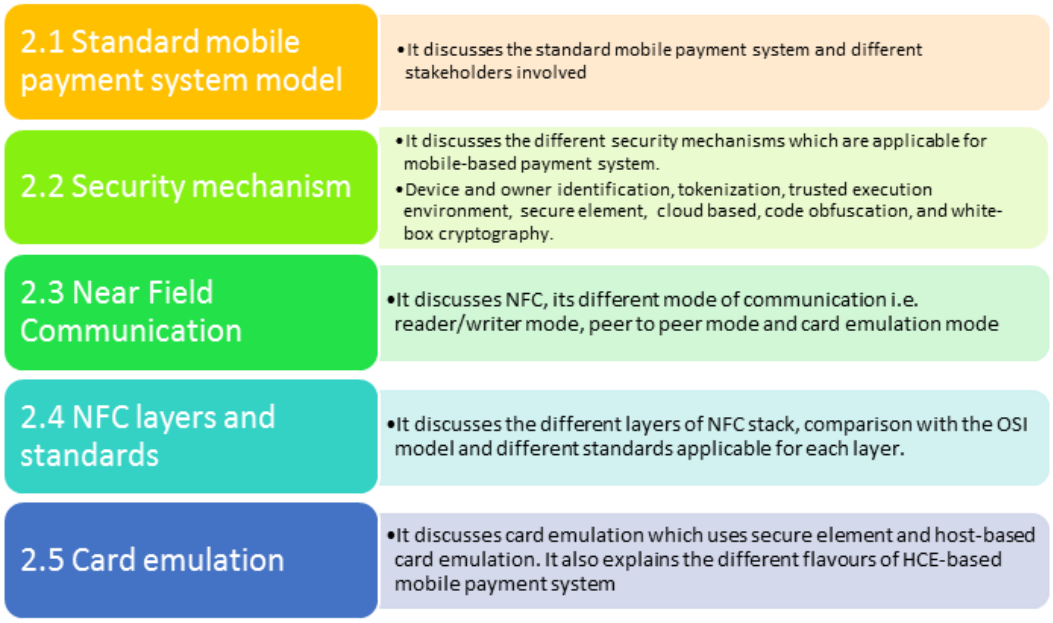


Figure 2.1: An overview of Chapter 2

## 2.1 Standard mobile payment system model

In this section, an overview of the mobile payment system model is outlined in order to understand the characteristics and interactions among engaging parties in mobile-based payment systems. The most common mobile-based payment model consists of five engaging parties.

1. *Client* is the user with a mobile device that requests to buy a product or service from a merchant.

2. *Merchant* is the entity which offers the products and services to the customers and has a payment terminal, i.e. POS, which allows the client to make the payment.

3. *Issuer* is the institution which manages the bank account of the client. It maintains and manages the funds on behalf of the client.

4. *Acquirer* is the institution which manages the bank account of the merchant and takes care of the fund transfers.

5. *Payment Gateway* is an entity which acts as a proxy between the issuer and acquirer. It acts as an interface between payment terminals at the merchant's place and bank private payment network.

To explain the transaction flow between the engaging parties, we discuss the NFC-based mobile payment systems. In this scenario, the client's mobile device has NFC capabilities which allow the device to emulate contact-less cards. With NFC, the client taps the mobile device on the POS terminal. This POS terminal acts as an interface between the card and payment gateway. The card details and other transaction data are sent to the acquirer. It connects to the Issuer via


Figure 2.2: The transaction flow and key parties in mobile payment model. Adapted from [15]

the payment gateway which authenticates the card data and transaction amount. The response is sent back to the POS terminal, which provides the receipt to the client. The issuer then transfers the fund to the acquirer and the transaction is complete. Figure 2.2 depicts all the engaging parties and the use case explained above.

## 2.2 Security Mechanisms

To make mobile payments secure, there are multiple security solutions available in the current mobile payment environment. This section discusses the different security mechanisms currently implemented in the mobile payment applications.

These security mechanisms are:

1. Device and owner identification
2. Tokenization

3. Trusted Execution Environment
4. Secure Element
5. Cloud-based approach (Secure Element on the cloud)
6. Code Obfuscation
7. White box cryptography

### 2.2.1 Device and owner identification

In the case of mobile devices, it is necessary to verify if the user who is making the payment is the authorized user. One of the common mechanisms is the use of PIN codes. They are usually 4 digits in length, which might be susceptible to brute force attack. Another method is the use of bio-metrics like fingerprint scanners or heartbeat sensors. Regarding fingerprint scanners, they are proven to be insecure as it is relatively easy to copy the fingerprint and fake it[1]. In the case of heart beat sensors, they alone are not fully secure as the heart beat is not unique for every individual. These sensors can be made more secure by combining them with other authentication mechanisms. In addition to user authentication, device identification is also necessary. One mechanism is to generate a randomized unique device number which cannot be easily guessed or copied. This number can be protected from being copied by using cryptographic mechanisms like encryption. Another proposed solution in [19] is derived credentials in which data required for authentication is generated using protocredential and user-provided secrets like a PIN code. The protocredential is a form of data structure which is stored in the mobile device. When the PIN is provided by the user, the cryptographic function uses the PIN and protocredential to generate a key pair for further transactions.

### 2.2.2 Tokenization

In a mobile-based payment system, the mobile device shares the cardholder data like Primary Account Number (PAN) with the merchant terminal. PAN is the

[1] `https://srlabs.de/spoofing-fingerprints`

number found on the payment cards like credit or debit cards. This number is allocated in accordance with ISO/IEC 7812[1]. The next section discusses PAN and other cardholder data in detail.

**PAN and other card data** As defined by Payment Card Industry Data Security Standard (PCI DSS)[20], the overall account data is split into two parts: Cardholder data and Sensitive Authentication Data. Figure 2.3 shows the individual components.

| Account Data | |
|---|---|
| **Cardholder Data includes:** | **Sensitive Authentication Data includes:** |
| ▪ Primary Account Number (PAN)<br>▪ Cardholder Name<br>▪ Expiration Date<br>▪ Service Code | ▪ Full track data (magnetic-stripe data or equivalent on a chip)<br>▪ CAV2/CVC2/CVV2/CID<br>▪ PINs/PIN blocks |

Figure 2.3: Account Data [20]

**PAN** A PAN, as defined in ISO/IEC 7812, is usually 16 digits but can be up to 19 digits. The structure[2] of a PAN is as follows:

- The first six digits refer to the Issuer Identification Number (IIN) which identifies the institution that has issued the card. Some of the well known institutions are VISA, MasterCard, etc.
- a variable length (up to 12 digits) individual account identifier which is issued by the banking or financial institution.
- The last digit is the check digit which is calculated using Luhn algorithm[3]. It is used for error detection.

**Cardholder name** This corresponds to the name of the person who holds the account.

**Expiration date** This corresponds to the date of the expiration of the card.

[1] http://www.iso.org/iso/catalogue_detail?csnumber=39699
[2] https://en.wikipedia.org/wiki/Bank_card_number
[3] http://planetcalc.com/2464/

**Service Code** This is a three digit number that specifies acceptance requirements and limitations for a magnetic-stripe-read transaction. The details of the three digits[1] are as follows:

*First digit*

- 1: International interchange OK
- 2: International interchange, use IC (chip) where feasible
- 5: National interchange only except under bilateral agreement
- 6: National interchange only except under bilateral agreement, use IC (chip) where feasible
- 7: No interchange except under bilateral agreement (closed loop)
- 9: Test

*Second digit*

- 0: Normal
- 2: Contact issuer via online means
- 4: Contact issuer via online means except under bilateral agreement

*Third digit*

- 0: No restrictions, PIN required
- 1: No restrictions
- 2: Goods and services only (no cash)
- 3: ATM only, PIN required
- 4: Cash only

[1] `https://en.wikipedia.org/wiki/Magnetic_stripe_card`

- 5: Goods and services only (no cash), PIN required
- 6: No restrictions, use PIN where feasible
- 7: Goods and services only (no cash), use PIN where feasible

**Full track data** The magnetic strip has multiple tracks which contain specific information. The details of the different tracks are as follows[1]:

*Track 1*

The Track 1 structure is specified as:

- STX : Start sentinel "%"
- FC : Format code "B" (The format described here. Format "A" is reserved for proprietary use.)
- PAN : Primary Account Number, up to 19 digits
- FS : Separator "^"
- NM : Name, 2 to 26 characters (including separators, where appropriate, between surname, first name etc.)
- FS : Separator "^"
- ED : Expiration data, 4 digits or Separator "^"
- SC : Service code, 3 digits or Separator "^"
- DD : Discretionary data, balance of characters
- ETX : End sentinel "?"
- LRC : Longitudinal redundancy check, calculated according to ISO/IEC 7811-2

[1] `https://en.wikipedia.org/wiki/ISO/IEC_7813`

*Track 2*

The Track 2 structure is specified as:

- STX : Start sentinel ";"
- PAN : Primary Account Number, up to 19 digits, as defined in ISO/IEC 7812-1
- FS : Separator "="
- ED : Expiration date, YYMM or "=" if not present
- SC : Service code, 3 digits or "=" if not present
- DD : Discretionary data, balance of available digits
- ETX : End sentinel "?"
- LRC : Longitudinal redundancy check, calculated according to ISO/IEC 7811-2

**CAV2/CVC2/CVV2/CID** A Card Security Code (CSC)[1] (also referred as card verification data, card verification number, card verification value, card verification value code, card verification code, verification code (V-code or V code), card code verification, or signature panel code) is a security feature for card-not-present payment card transactions. This is usually a 3 digit or 4 digit number present on the card.

**PIN** A 4 digit number which is used to authenticate the owner of the card.

In tokenization, the sensitive data i.e. PAN, is replaced by a payment token, which is shared with the payment terminal. This token is a surrogate value which is generated in such a way that it does not leak any information about

[1] `https://www.cvvnumber.com/`

the actual PAN. This payment token is different for each transaction. The main advantage is that the actual PAN is never shared with the merchant terminals which decreases the overall impact of a successful breach of payment data from the merchant terminal or database. Multiple approaches have been proposed relating to tokenization. One of them is proposed by Europay, MasterCard, and Visa (EMV) which provides the specifications for tokenization. In this specification, the payment token is a 13 to 19 digit numeric value which complies with the rules for a genuine account number. To get an overview of tokenization, one of the use cases for NFC-based mobile payment is explained in the specification[9].

As shown in Figure 2.4, the NFC mobile device either stores the payment token or receives it from the cloud-based storage. This token is shared with the merchant terminal. The Token Service Provider has a token vault which contains a mapping for the token and the corresponding PAN. This PAN is shared with an issuer for validating and completing the transaction. The token can be used with all Cardholder Verification Methods (CVM) but are limited in scope, for example, use of the token can be restricted to a single merchant or duration by putting an expiry value. For the payment token, the issuer needs to perform Identification and Verification (ID&V) steps to make sure the token is mapped to a valid and authorized PAN. There are additional controls like token domains and token assurance level, which help in preventing fraud in the use of tokens. Token domain restricts the use of token to a specific channel (e.g. NFC only), Merchant-specific, digital wallet-specific, or a combination of any of the above. On the other hand, the token assurance level indicates the confidence level that relates to the binding of the payment token and PAN. This level is determined by the result of the ID&V steps.

As mentioned in [25], one of the key challenges in the tokenization approach is to use methodologies which generate cryptographically-irreversible tokens. From the given token, it should be impossible to get any clue of the PAN. Another challenge is to provide adequate protection to the Token Vault as the token-PAN mapping is stored in it. Because of the sensitive information stored in the vault, it becomes an attractive target for criminals. In addition to this, tokenization

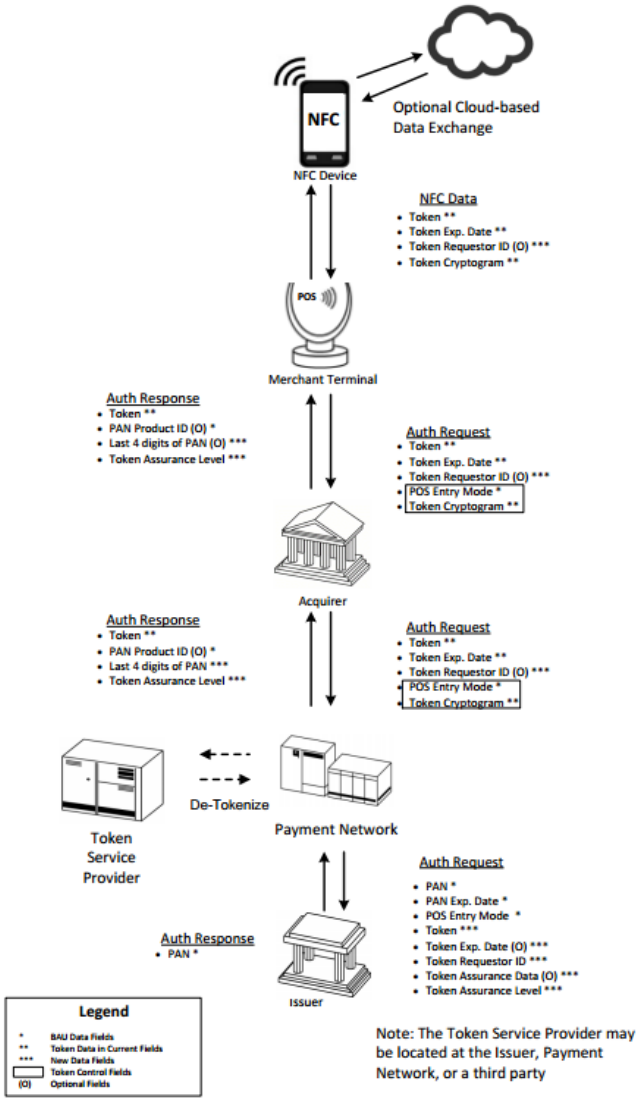


Figure 2.4: The flow and key elements in token-based payment mechanism [9]

is also susceptible to replay attacks so proper controls need to be in place to validate the authenticity and integrity of the tokens. Also, the implementation of the ID&V steps needs to be correct so that the token requests cannot be forged.

One of the examples of tokenization is explained in LoopPay Tokenization Whitepaper [33]. LoopPay does not replace the PAN, it replaces the other data

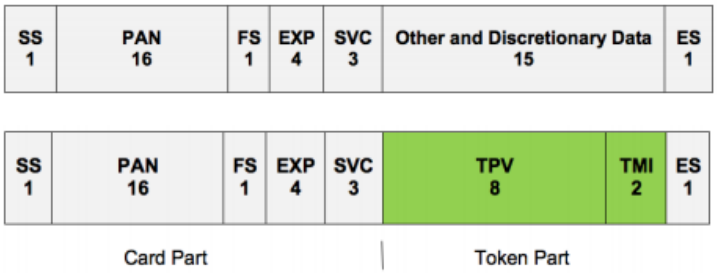


Figure 2.5: Typical Track2 Data vs LoopPay Tokenized Track 2 data [33]

and discretionary data. Figure 2.5 shows the data which is replaced. As suggested in the white paper [33], PAN or some part of PAN is used by merchants for various functions like routing, etc. The replaced data is the token which comprises Token Part Value (TPV) and Token Mode Indicator (TMI). The TPV is generated from the following three elements.

- PAN, Expiry and service code.
- Token Sequence Number
- Hash obtained from discretionary data and Loop Account ID.

Using the TPV generation algorithm, the dynamic one-time use token is generated. The token sequence number adds the randomness to the generated token. Figure 2.6 shows the details of TPV generation algorithm. The TMI allows card issuers to identify tokenized cards, validate them and manage the tokens' life cycles.

The tokenization specification published by PCI DSS [32] shows the sample examples in which the PAN is replaced. Figure 2.7 shows the sample examples.

### 2.2.3 Trusted Execution Environment

Trusted Execution Environment (TEE) is a secure area in the main processor of the mobile phone which allows safe execution of the trusted applications. It also provides mechanisms for secure storage of sensitive data like payment card information. TEE runs its own operating system which segregates the hardware

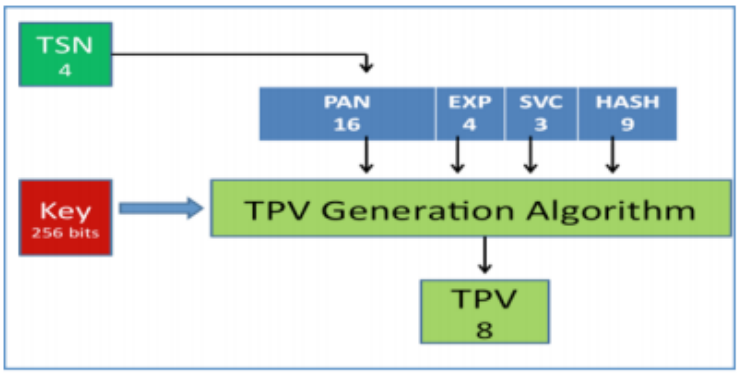


Figure 2.6: Generate TPV [33]

| PAN | Token | Comment |
|---|---|---|
| 3124 005917 23387 | 7aF1Zx118523mw4cwl5x2 | Token consists of alphabetic and numeric characters |
| 4959 0059 0172 3389 | 729129118523184663129 | Token consists of numeric characters only |
| 5994 0059 0172 3383 | 599400x18523mw4cw3383 | Token consists of truncated PAN (first 6, last 4 of PAN are retained) with alphabetic and numeric characters replacing middle digits. |

Figure 2.7: Sample token format as defined by PCI DSS [32]

and software resources from the main mobile operating system. It enforces access controls to protect access to sensitive data and execution of trusted applications. One of the popular TEE implementations is observed in ARM i.e. TrustZone[7]. In this framework, the processor cores are divided into two virtual cores, which represent the normal world and secure world respectively. Figure 2.8 shows the normal and secure world environment. By default, the secure world can access all states of the normal world, but not vice-versa. This creates another level of execution privilege in addition to the traditional distinction of user and kernel modes. Swapping between the two worlds is carefully controlled by the monitor mode. This mode is a higher privilege mode, which can control the mode which needs to be active. In addition, each virtual processor has access to its own virtual Memory Management Unit (MMU). Cache memories have additional tag bits to identify if the content is cached by the secure or normal world. Figure

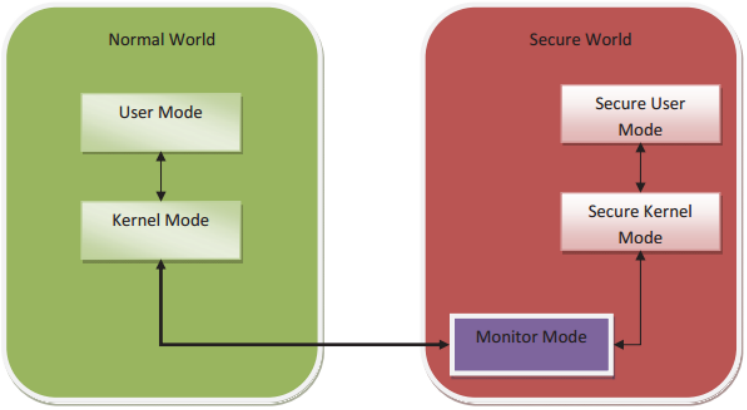


Figure 2.8: The normal and secure world environment [7]

2.9 shows the TEE architecture defined by GlobalPlatform [12]. As shown, the mobile operating system is segregated from the TEE kernel. The TEE manages the Trusted Applications as well as secure resources.

Despite its benefits, TEE is susceptible to cloning attacks, side channel attacks, changing the behavior of the TEE (e.g. Using buffer overflow attacks) to get access to the sensitive data or make the TEE execute unauthorized services [8]. The major limitation of this mechanism is its high implementation cost as well as lack of programmability.

### 2.2.4 Secure Element

SE is a tamper-resistant platform which is a secure micro-controller which can store trusted applications and their confidential data like secret keys, PAN, etc. It is present in three forms: UICC, eSE and Micro-SD [10]. Out of the three, UICC is more commonly used.

The UICC is a smart card which has its own operating system. It can store confidential data as well as run sensitive applications. In UICC, the mobile operator has the access to securely provision the application and the cryptographic data in the chip. The mobile payment application is provided by the issuing bank

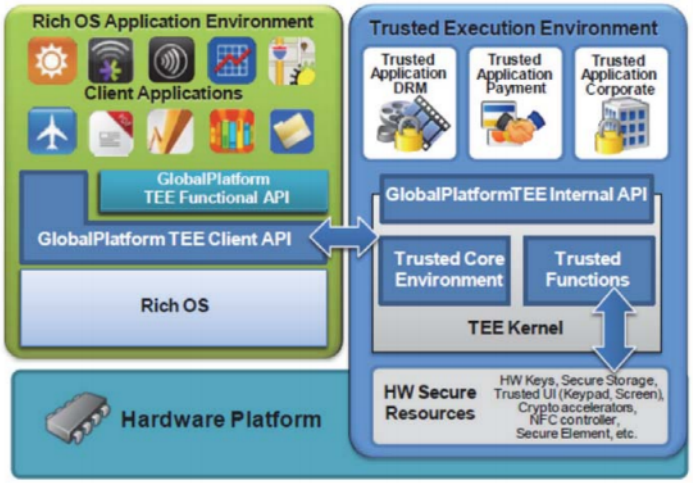


Figure 2.9: TEE Architecture defined by GlobalPlatform [12]

and the mobile operator needs to provide access to them. To make this happen in a secure manner, Trusted Services Manager (TSM) is used. Once the application is set up along with the data, NFC can be used to communicate with the POS terminals. The NFC antenna has access to the SE and it can only communicate with the mobile payment applications and the data. It uses ISO/IEC 14443 A/B for the communication. The mobile operating system does not have access to the SE. Therefore, the application and data stored in the SE cannot be accessed by any malicious application running in the mobile operating system environment.

In [6], two possible attack scenarios have been proposed against SE. One of them is a denial-of-service attack and second is relay attack. This paper considers the card emulation internal mode, which allows the mobile application running in the application processor to access the SE using the API. In this mode of embedded SE, ten successive authentication failures lock down the card in TERMINATED state. Therefore, if the card is in TERMINATED state, it can no longer be accessed. However, for card management tasks access to the SE API is necessary. A malicious application can gain access to this API and perform a trivial denial-of-service attack by locking down the card and making it unavailable for use. To protect the SE from such attacks, the API access needs to be restricted only to the trusted applications.

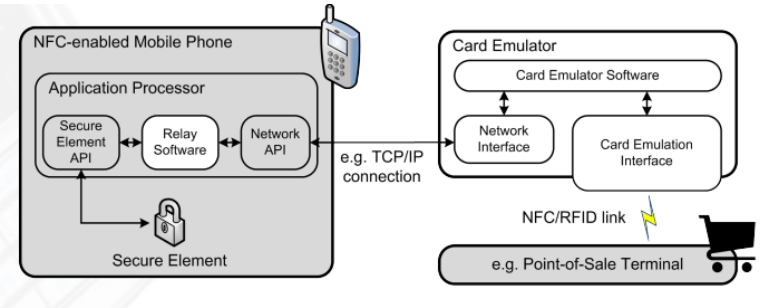


Figure 2.10: Software based relay attack [6]

Another possible attack scenario is the relay attack in which an attacker manages to install malicious relay software on the smartphone. In addition, the attacker places a card emulator close to the POS terminal. The relay software forwards all Application Protocol Data Unit (APDU) to the card emulator and vice versa, thus making the terminal and secure element believe that they are close to each other and are communicating directly with each other. The channel used for relaying the APDU can be Bluetooth, WiFi, etc. Figure 2.10, shows the working of this attack.

### 2.2.5 Cloud based approach

In cloud-based approach, confidential data can be stored on the cloud servers and can be accessed by the mobile applications when needed. This transfers the security requirements of the mobile device to the cloud service providers. In tokenization, the token can be generated or stored in the cloud and can be shared with the mobile payment application during transaction requests. However, this requires secure mechanisms to transfer the tokens as well as to store the data on the servers. It is also necessary to have proper authentication mechanisms in place so that unauthorized access to the cloud can be blocked.

In the case of SE in the cloud, issuing banks need to be sure that the device is authorized to access the data. This can be done using device fingerprinting [11]. Another challenge with a cloud-based approach is that the mobile device

needs to have an internet connection, to access the cloud, which might not be always feasible. Without Internet access, the device cannot get the keys as well as tokens. One possible solution is to store this data locally on the mobile device. This mode is called offline authentication. It needs additional security measures to protect the data which is stored temporarily in the device.

### 2.2.6 Code Obfuscation

Code obfuscation is a technique to protect mobile code from reverse engineering. Lots of mobile applications in the android market are written in Java, which is open to reverse engineering [4]. The reverse engineering allows malicious users to understand the logic of the source code, cryptographic implementation and extract confidential data. It also allows them to modify the source code or inject malicious code into legitimate applications and modify the behavior of the applications. Obfuscated code makes it difficult for the automated tools to de-compile the binary files to generate the meaningful source code. The obfuscated source code makes it difficult to understand the logic. Some of the code obfuscation techniques proposed in [4] are: identifier renaming, control statement insertion or dead code insertion and comment removal. Modifying the class names, attributes and method names allows the information about their functionality to be hidden. A second technique inserts an unnecessary control statement to create confusion in the program flow. Lastly, comments are very useful to understand the purpose of the methods or attributes. Removing this information will make it difficult to figure out.

Another obfuscation technique is explained in [2]. It proposes using the same identifier name repeatedly so that the decompiler or reverse engineering tools get confused and cannot understand the context behind the use of the identifier. However, it needs to take into consideration naming conventions like in Java as the different sub-packages in a package cannot have a same name.

### 2.2.7 White-box Cryptography

The major issue with the implementation of cryptographic algorithms is the secrecy of the keys. In white-box cryptography it is assumed that the attacker has complete access to the device but is still unable to uncover the key. In this approach, the keys are embedded in the implementation itself so that they are hidden and there is no need to input the keys. The attacker can copy the implementation and clone it, but the key remains the same. The cloning will consequently be detected if each application uses a different key. In symmetric block ciphers, there are substitution boxes as well as a linear transformation. White-box cryptography allows the creation of the huge lookup table which takes plain text and provides cipher text for the specific key. This transformation can be embedded with random input and output encoding to create ambiguity and make it difficult to figure out the key.

This approach will be useful for mobile payment applications as the separate secure storage requirements for the keys are not needed. However, the major drawback of this approach is to generate a unique application for each device which embeds the unique key. In addition to this, white-box cryptography is limited to symmetric cryptography and it is slower and needs more resources for implementation.

## 2.3 Near Field Communication

Near Field Communication is a short range radio communication technology. It enables mobile phones and other devices to establish radio communication with each other by touching them together or bringing them in close proximity. It allows the exchange of data between devices typically over a distance of 10 cm (3.9 in) or less. It operates at a frequency of 13.56 MHz and data transfer rates ranging from 106 kbps to 424 kbps.

### 2.3.1 Modes of communication

NFC devices use the electromagnetic induction i.e. Radio Frequency (RF) field generated by the loop antennas to communicate with each other. During the communication, each device can operate as either an active or a passive device. The active device is the one which has its own energy source to generate the RF field while the passive device is the one which does not have its own energy source and uses the RF field generated by the active device to power itself. NFC device supports three modes of communication:

- In *Reader/Writer mode*, an NFC device behaves as a reader for NFC tags, such as contactless smart cards and Radio-frequency Identification (RFID) tags. It detects a tag immediately in close proximity by using a collision avoidance mechanism. Once detected, it can either read data from or write data to the detected tag. Smart posters are an important application for this mode.

- In *Peer-to-Peer mode*, two NFC-enabled devices can exchange information between each other. This is the mode used by Android Beam technology. Exchanging photos, business cards and performing money transfer between friends are some of the applications for this mode.

- In *Card-emulation mode*, an NFC device behaves like a contactless smart card. In this mode, the mobile phone does not generate its own RF field but the NFC reader creates this field instead. Therefore, as long as a mobile platform supports the emulation of protocols surrounding ISO/IEC 14443 that regular contactless cards use, it should be able to function properly. Both Android and Blackberry do this and can therefore be used to emulate contactless cards. In this mode, mobile phones can be used in place of credit cards, debit cards, transit cards, access cards and so on.

## 2.4 NFC layers and standards

The NFC layers can be compared with the different layers of the Open Systems Interconnection (OSI) model. Figure 2.11 shows a comparison of the layers in

NFC and OSI. In [16], the author explains the different NFC layers as follows:

**Physical characteristics** This layer provides specifications for the size and temperature operating limits of the NFC device.

**Radio frequency interface** This layer specifies the protocol used to transform the bits to the physical signal.

**Initialization and anti-collision** This layer provides the specifications for the initialization of NFC devices when they are in close proximity. It also specifies anti-collision protocols to resolve if there are multiple NFC devices in close proximity.

**Transmission protocol** This layer corresponds to the error detection for reliable communication.

**Application protocol** This layer corresponds to the specifications for formatting of the data.

Figure 2.12 shows the standards involved in contact and contactless cards. (The contactless card refers to one using NFC). In [14], the author describes the different parts of ISO 14443 standards.

**ISO 14443-1** The physical characteristics are defined by ISO 14443-1. It define several environmental operating factors that the card must be capable of withstanding without any permanent damage to the functionality. For example, the environment temperature range should be between 0 to 50 degree Celsius.

**ISO 14443-2** This define the radio frequency power and signal interface. There are two types of cards i.e. Type A and Type B. These cards differ in the bit rates and modulation schemes [34].

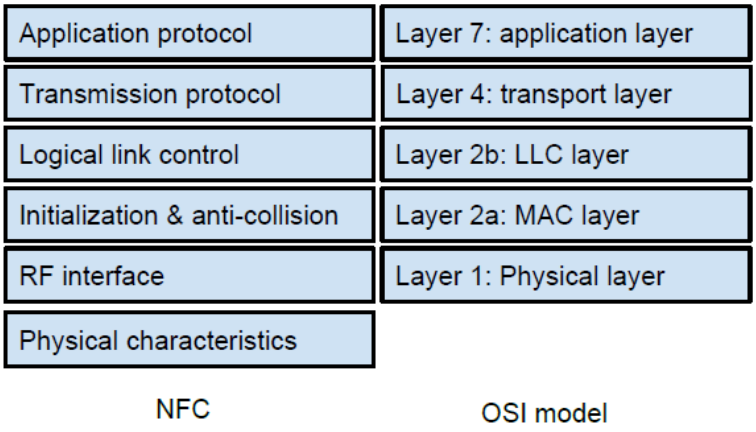


Figure 2.11: Comparison between NFC layer and OSI layer [16]

**ISO 14443-3** This defines the initialization and anti-collision mechanisms implemented for Type A and Type B. It defines the anti-collision commands, responses, data frame, and timing.

**ISO 14443-4** This defines the high-level data transmission protocols for both type of cards i.e. Type A and Type B. It provides supports for the application layer protocol i.e. ISO 7816-4.

**ISO 7816-4** This defines the format of the message being transferred between different NFC devices. The messages transferred are termed as APDU. It defines the APDU command, APDU response, class byte, instruction byte, parameter bytes, data field bytes, status word, basic channel and logical channel [29].

Android 4.4 supports emulating cards that are based on the ISO-DEP specification (which is based on ISO 14443-4) and process the APDUs as defined in the ISO 7816-4 specification. Figure 2.13 shows the protocol stack in Android HCE [1].

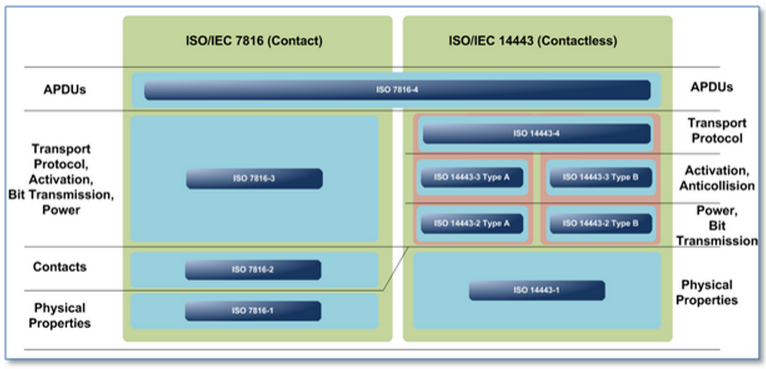


Figure 2.12: ISO standards for contact and contactless cards [14]

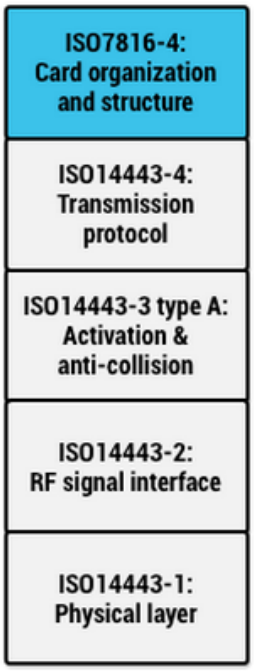


Figure 2.13: Android HCE protocol stack [1]

## 2.5 Card Emulation

It is possible for a mobile device to behave as a smart card with the use of technology like card emulation. It can emulate a contactless card using dedicated hardware like SE or software like HCE. Card emulation using SE has already been discussed in the section 2.2.4. In the next section, HCE is explained in detail.

### 2.5.1 Host-based Card Emulation

HCE is an alternative solution for SE. As SE adds cost, and UICC SE has access restrictions from mobile operators, there was a need of a solution such as HCE. In Android 4.4 KitKat, HCE was introduced which allows the NFC controller to send the commands for Card Emulation Mode to the HCE service running on the host CPU instead of SE. For example, the user can download a mobile payment application from the store and its AID is registered with the NFC Controller routing table which can route the commands to the Host CPU. This removes the involvement of mobile operators as well as TSM. As SE is considered to be tamper-resistant as well as more secure, HCE needs to have additional security measures especially for isolation and access control to prevent rogue applications on the smartphone easily accessing and exfiltrating payment data.

In Android 4.4, HCE is implemented as a service as it is allowed to run as a background process. Android provides *HostApduService* class which needs to be extended in the implementation of the bank application. *HostApduService* class declares two abstract methods that need to be overridden and implemented. The first method is *processCommandApdu()* which is responsible for handling all the APDUs received from the NFC reader. The second method is *processCommandApdu()* which is responsible for sending the response APDUs back to the NFC reader. Method *onDeactivated()* is called when one of the following cases occur:

- The NFC reader sends another "SELECT AID" APDU, which the operating system resolves to a different service,
- The NFC link between the NFC reader and your device is broken.

Figure 2.14[1] shows the flow of APDUs if the device has both HCE as well as SE. The default route in Android is set to HCE. If there is no route information available for specific AID, it will be sent to HCE instead of SE. The APDUs which are sent to SE are not passed through the HCE service. There is no Android API provided to access the SE from the host CPU. The application implementing the

[1] `https://developer.android.com/guide/topics/connectivity/nfc/hce.html`

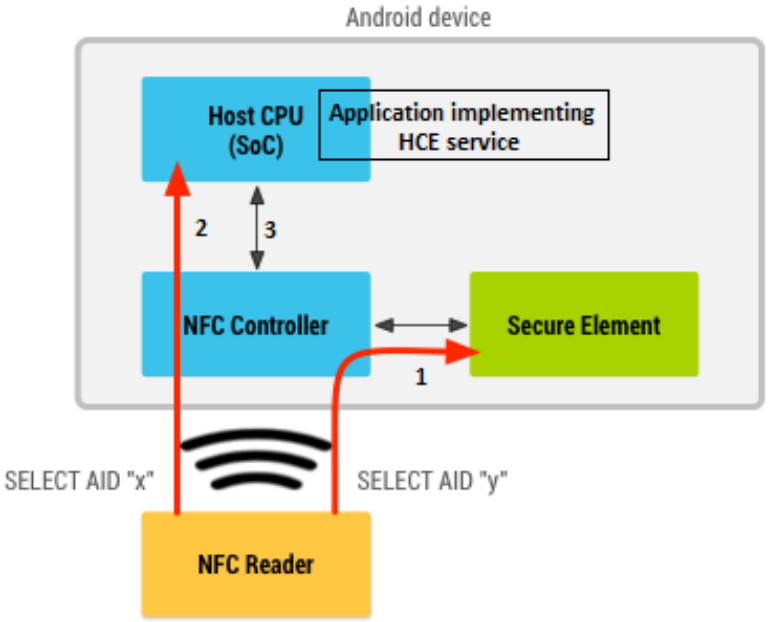


Figure 2.14: Android operating with both SE and HCE Adapted from [1]

HCE service runs on the Host CPU. Therefore, path 3 defines the route if the AID belongs to the application implementing the HCE service. If the application is hosted on the SE, path 1 is selected. If no information is available for a specific AID, the default route is path 2. The route information is stored in the routing table in the NFC controller.

### 2.5.2 Different flavors of HCE-based Mobile Payment System

One of the important aspects in the implementation of HCE is the storage location of the credentials. These credentials can comprise PAN or cryptographic keys or tokens. In general, three approaches can be considered. The first approach would be to use the phone memory for storing the credentials. It would be relatively easier to implement but, in terms of security, it would not provide an adequate level of security to protect the credentials. A malicious application could get access to the memory and dump the credentials. The second approach would be to use TEE to store the credentials. This would provide additional security as,

in the case of TEE, it has dedicated secure storage which has restricted access. Since it has its own operating system, even if the rich operating system in the mobile device is compromised it would not affect the data kept in the secure storage of TEE. Different components of the application can be executed in the TEE. For example, the user interface which allows PIN entry can be executed in TEE so that a malicious application cannot sniff the entered PIN. To protect the device from malicious application there are different tools which can identify the presence of malware or detect if the device is rooted. In addition to this, there is an option in the Android operating system which prevents the device installing applications from unknown sources. However, as discussed in section 2.2.3, TEE is susceptible to side channel attacks and it increases the overall cost of the implementation. The third approach would be to use cloud-based storage, in which the credential would be stored on the cloud server. In this approach, the security risk is shifted from the mobile device to the cloud server and the communication channel between them. The cloud-based storage can also utilize tokenization so that actual credentials are never moved out of the cloud storage. The token can be stored in the mobile device which can be utilized for performing the transaction. This token is a surrogate value which is provided by the token service provider to replace the actual card data. This token can have multiple limitations, such as the token can only be used for a limited time period or it can only be used to make payment in limited stores, etc. These limitations help to decrease the overall risk level, if compared to the compromise of actual card data. However, security of the token also depends on the device authentication mechanisms in place. When the mobile device requests a token, the cloud server should verify if the device is a genuine one and is authenticated. This authentication can be done using PIN codes or using derived credentials as discussed in section 2.2.1.

# Chapter 3

# Secure Element and its different flavors

This chapter discusses the different forms of secure storage used in the NFC-based mobile payment environment. A comparison of different forms of SE is performed. This chapter also analyzes the benefits and limitations of each of them. This chapter provides the answer to the first research question (Section 1.4) by providing a detailed comparison between different forms of SE. Figure 3.1 provides an overview of this chapter.

## 3.1 Secure Element

Global Platform[1] defines SE as: "a tamper-resistant platform (typically a one-chip secure micro-controller) capable of securely hosting applications and their confidential and cryptographic data (e.g., key management) in accordance with the rules and security requirements set forth by a set of well-identified trusted authorities" [10]. This platform provides secure storage as well as a secure environment for the execution of the application. It is the same chip which is present in smart cards like credit cards, transit cards, etc. For mobile devices, this chip is present in three different forms:

[1] https://www.globalplatform.org/

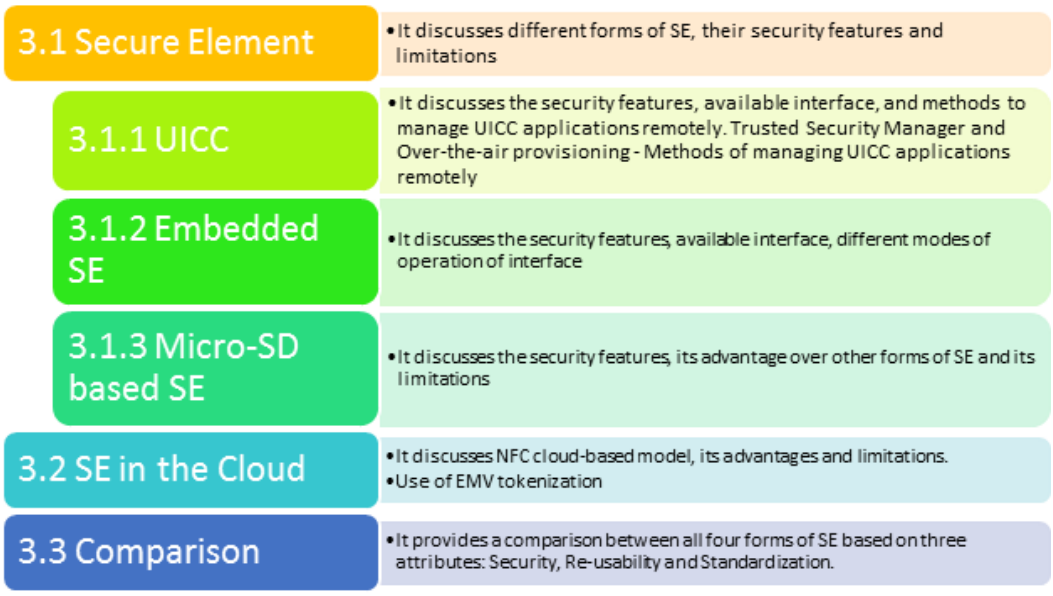


Figure 3.1: An overview of Chapter 3

- UICC
- eSE
- Micro-SD

### 3.1.1 UICC

UICC is the smart card used in Global System for Mobile Communications (GSM)[1] and Universal Mobile Telecommunications System (UMTS) networks[2]. It is provided and controlled by MNOs. It usually contains the UICC application and the corresponding data. This application is stored by MNOs which allow secure access to the data stored in UICC. This data contains credentials for verifying and authenticating itself to the MNOs. This platform is based on the smart card so it has provision for storing multiple applications and the corresponding data. It therefore provides a suitable platform for storing payment applications. It can store the payment applets and the credentials like keys or PIN codes. As

[1] http://www.mobileburn.com/definition.jsp?term=GSM
[2] http://goo.gl/R5LEUq

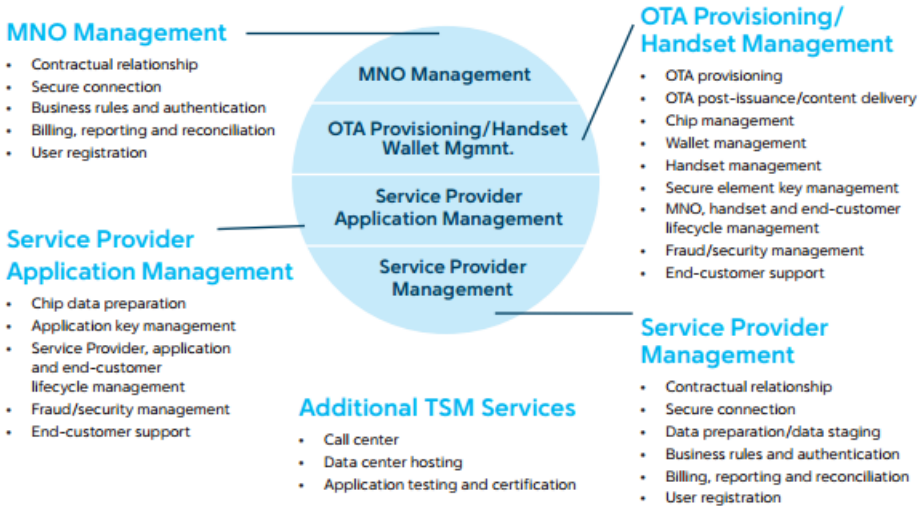


Figure 3.2: Key capabilities of TSM [21]

access to the UICC is controlled by MNOs, the payment applet and credential need to be provisioned by MNOs in coordination with bank/service providers. This is done with the help of TSM or Over-the-air (OTA) update. This restricted access provides a good security level.

#### 3.1.1.1 Trusted Security Manager

In mobile payment, TSM bridges the channel between the financial institutions (banks or service providers) and MNOs. The financial institutions own the payment application which will allow their users to perform banking operations like money transfer, viewing account summaries, etc., using their mobile phones. This payment application and accompanying credentials need to be provisioned securely in UICC. This is done using TSM. Apart from application provisioning, it is also responsible for application key management, SE key management, OTA provisioning, etc [21]. Figure 3.2 shows the different functionalities provided by TSM. In [21], all the functionalities of TSM are explained in detail.

#### 3.1.1.2 Over-the-Air provisioning

OTA is a technology which allows the MNOs to remotely manage the application and data installed on the UICC, without any need for a physical connection. OTA is based on client-server architecture in which the MNOs have a back-end server and UICC acts as a client. The standard OTA architecture[1] involves the following components:

- A backend system owned by MNOs to send the message.
- An OTA gateway is responsible for transforming the message so that it can be understood by UICC.
- The Short Message Service Centre (SMSC) receives the message from the OTA gateway and sends the message to the wireless network. A maximum of 160 alphanumeric character can be sent in a message to a mobile phone. If the mobile phone is switched off or is not in the coverage area, the message is stored and sent to the UICC when the mobile is switched on or is in the coverage area of the network.
- A channel which is present between the SMSC and UICC.
- A mobile phone containing the UICC.

  As discussed in section 3.1.1.1, in the case of TSM, OTA is provided to manage the payment application and credentials remotely. It allows the financial institution to push the software updates or security keys updates directly to the UICC using the TSM platform in a secure manner.

#### 3.1.1.3 Security features and available interface

In UICC, the payment application and credentials are stored within its secure storage space. The mobile phone contains the user interface application which allows the user to perform different tasks like verifying the PIN code and interacting with the data stored in the UICC. This application needs an interface to

[1] `http://www.gemalto.com/techno/ota`

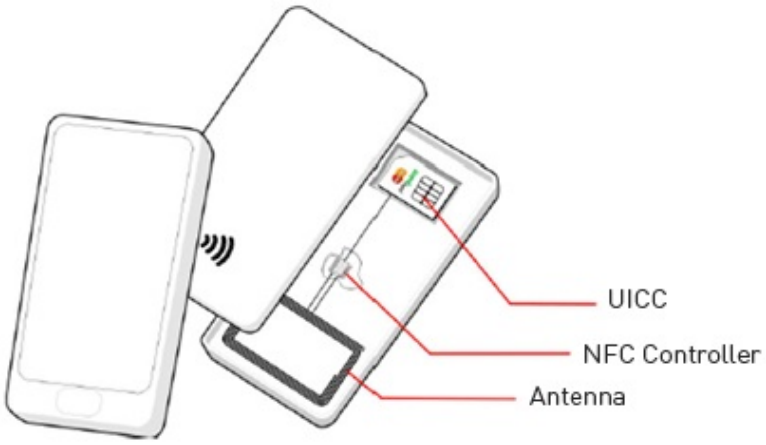


Figure 3.3: UICC-based SE [26]

interact with UICC using the NFC controller. Figure 3.3[1] shows the connectivity between the NFC controller and UICC. The interface used for communication between them is Single Wire Protocol (SWP)[2]. The UICC is connected to the baseband processor, but there is no connection to the application processor. Therefore, access to the application and data on UICC cannot be achieved by any malicious application running on the application processor. The OTA update can be made using the communication channel between the baseband processor and UICC using a different interface i.e. Inter Process Communication (IPC) interface in the form of AT commands[3].

UICC creates a security domain to provision the application of different vendors on the UICC. MNOs install their own application in the main security domain. The main security domain is the Issuer Security Domain (ISD) which is managed by the MNOs. Applications of the service provider (like financial institutions) are provisioned in UICC using TSM. In the case of TSM, multiple security domains are created for each service provider in the UICC. Access to each security domain is restricted using cryptographic keys which are only known

[1] https://mobile.mastercard.com/Partner/MobilePayPass/SecureElements

[2] http://www.sourcemediaconferences.com/CTST09/PDF09/D/Wednesday/OuahsineRaphik.pdf

[3] http://nelenkov.blogspot.nl/2012/08/accessing-embedded-secure-element-in.html

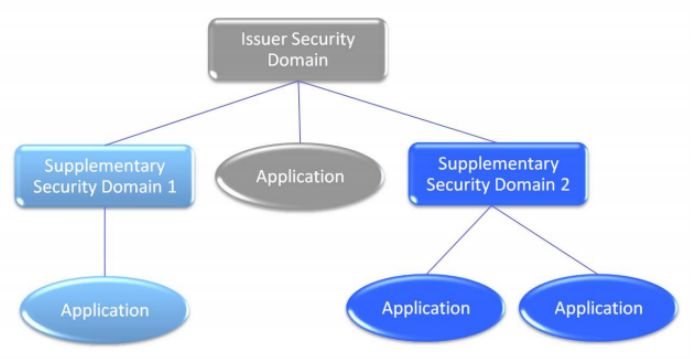


Figure 3.4: ISD and SSD hierarchy as suggested by Global Platform [5]

to MNOs and the service provider. A Supplementary Security Domain (SSD) is created for each service provider so that application and data are kept segregated. Figure 3.4 shows the overall layout of the security domains as suggested by Global Platform [5].

### 3.1.2 Embedded Secure Element

eSE is a smart card which is embedded in the mobile device by the device manufacturer. In comparison with UICC, eSE has access control restrictions set by the device manufacturer. Therefore, the application and credentials need to be provisioned by the device manufacturer. The interface which is available for communication between the NFC controller and eSE is Near Field Communication Wired Interface (NFC-WI) [24]. This interface provides three modes of operation:

- OFF mode
- Wired mode
- Virtual mode

In the *OFF* mode, there is no communication between the NFC controller and the eSE. In the *WIRED* mode, the eSE is visible to the Android operating system as if it is a contactless smart card connected to the NFC reader. In the *VIRTUAL* mode, the eSE is visible to the external reader as if the phone is

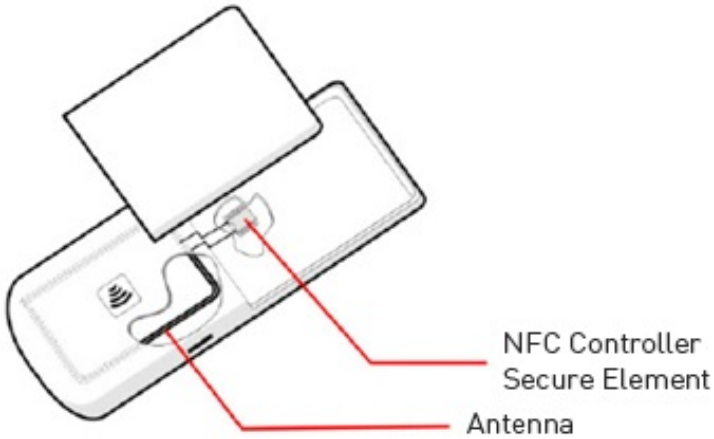


Figure 3.5: Embedded SE [26]

a contactless smart card. These modes are mutually exclusive i.e. two modes cannot be active at a point, so the communication with the eSE is possible either via the contactless interface (e.g., from an external reader) or through the wired interface (e.g., from an Android application). The mode is controlled by the device manufacturer. The *WIRED* mode can be abused by a malicious Android application if it has root privileges. A malicious application with root privilege can also intercept the traffic between NFC controller and eSE. Figure 3.5 shows the architecture of eSE.

### 3.1.3 Micro-SD-based SE

Micro-SD-based SE is a form of smart card which can be plugged into the mobile device into the corresponding slot. It provides a similar level of security compared to a smart card but the access control is either managed by the Micro-SD card manufacturer or the entity, such as a bank, which provisions their application in the Micro-SD card and delivers it to their customer. There is no particular standard suggested for the interaction between NFC controller and SE in the Micro-SD card. This SE removes the dependency on the involvement of MNOs or the mobile device manufacturer to embed their application in the UICC or eSE. The service provider, such as a bank, can use the Micro-SD card to deploy their application within it. However, maintenance of this form of SE is difficult as there

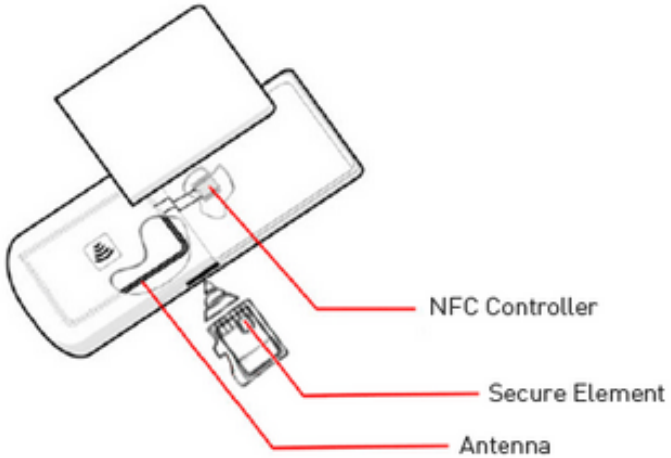


Figure 3.6: MicroSD based SE [26]

are very limited ways of remote maintenance. Figure 3.6 shows the architecture of the Micro-SD based SE.

## 3.2 SE in the Cloud

The limitations of the different forms of SE discussed in Section 3.1 forced companies to look for another location to store the credentials and payment applications. SE in the cloud is one of the solutions which removes the dependency on different parties like MNOs or device manufacturers and allows the card issuer or service provider to completely manage the payment application and the corresponding data. The SE in the cloud is also termed as *Virtual SE*.

One of the cloud-based SE models was discussed in [30]. In this paper, the author suggests an NFC cloud wallet model. The sensitive data is stored in the cloud server which can be managed by the financial institution. In another scenario, the financial institution can partner with the cloud service provider to manage the cloud infrastructure on their behalf.

When the NFC-enabled mobile device is placed close to the POS terminal, the sensitive data and application are downloaded from the cloud and stored in the temporary storage in the mobile device. The required data is transferred to the POS terminal to perform the transaction. Once the transaction is complete, the sensitive data downloaded from the cloud is removed from the mobile device. The POS terminal also connects to the cloud server to verify the credentials and other transaction data.

Some advantages of this approach are that it is easier for the financial institution to manage the customer credentials and payment applications. It also makes it easier to manage virtual SE in the cloud as the storage space in the cloud is quite large compared to the other forms of SE like UICC, eSE, etc. which has limited storage space.

Some of the limitations of this approach are that it needs Internet connectivity while making the payment which makes it little challenging for locations which have limited or no connectivity. It adds latency in the transaction time as it rely on the Internet speed which might not be consistent. The device needs to authenticate to the cloud before it can receive the credentials. As the communication between the device and the cloud is over public Internet, it is susceptible to attacks if not implemented correctly.

Another approach which was suggested by EMV[1] is tokenization. The tokenization is already explained in Section 2.2.2. The benefit of this approach is that the actual card data never leaves the cloud virtual SE. Limited use tokens are transferred and stored in the mobile device's temporary storage. These tokens are used to perform the transaction. Some of the limitations of this approach include that it is still challenging to provide proper device authentication. Adequate protection of the tokens stored in the mobile phone temporary storage is also required.

[1] `https://www.level2kernel.com/emv-guide.html`

## 3.3 Comparison of the different forms of SE

In [3], the author has provided a detailed comparison between different forms of SE i.e. UICC, Micro-SD-based SE and eSE. The author uses the following key attributes for providing the comparison: Security, Re-usability and Standardization. Security corresponds to the availability of secure storage and restricted access control. Re-usability means the ease of using the mobile payment solution when the mobile device or SE in the device is changed. Standardization refers to the availability of standards for the communication between the NFC controller and SE, storing and accessing data in SE.

UICC-based SE provides very good security as access to the secure storage containing the payment application and credentials is restricted by MNOs using cryptographic keys. It is also tamper-resistant so that the cryptographic keys cannot be extracted from UICC. It uses security domains to deploy applications from the third parties. Access to the security domains is controlled using cryptographic keys which are unique to each domain [13]. In terms of re-usability, UICC-based SE makes it easier for the consumer[1] to switch to a different mobile device. The UICC can be easily removed from one mobile device and plugged into another. The application and credentials are stored in the UICC, so the consumer will only need to install the user interface application on the new mobile device. If the UICC is stolen and the consumer receives a new UICC, the application and credentials will have to be installed again. It is also possible that the consumer switches to a new MNOs which would mean the application and credentials cannot be moved to the new UICC. This effects the re-usability attribute of the UICC-based SE. In terms of standardization, the UICC-based SE has been available in the market for some time and multiple standards are available. SWP is a standard interface for interaction between NFC controllers and UICC. The International Organization for Standardization (ISO)-7816[2] standard is used for interaction between application processors and UICC.

[1] Consumer is referred to the user of the mobile device and payment application
[2] `http://www.iso.org/iso/catalogue_detail.htm?csnumber=54550`

eSE provides similar level of security as UICC. The main difference is that with eSE the access control is managed by the mobile device manufacturer while in UICC it is managed by MNOs. eSE also uses cryptographic keys to restrict access to the application and credentials stored. It is also tamper-resistant which provides protection against any leakage of cryptographic keys. In terms of re-usability, it is not good as the eSE is embedded into the mobile device and cannot be removed. Therefore, if the consumer changes the device, the application and credentials have to be installed again in the eSE of the new mobile device. In terms of standardization, eSE has some standards available. NFC-WI is one of the standards accepted for communication between the NFC controller and eSE. However, different mobile device manufacturers might use some other proprietary standard for the same. It will take some time for different mobile device manufacturers to accept one standard interface.

Micro-SD-based SE is comparatively less secure than UICC and eSE. There is no standard approach defined for managing access control for the application and credentials stored in Micro-SD-based SE. Financial institutions can deploy the application and credentials on the Micro-SD and develop their own access control mechanism[1]. In terms of re-usability, Micro-SD is better when compared to UICC and eSE. If the mobile device is changed, the Micro-SD can be moved from the old mobile and can be installed in the new mobile device. The consumer will only be required to install the user interface application on the new mobile device. If the Micro-SD is lost or stolen, the financial institutions can provide a new Micro-SD with the application and credentials pre-installed on it. Micro-SD does not depend on the change of mobile device or MNOs. In terms of standardization, different Micro-SD manufacturers provide support for different interfaces. For example, SD Association uses the SWP interface for communication between the NFC controller and Micro-SD [17]. Micro-SD supports different interfaces and standards but there is a lack of a common approach used by different Micro-SD manufacturers.

As a result of this thesis, the comparison of virtual SE or SE in the cloud

[1]access control mechanism refers to the security mechanism used to protect the application and credentials from unauthorized access

| Criteria / SE alternatives | Security | Reusability | Standardization | Total |
|---|---|---|---|---|
| **SE in the Cloud** | + | ++ | + | ++++ |
| **Embedded SE** | ++ | - | + | ++ |
| **MicroSD based SE** | + | ++ | + | ++++ |
| **UICC** | ++ | + | ++ | +++++ |

**Note:+ means Satisfactory ++ means Good - means Not Good -- means Bad**

Figure 3.7: Comparison of different forms of SE [3]

is presented based on the same three attributes. In terms of security, virtual SE is considered to be secure as the sensitive data is stored on the cloud server. The security relies on the implementation of controls to restrict access to the cloud servers. The communication channel between the cloud server and the mobile device is susceptible to attacks. Therefore, it is important to secure the data while in transit. Authentication of the device is also important as access to the cloud server needs to be verified. In terms of re-usability, virtual SE is comparatively easier to configure when the device is changed, The mobile device has very limited information stored locally and all the sensitive information is stored on the cloud server and it is therefore easier to configure and switch to a new device. In terms of standardization, some standards are available i.e. usage of SSL/TLS[1] for communication between the mobile device and the cloud=based SE, EMV tokenization for storing alias of card data in the mobile device and API[2]-based access. Figure 3.7 provides a summary of the comparison between different forms of SE. Overall, UICC is better when compared to other forms of SE. This is also reflected in the current landscape of the NFC-based mobile payment system. A UICC-based payment system is the most prevalent approach. After UICC, the most prevalent approach is SE in the cloud which uses HCE which is discussed in detail in Chapter 4.

[1] `http://www.sans.org/reading-room/whitepapers/protocols/ssl-tls-beginners-guide-1029`

[2] `http://www.quora.com/What-is-an-API`

# Chapter 4

# Host-based Card Emulation

This chapter discusses different security mechanisms and attacks vectors which are applicable for the HCE-based mobile payment system.

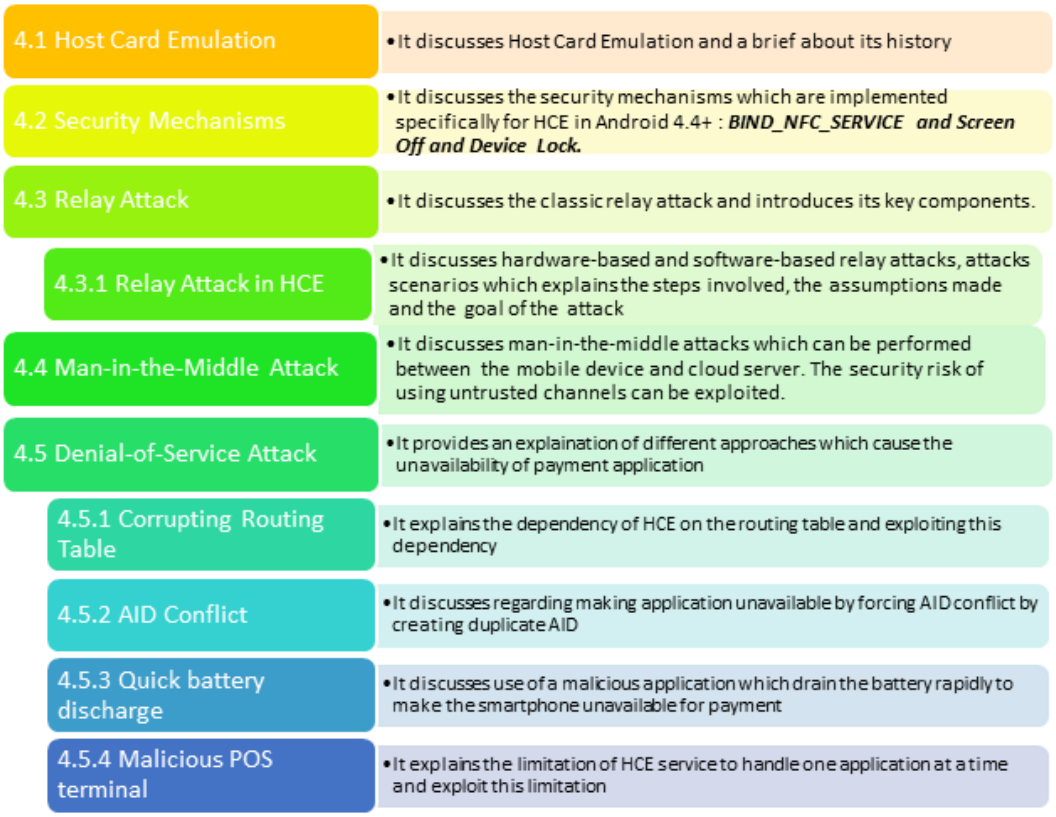


Figure 4.1: An overview of Chapter 4

This chapter discusses the different attack scenarios which are based on our analysis of the literature and documentation available. The attack scenarios discussed are based on the attack vectors identified by us, as part of this thesis (Only one of them, a hardware-based relay attack on HCE has been tested). This chapter provides answers to the second research question mentioned in Section 1.4. Figure 4.1 provides an overview of this chapter.

## 4.1 Host Card Emulation

HCE is the card emulation mode in which the functionality of a smart card is emulated without any need for hardware-based SE. The term HCE was first coined by the founders of SimplyTapp, Inc., and describes the ability of the mobile device to act as a smart card and use a channel to communicate between the contactless POS terminal and a remotely-hosted SE containing financial payment card data [18]. SimplyTapp implemented this new technology on the Android operating system[1]. As well as Android, Blackberry implemented HCE in its operating system and later Microsoft announced support for HCE in Windows 10[2]. With the introduction of HCE in Android 4.4, multiple entities such as payment processors VISA and MasterCard started supporting HCE technology [27] [23]. The details of the implementation of HCE in Android 4.4 are explained in Section 2.5.1.

## 4.2 Security Mechanisms

HCE removes the dependency on hardware-based SE for implementing NFC-based mobile payment systems. The security of this implementation relies on the security permission model of Android[3]. Therefore, Android has implemented the following additional security mechanisms which are specific to HCE:

- *BIND_NFC_SERVICE* is a system permission which binds with the HCE service. This permission allows only the operating system to communi-

[1] `http://en.wikipedia.org/wiki/Host_card_emulation`
[2] http://www.nfcworld.com/2015/03/25/334722/windows-10-for-mobile-gets-hce/
[3] `http://developer.android.com/guide/topics/security/permissions.html`

cate with the HCE service. This ensures that all the APDUs received by the HCE service come from the operating system, which is received from the NFC controller. In addition, it ensures that the response APDUs are sent back to operating system which are then sent to the NFC controller. This permission provides protection against any sniffing of the APDUs by a malicious application which does not have the required permission.

- *Screen Off and Device Lock* is another security feature implemented in Android. The NFC controller and the application processor are turned off when the device screen is off or is in a locked state[1]. Therefore, the HCE service will not work in such case. This protects the device from skimming and unintended interaction with an unknown device without the device owner's knowledge. By default, the HCE service will work if the device screen is in the locked state as the *android:requireDeviceUnlock* attribute is set to false. If it is set to true, the device needs to be unlocked to enable the HCE service. This attribute is set in the apduservice.xml file.

## 4.3 Relay Attack

In the payment system environment, a relay attack can be considered as one of the prevalent attack vectors because of its simplicity, its effects and its ability to circumvent the cryptographic protections. The standard architecture of any relay attack will comprise the following four components:

1. *Target Device*: This device can be a physical contactless card or a mobile phone with card emulation. One of the assumptions about this target device is that it is physically far away from the POS terminal and is the legitimate holder of the credentials.

2. *Mole*: This device is a NFC-enabled computing device which is kept in close proximity to the target device. It is under the attacker's control. It can be in the form of a hardware-based computing device like a mobile device which can act as a NFC reader, or an open-source electronic prototyping platform

[1] `https://developer.android.com/guide/topics/connectivity/nfc/hce.html`

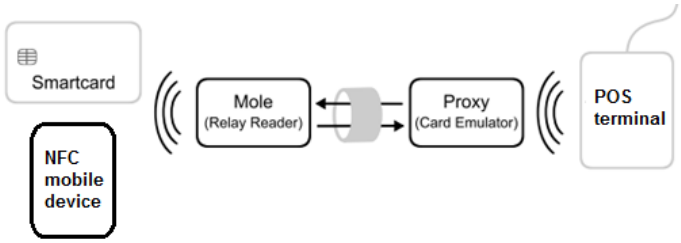


Figure 4.2: Standard architecture of relay attack [28]

like Arduino or Raspberry pi with NFC shields. It can be in the form of software (malware), which is installed in the target device itself. This software needs root privilege on the target device to relay the information between proxy and emulated card.

3. *Proxy*: This device contains an application which acts as a proxy between the target device and POS terminal. This device can be a NFC-enabled mobile device with card emulation. It is also under attacker control.

4. *POS terminal*: This device is a NFC reader which provides an interface to the bank payment gateway. The attacker has no control over it.

Apart from the four components discussed above, there is a communication channel between Mole and Proxy. This communication channel can be wired or wireless. This channel is also under the control of the attacker. Figure 4.2 shows the overall architecture of a standard relay attack.

In a relay attack, the *Mole* is kept close to the *Target device* and the *Proxy* is kept close to the *POS terminal*. The APDU from the POS terminal is received by the proxy, which is then forwarded to the mole. The mole sends the APDU to the target device and gets the response APDU. This APDU is sent back to the POS terminal via proxy. The Mole pretends to be the POS terminal to the target device and Proxy pretends to be the target device to the POS terminal.

### 4.3.1 Relay Attack on HCE-based Mobile Payment System

Relay attacks have been performed on SE-based mobile payment systems such as Google Wallet[6]. As a part of this thesis, the relay attack is extended to an HCE-based mobile payment system. The different types of relay attacks and attack scenarios discussed are based on the assumptions and motivations discussed in further sections.

In the case of HCE in Android, the bank application implements the HCE service which is responsible for interacting with the NFC controller. The HCE service binds with BIND_NFC_SERVICE which ensures that only the system can bind to the HCE service. This allows the HCE service to interact with the operating system which in turn interacts with the NFC controller. The communication between the NFC controller and NFC reader is done using APDUs as defined in the ISO/IEC 7816-4 specification.

#### 4.3.1.1 Hardware-based Relay Attack

In the hardware-based relay attack, the *Mole* is a hardware device which act as a NFC *POS* terminal to the *Target device*. The hardware device has NFC capabilities.

To perform a hardware-based relay attack, it is necessary that the *Mole* is in close proximity to the *Target device*. Android 4.4 has introduced some security measures which will prevent the *Mole* communicating with the *Target device* despite being in close proximity to it. The NFC controller and the application processor are turned off completely when the screen of the device is turned off. Therefore, the HCE service will not work when the screen is off. However, if the device screen is on as well as locked, HCE services will still function normally. This is controlled by the *android:requireDeviceUnlock* attribute in the *host-apdu-service* tag of the HCE service. By default, device unlock is not required as the HCE service will be invoked even if the device is locked.

This additional security measure requires the device screen to be switched on

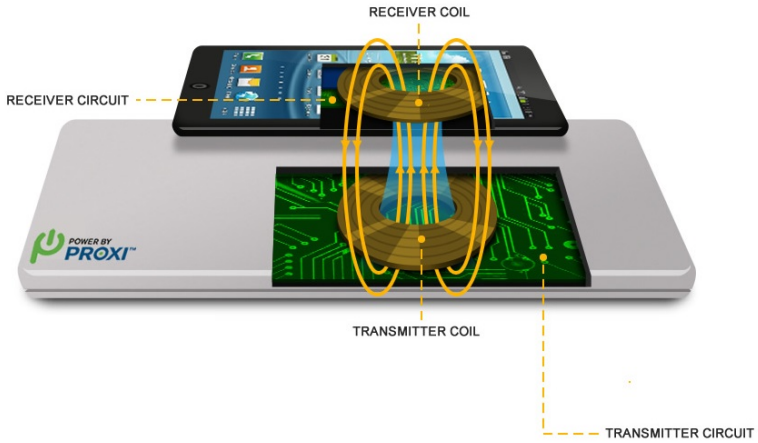


Figure 4.3: Wireless Charger

while the attacker needs to skim the *target device*. Usually, an attacker will want to skim the *target device* when the user is not using the *target device*. For example, when the *target device* is inside the user's pocket or is left unattended. There are some attack scenarios where these security measures can still be bypassed.

#### 4.3.1.2 Attack Scenario

In this section, one of the attack scenarios will be discussed in which the additional security measures mentioned in the previous section can be bypassed.

- *Wireless Charger as a Mole:* Wireless chargers allow the mobile device to charge its battery without any need to physically connect the device to a cable. Figure 4.3[1] shows one of the wireless chargers from *Proxi*. When a mobile device is kept on the wireless charger platform, the device screen is enabled for a small time interval showing the notification that the device is charging. The *Mole*, which is in the form of Arduino with NFC, is transplanted inside this wireless charger and can utilize the small time interval when the device screen is active to relay the APDUs to the POS terminal. This customized wireless charger can be placed in open environments like Airport terminals, libraries, coffee shops, etc. The transaction flow is as follows:

[1] `http://powerbyproxi.com/wireless-charging`


Figure 4.4: Wireless Charger as a mole for relay attack

- The attacker keeps the *Proxy* connected to the customized wireless charger using WiFi or Bluetooth.
- The user keeps the *target device* on the customised wireless charger which act as a *Mole*.
- The *Mole* sends the *SELECT AID* APDU to the *target device* as soon as the device screen is active.
- The HCE service corresponding to the AID will handle the request and send the response APDU back to the *Mole* which sends it back to the *Proxy*.
- The attacker places the *Proxy* close to the *POS terminal* which processes the response APDU. Some *POS terminals* perform the initial challenge response protocol to authenticate the card. In such cases, the data used for challenge response can be relayed as well.

Figure 4.4 shows the overall architecture of this attack scenario.

A similar approach can be set up using a phone holder in cars used while functioning as a navigation device or wireless music dock stations. In all these scenarios, the device screen will be turned on and the phone will be to close proximity with the *Mole*.

**Assumptions** The assumptions in the hardware-based relay attack using a wireless charger is as follows:

- The wireless charger is controlled by the attacker. It can monitor when the target device is kept on the wireless charger.

- The target device has the NFC option enabled and has a payment application installed.
- The options in the payment applications is configured such as PIN code entry is not required for each transaction.

**Motivation** The main motivation behind this attack scenario is to perform a relay attack even if the phone has protection mechanisms like turning the NFC controller *OFF* when the device screen is inactive or locked. This attack scenario shows the flaw in the design of the implementation of the security mechanism mentioned before. The NFC controller is turned *ON* when the device screen is active. It does not verify which event is turning the device screen *ON*. It does not verify if the device screen is turned *ON* manually by pressing the power button or automatically when the device is attached to a charger or when it receives any notification.

#### 4.3.1.3 Software-based Relay Attack

In a software-based relay attack, the *Mole* is a piece of software which needs to be present in the *target device*. This software is assumed to have system privileges. The HCE service completely relies on the Android permissions for preventing the APDUs from being intercepted by another application. However, if the malicious application i.e. *Mole* has system privileges, it can intercept the APDUs between the HCE and NFC controller. In normal scenario, the APDUs are received by the NFC controller and handed to the operating system to send them further to the correct HCE service. With malicious software, i.e. *Mole*, having root access can forge this and relay the APDUs received from the *proxy* device to the HCE service. This service can process these APDUs and send the response APDUs back to the *Mole*.

#### 4.3.1.4 Attack Scenario

In this section, an attack scenario is discussed for a software-based relay attack. Figure 4.5 shows the overall setup. The steps involved in this attack are as follows:

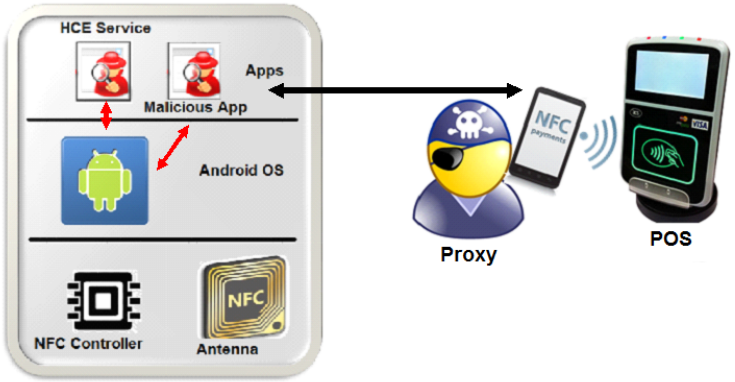


Figure 4.5: Software-based relay attack

- The malicious software i.e. *Mole* is installed on the device. *Mole* is connected to the *Proxy* device using a wireless channel.
- The *Proxy* device is kept in close proximity to the *POS Terminal*. The APDUs sent from the *POS terminal* are relayed to the *Mole* using the wireless communication channel via *Proxy*.
- The APDUs received by the *Mole* are sent to the HCE service. This HCE service will not be able to distinguish if these APDUs are received via the NFC Controller or not. As the *Mole* has system privileges, it can interact with the HCE service despite this service being bound to NFC_BIND_SERVICE. The HCE service trusts the Android permissions which ensures that the APDUs are received from the NFC controller but with system privileges this trust level is bypassed.
- The response APDUs are sent back to the *POS terminal* via *Mole* and *Proxy*.

The HCE service runs in the background which allows the *Mole* to communicate with it when the user is interacting with the *target device*. If the payment application does not provide any kind of notification to the user when the HCE service is active or when the transaction is processed, the user will not even know that a payment transaction has been completed using the payment application

on the *target device*. If there is a notification in place, then it can be blocked by the *Mole* so that the user does not get any information about the fraudulent transaction.

**Assumptions** The following assumptions are made in a software-based relay attack:

- The malicious application i.e. the *Mole* is installed in the *Target device*. This application has the needed system privilege. The *Mole* can be a payment application which can directly communicate with the HCE service or it can be a separate application which emulates the NFC signals and can interact with the genuine payment application, which in turn interacts with the HCE service.
- The payment application does not need any manual user action to complete the payment transaction.
- The options in the payment applications is configured such as PIN code entry is not required for each transaction.
- The attacker has control over the *Mole* installed on the *Target device*.

**Motivation** The motivation behind this attack scenario is to perform a relay attack without needing a physical device to skim the target device. A skimming device, or the presence of the attacker, is not required. With attacker controlled software on the target device, the relay attack can be performed with ease.

## 4.4 Man-in-the-Middle Attack

A man-in-the-middle attack is an attack where the attacker secretly relays and possibly modifies the communication between two parties who believe they are directly communicating with each other[1]. One of the most common implementations of HCE is observed together with cloud-based storage, i.e. SE in the cloud.

[1] `http://www.veracode.com/security/man-middle-attack`

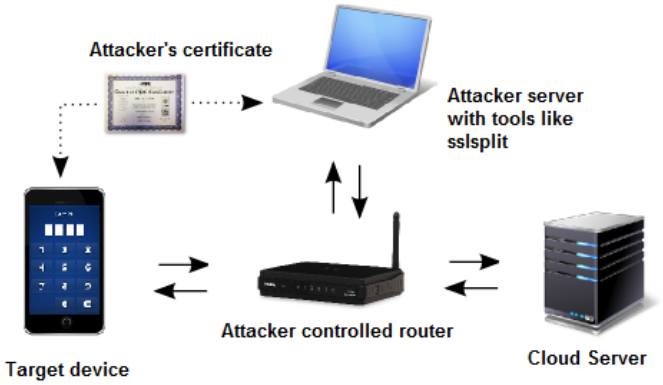


Figure 4.6: Man-in-the-middle attack

In this approach, the card data along with other credentials are stored on the cloud server. When the user needs to initiate a transaction, either the credentials or limited use tokens are transferred to the mobile device. To receive the credentials or tokens, the device needs to be authenticated to the cloud server. This authentication can be done using PIN codes or using certificate-based access. The mobile device needs to communicate with the cloud server for authenticating itself over the untrusted channel. The security risk of using this untrusted channel can be overcome by using Secure Socket Layer/Transport Layer Security (SSL/TLS) protocol. This channel is still susceptible to a man-in-the-middle attack if the SSL/TLS protocol is not correctly implemented. For example, the certificate is self-signed or signed by the trusted Certificate Authority (CA) which the server accepts. One possible attack scenario is explained as follows:

- The target device connects to the open WiFi which is under the control of the attacker. One of the assumptions is that the target device has added the certificate provided by the attacker to its trusted certificate store or the application is not verifying the certificate.
- The attacker has set up the WiFi connection along with tools like sslsplit which will act as a proxy between the target device and the cloud server. The target device is connected to the server running sslsplit and this server pretends to be the actual cloud server to the target device. On the other

side, the server running sslsplit is connected to the cloud server and pretends to be the target device to the cloud server.

- The traffic is proxied between the target device and the cloud server by sslsplit and all the traffic can be sniffed in clear text. This will allow the attacker to gain access to the credentials like PIN codes or tokens.

Figure 4.6 depicts the overall organization of the set up for this attack scenario.

**Assumptions** The assumptions behind this man-in-the-middle attack are as follows:

- The target device is connected to the WiFi controlled by the attacker.
- The cloud server accepts the certificate provided by the attacker.
- The target device has installed the certificate provided by the attacker by allowing certificates to install from local storage.

**Motivation** The motivation behind the man-in-the-middle attack discussed above is to intercept the traffic between the payment application installed on the target device and the cloud server. The intercepted traffic might lead to the leakage of credentials like PIN code, information related to tokens, etc.

## 4.5 Denial-of-Service Attack

A denial-of-service attack is an attempt to make a machine or network resource unavailable to its intended users[1]. The availability of the payment application relies on the capability of the HCE service to identify and route the APDUs to the correct application. If this capability is affected, the correct payment application might become unavailable.

[1] `https://www.incapsula.com/ddos/ddos-attacks/denial-of-service.html`

### 4.5.1 Corrupt Routing Table

The HCE service relies on the routing capability of the NFC controller to map the correct AID to the corresponding HCE service. When an application is installed on the mobile device, its corresponding AID is registered to the routing table of the NFC controller. When the POS terminal sends SELECT AID APDU, the NFC controller uses its routing table to map the AID to the corresponding application. It might be possible for a piece of malware with root privileges to gain access to this routing table and delete or corrupt the entries in it. This way the correct application cannot be determined by the NFC controller and APDUs cannot be delivered to the corresponding HCE service. This will cause unavailability of the application to the POS terminal.

**Assumptions** The following assumptions are made for the corrupting routing table approach:

- The malicious application has the needed system privileges.
- It has access to the location where the routing table is stored and can modify the entries in the routing table

**Motivation** The goal of this approach is to make the payment application unavailable to the POS terminal. This approach will exploit the dependency of the availability of payment application on the routing table, which acts as a single point of failure.

### 4.5.2 AID conflict

The denial-of-service attack can be performed by not allowing the genuine payment application[1] to interact with the POS terminal. In other words, the HCE service will route the traffic to the malicious application (which is under the

[1]Genuine payment application refers to the application which is provided by the known service providers like Google Wallet

control of the attacker) instead of the genuine payment application. In this approach, a malicious payment application will be installed on the target device. This application will have a duplicate AID which will be equal to the AID of a genuine payment application installed on the same device. When the NFC controller receives SELECT AID APDU, there will be an occurrence of AID conflict. The HCE will therefore not be able to decide which application it should select. The Android platform resolves AID conflicts depending on which category an AID belongs to[1]. An AID can belong to *payment* or *others* category. In the *payment* category, the user needs to select the default payment application in the Android settings interface. If there is no default application selected by the user, the conflict will not be resolved and the genuine payment application will not be available to communicate with the POS terminal. It might also be possible to select the malicious application as a default payment application. This will also lead to the unavailability of the genuine payment application. In the case of AID belonging to the *others* category, the user will always be prompted to choose the correct application.

**Assumptions** The following assumptions are made for this approach:

- The mobile device has installed a malicious application which has the same AID when compared to a genuine payment application.
- The malicious application is selected as a default payment application in the NFC settings of the device.

**Motivation** The main goal of this approach is to make the genuine payment application unavailable by causing AID conflict. This can be considered as a target attack where payment application with a specific AID can be made unavailable and prevents the transaction from being completed. If the appearance and the user interface of the malicious application resemble the genuine payment

[1] `https://developer.android.com/guide/topics/connectivity/nfc/hce.html#AidConflicts`

application, the user can be tricked and can also leak sensitive information such as the PIN code.

### 4.5.3 Quick Battery Discharge

This approach targets the fact that NFC-based mobile payment applications rely on the battery of the mobile device. As long as the battery is alive, payments can be done but if the battery is drained, the payment application will no longer be available. In this approach, a malicious application is installed on the target device which consumes high Central Processing Unit (CPU) resources and causes the battery to drain quickly. The NFC antenna needs energy from the battery to communicate with the POS terminal. If the battery is drained quickly by the execution of the malicious application, the payment application will no longer be available to start a transaction. Consequently, the dependency of the NFC antenna on the battery can be exploited to cause a denial-of-service attack.

**Assumptions** The following assumptions are made for this approach:

- The malicious application installed on the device runs in the background and does not require any user interaction.
- If the mobile device is restarted, the malicious application is automatically restarted.

**Motivation** The motivation behind this approach is to exploit the dependency of the power source for the NFC antenna solely on the battery of the mobile device. This dependency is also a single point of failure.

### 4.5.4 Malicious POS terminals

In this approach, there is an additional NFC terminal (which is under an attacker's control) which is kept in close proximity to the actual POS terminal. As mentioned earlier, the HCE relies on the AID to choose the corresponding application. This approach tries to create confusion for the HCE service by continuously changing the AID value. This will create a hindrance for an application


Figure 4.7: Malicious POS terminal causing DoS attack

to complete the transaction.

The mobile payment application implements the HCE service which responds to the APDUs received from the POS terminal. Each application implements the HCE service and registers a unique AID for each application in the *apduservice.xml* file. This xml file helps the NFC controller to route the received APDUs to the corresponding HCE service. The correct HCE service is selected using the SELECT AID APDU sent by the POS terminal. Once the correct HCE service is selected, the NFC controller will continuously send further APDUs to the same HCE service unless it receives another SELECT AID APDU. If a malicious NFC reader can be kept physically close to the POS terminal and can continuously broadcast SELECT AID APDUs with random AID values, it will cause the NFC controller to stop sending the genuine APDUs from the POS terminal to the corresponding HCE service. Therefore, it will cause a denial-of-service attack.

As shown in figure 4.7, the malicious NFC reader is kept close to the POS terminal. This malicious NFC reader is broadcasting the SELECT AID APDUs with random AID values. For example, it sends SELECT APDU F0394148148100. The HCE service corresponding to the AID F0394148148100 is selected. After some time interval, another SELECT APDU F0394148148200 is sent. This will force the HCE service corresponding to AID F0394148148200 to be active and HCE service corresponding to AID F0394148148100 will no longer be available.

The malicious NFC reader can also be embedded inside the POS terminal. For example, it might be more practical to keep this NFC reader in the transit station POS terminal.

**Assumptions** The following assumptions are made for this approach:

- The malicious NFC reader is kept in close proximity to the POS terminal in a deceptive way so that it cannot be identified.
- The mobile device has at least two applications implementing HCE and their AID is known to the attacker. The malicious NFC reader will continuously broadcast SELECT AID APDU with the value of AID changing at a regular interval. These AIDs correspond to the applications installed on the mobile device as mentioned earlier.

**Motivation** The goal of this approach is to interrupt the transaction flow of one application. This approach exploits the feature that only one application implementing HCE can be active at a time. If there are multiple applications installed on the mobile device which implements HCE, they cannot perform transactions simultaneously.

## 4.6 Summary

Relay attack is one of the attacks discussed in this chapter. The security mechanisms implemented in Android specifically related to HCE can be bypassed by the hardware-based relay attack. With the close proximity of a *Mole* i.e. a wireless charger, it is possible to enable the display screen of the mobile device. A software-based relay attack is also promising as it removes the need for a hardware *Mole*. However, its challenge is to trick the user into installing the malicious software and to gain necessary system privileges to interact with HCE. Countermeasures like distance or time bounding protocols can prevent relay attacks but it is difficult to implement them in an effective way. A Man-in-the-middle attack is also a promising attack as it can allow the attacker to intercept the traffic between the mobile device and the cloud-based SE. However, it depends

upon the implementation of SSL/TLS. This attack might lead to the leakage of PIN codes or tokens, etc. Denial-of-Service is another attack which is not very promising in the case of HCE-based mobile payment, as it has limited effect. For example, with an AID conflict, the application with specific AID can be targeted. In addition to this, the attack relies on the user's choice of selecting the malicious application as a default payment application.

# Chapter 5

# NFC SIM Card

This chapter discusses the Vodafone NFC SIM card which enables Vodafone customers to make NFC-based mobile payment. We provide a comparison between UICC and NFC SIM cards. In addition, we propose two attack vectors based on the analysis of the information currently available. This chapter provides answer for the third research question mentioned in Section 1.4. Figure 5.1 provides an overview of this chapter.

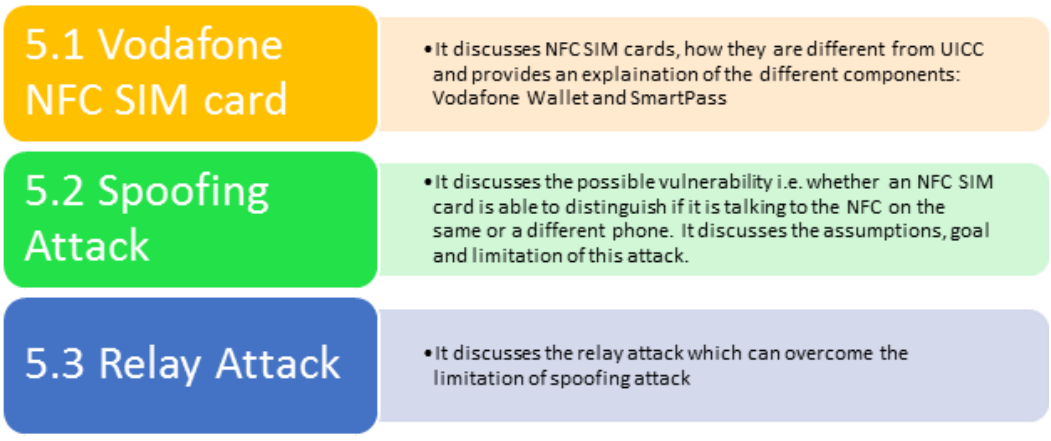


Figure 5.1: An overview of Chapter 5

## 5.1 Vodafone NFC SIM card

Vodafone recently launched an NFC-based mobile payment solution in the Netherlands, which allows Vodafone customers to make payments using Vodafone Wallet[1], Vodafone SmartPass[2] and an NFC SIM card[3]. In this SIM card, the NFC chip is embedded within the same card, which allows a mobile phone with NFC to communicate with the NFC chip on the SIM card. This channel provides an additional way to communicate with the SIM card. In the normal SIM card, the interface allows communication between the baseband processor and the application on the SIM. The interface between baseband processor and SIM card uses SWP which is discussed in Section 3.1.1.3. This interface allows MNOs to manage the applications on the SIM, such as provide OTA updates to the applications. This interface does not allow any application running in the application processor to communicate with the SIM card. On the other hand, in an NFC SIM card the application running in the application processor can communicate with the embedded NFC chip in the SIM card to access the data stored in the SIM card storage.

### 5.1.1 Vodafone Wallet

Vodafone wallet is a digital wallet application which stores the alias of the card in the secure storage of the SIM card. It currently supports VISA cards which allows the user to add the VISA card into the wallet and make contactless payments with the use of the NFC SIM card. The wallet is protected with a PIN code.

### 5.1.2 Vodafone SmartPass

Vodafone SmartPass is a prepaid account in which a user can load or recharge money to make the payment. The user can add a debit or credit card to the

[1] `https://www.vodafone.nl/shop/mobiel/abonnement/extra-opties/apps/wallet.shtml`

[2] `https://www.vodafone.nl/shop/mobiel/abonnement/extra-opties/apps/smartpass.shtml`

[3] `http://www.vodafone.com/content/index/media/vodafone-group-releases/2015/contactless-cardpayments.html/`

SmartPass account. Before making the payment, the user needs to add money into the SmartPass account from the added cards or else from the Vodafone wallet. The user can make contactless payment using the NFC SIM card. No PIN code is needed for making payments of up to 25 euros. Any POS terminal with NFC capability can also communicate with the Vodafone NFC SIM card without any interaction with SmartPass running in the mobile device[1].

## 5.2 Spoofing Attack

A spoofing attack refers to an attack in which the NFC SIM card is tricked that it is communicating with the mobile device containing the same NFC SIM card. The SmartPass account is a prepaid account which allows users to recharge, see their transaction history and change the PIN. This prepaid account can be recharged using bank transfer, debit or credit card or voucher. The user can add a debit or credit card and authorize by verifying the VISA 3D Secure password. Once it is verified, the alias information is stored in the NFC SIM card in the Vodafone Wallet. A user can then add money to the prepaid account from the verified VISA debit card stored in the wallet. It is unclear whether the SmartPass account is linked to the VISA debit card stored in the wallet.

The SmartPass application installed on the phone communicates with the NFC SIM card using the NFC of the phone[2]. It is also unclear whether the NFC SIM card is able to distinguish if it is communicating with the NFC chip on the same phone or an NFC chip on a different phone. The same phone refers to the one containing the NFC SIM card. If it is not able to distinguish, it is a possible vulnerability which can be exploited by the attack scenario discussed in the Section 5.2.1

[1] https://vodafonesmartpass.com/reg/static/html/nl/faq.html#3-9
[2] https://vodafonesmartpass.com/reg/static/html/nl/faq.html#3-7

### 5.2.1 Attack Scenario

This attack scenario is based on the analysis of the information currently available. In this scenario, an attacker might be able to add money to his SmartPass account up to 25 euros using another user's wallet.

The details of this scenario are as follows:

- There are two NFC mobile phones A and B with Vodafone SmartPass installed on each of them. Phone A has an NFC SIM card. Phone A is the target device and the attacker has no control over it. Phone B is the attacker's device and is under his control.
- Mobile phone A has Vodafone wallet in its NFC SIM card. Mobile phone B does not have Vodafone wallet. It needs to add money into the SmartPass account using the Vodafone wallet stored in A's NFC SIM card.
- Mobile phone A and B are kept close to each other so that there is a physical contact between them. The NFC is enabled in mobile phone B while it is disabled in mobile phone A.
- The User of B, i.e. the attacker, logs into the SmartPass account and taps on "Add money" to the prepaid account. It discovers the wallet stored in the NFC SIM card of mobile A. The communication is between the NFC in mobile phone B and NFC SIM card in mobile phone A.

Figure 5.2 explains the attack scenario in detail.

### 5.2.2 Assumptions

To exploit the vulnerability discussed above, some of the assumptions are as follows:

- The mobile phone A with the NFC SIM card has turned its NFC off.
- The NFC SIM card has a registered VISA debit card whose information (alias) is stored in the NFC SIM card.

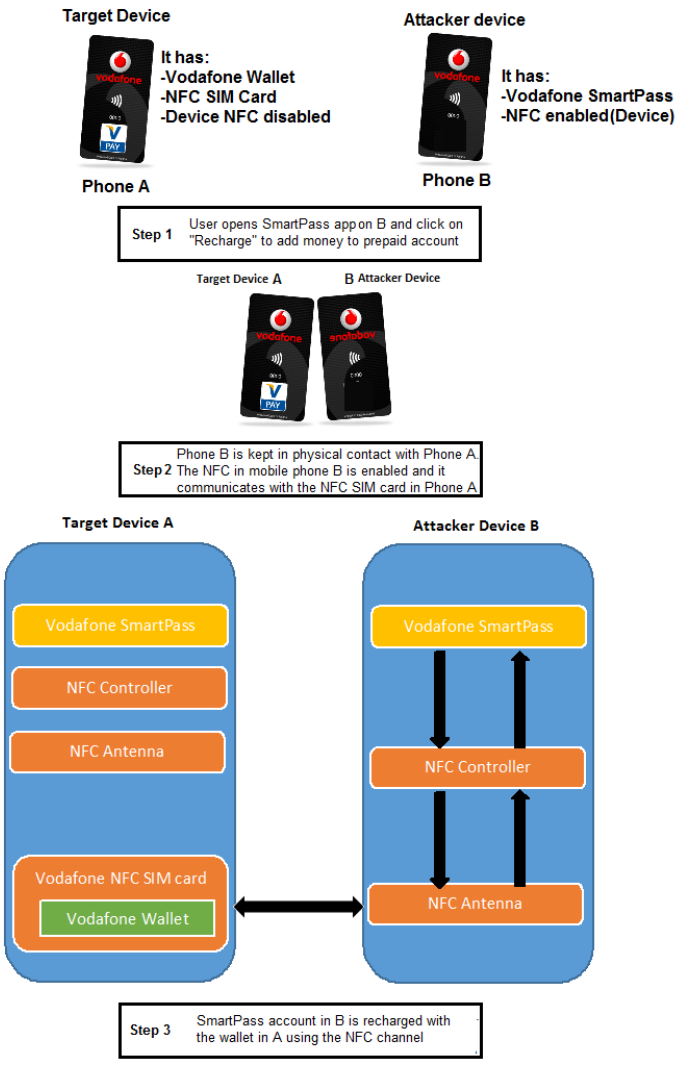


Figure 5.2: Spoofing attack Scenario for NFC SIM card

- There is no PIN verification required while making a transaction below 25 euros. This also includes transactions made while adding credit to the SmartPass prepaid account from the registered VISA card stored in the wallet.
- The attacker's phone B is kept physically close to the target phone A containing the NFC SIM card.

### 5.2.3 Goal of this attack

The main goal of this attack is to allow the attacker to add money to his SmartPass prepaid account using the VISA card information stored in the target mobile phone's NFC SIM card. This attack will verify if there is any access restriction on the data stored on the NFC SIM card when accessed through SmartPass. As SmartPass, Vodafone wallet and the NFC SIM card are provided by Vodafone, we assume that SmartPass is a privilege application which can access the information stored in the wallet in the NFC SIM card.

### 5.2.4 Limitation

There are some limitations of this attack such as that the maximum amount that can be stolen at a time is 25 euros and the details of the prepaid account which the attacker uses will be revealed. The first limitation can be overcome by gaining access to the PIN which will allow transactions greater than 25 euros. The second limitation can be overcome by relaying the information directly to a payment terminal instead of adding value to the prepaid account. In such a scenario, the methodology used for the relay attack performed on NFC-enabled payment cards [31] can be considered similar to the one in the NFC SIM card.

## 5.3 Relay Attack

In a classic relay attack, communication with two parties is initiated by the attacker who then merely relays messages between the two parties without manipulating them or even necessarily reading them[1]. As mentioned in Section 5.2.4, the limitation of the attack scenario discussed can be overcome by relaying the messages directly to the POS terminal.

### 5.3.1 Attack Scenario

In this approach, the *Mole* can be a NFC mobile device or an Arduino-based NFC reader like a customized wireless charger (as discussed in Section 4.3.1.1).

[1] `https://eprint.iacr.org/2011/618.pdf`


Figure 5.3: Wireless charger as a mole

The overall setup is shown in the Figure 5.3.

The following steps are involved in this scenario:

- The attacker keeps the *Proxy* connected to the customized wireless charger using WiFi or Bluetooth.
- The user keeps the *target device* on the customised wireless charger which act as a *Mole*.
- The *Mole* sends the *SELECT AID* APDU to the *target device*.
- The wallet stored in the NFC SIM card, corresponding to the AID, will handle the request and send the response APDU back to the *Mole* which sends it back to the *Proxy*.
- The attacker place the *Proxy* close to the *POS terminal* which processes the response APDU.
- The card information stored in the wallet of the *target device* is used to make the payment at the *POS* terminal.

### 5.3.2 Assumptions

To exploit the vulnerability discussed above, some of the assumptions are as follows:

- The *Target device* with the NFC SIM card has turned its NFC off.

- The NFC SIM card has a registered VISA debit card whose information (alias) is stored in the NFC SIM card.
- There is no PIN verification required for transactions below 25 euros.
- The attacker's phone, i.e. *Mole*, is kept physically close to the *Target device* containing the NFC SIM Card.
- The *Proxy* mobile device (which is under the attacker's control) is kept close to the *POS* terminal.

### 5.3.3 Goal of this attack

The goal of this attack is to steal money from the Vodafone wallet stored in the target mobile device to make a payment at the POS terminal. This will help in hiding the identity of the attacker. Vodafone also claims that payment can be made even if the battery of the mobile device is dead. This confirms that the NFC chip in the SIM card can be directly read by the NFC reader. This makes the attack scenario discussed in Section 5.3.1 more feasible.

### 5.3.4 Limitation

The limitation of this attack scenario is that only transactions up to 25 Euros can be executed at a time. In addition to this, the *Mole* has to be in close proximity to the *target device*.

# Chapter 6

# Countermeasures for Relay attack

This chapter discusses the new countermeasures for relay attacks which we propose as a part of this thesis. The countermeasures utilizes the challenge response protocol. This chapter discusses a sample challenge response protocol, how it provides protection against replay attacks and how it can also be used to provide protection against relay attacks. This chapter provides an answer for the last research question mentioned in Section 1.4. Figure 6.1 shows an overview of this chapter.

## 6.1 Challenge Response Protocol

In a challenge response protocol there are two parties involved who want to communicate with each other. One party is responsible for generating a challenge and sending it to the other party. The other party uses the challenge and generates the response. When the first party generates the challenge, it also knows what response is expected from the other party. If both of them match, the first party can be assured that he is communicating with the correct party. In other words, it helps in authenticating one party to another.

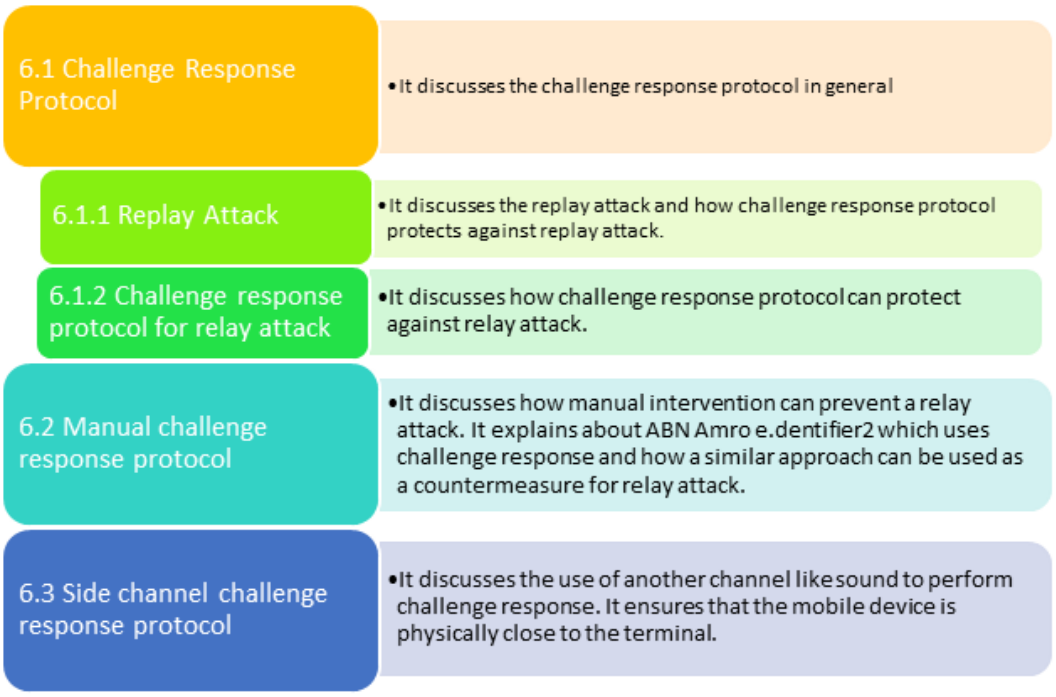


Figure 6.1: An overview of Chapter 6

### 6.1.1 Replay Attack

A replay attack[1] is a form of network attack in which the communication between two parties is captured and then replayed. This is carried out either by the originator or by an adversary who intercepts the data and re-transmits it, possibly as part of a masquerade attack[2]. When the captured data is replayed, the second party gets spoofed that it is still communicating with the original party. But, the attacker spoofs the identity of the original party.

**Challenge response protocol for protection against replay attack** In the replay attack, the requirement is that the data that is captured and replayed is static and there is no randomness in it. In the challenge response protocol, if the generated challenge and expected response are always same and static, the replay attack is possible. However, if the generated challenge and response are

[1] `http://all.net/CID/Attack/papers/Replay.html`
[2] `http://www.veracode.com/security/spoofing-attack`

unique for every transaction, a replay attack will no longer work.

### 6.1.2 Challenge response protocol for protection against relay attack

In a relay attack, the data relayed between the target device and POS terminal does not involve any manual intervention. Because of this the data can be acquired without the user's knowledge. If the manual intervention can be enforced by any means, it will no longer be possible to relay data without the user's knowledge. This manual intervention should not impact the usability of the whole mobile payment system. Both manual intervention and good usability can be enforced by challenge response protocol.

## 6.2 Manual challenge response protocol

Challenge response protocol is usually considered as one of the countermeasures for a replay attack. We propose a challenge response protocol which can provide protection against relay attack as well. This proposal is inspired from the e.dentifier2 provided by ABN Amro bank[1]. *e.dentifier2* is a device provided by ABN Amro bank which is responsible for generating a unique response for the challenge generated by the ABN Amro bank backend system. e.dentifier2 also enables two-factor authentication: the first is the verification of the PIN code and the second is the response generated for the entered challenge. When a user needs to make an online transaction, the e.dentifier2 and the debit card are needed. The web page displays a challenge which is usually a 7 or 8 digit random number. The user needs to insert his debit card into the e.dentifier2 and enter the PIN code. Once the PIN code is verified, the challenge shown on the web page needs to be manually entered by the user in the e.dentifier2, which generates a response code. This code is submitted on the web page and the transaction is complete. A similar approach can be used in case of NFC-based mobile payment system.

[1] `https://www.abnamro.nl/nl/prive/betalen/edentifier/kenmerken.html`

In a normal scenario, when the user taps the mobile phone on the POS terminal, the terminal will generate a challenge and send it to the mobile phone. The mobile phone acquires the received challenge and generates the response for the same. This response is sent back to the POS terminal. In this scenario, the cryptographic protections do not provide any protection against relay attack. In our proposal, instead of the POS terminal automatically sending the challenge to the mobile phone, it will display this challenge on the POS terminal screen (similar to the challenge displayed on the web page discussed above). The user is forced to enter the challenge manually into the mobile phone and generate the response. This response can then be transferred using NFC to the POS terminal or can be manually entered. This forced manual entry of challenge and generating the response will make sure that the actual user who owns the device is physically present while making the payment and is involved in the complete transaction process. Figure 6.2 shows the overview of the steps involved in this protocol.

The advantage of this approach is that relaying of the challenge and response is no longer possible due to manual intervention. Without completing the challenge response protocol, the transaction cannot be initiated. This approach will not require any change in the current infrastructure as current POS terminals have display screens as well as keypads for entering the codes.

This approach will alter the usability of the NFC-based mobile payment system. It is more irritating for a user to have to enter a PIN code for every transaction as he needs to remember the PIN code. If the user needs to enter PIN code for every transaction, the chances of gaining information related to the PIN code are higher as the malicious attacker has more opportunities to observe the user entering the PIN code by using techniques like shoulder surfing. In the proposed approach, the user does not need to remember any code, he just needs to enter the challenge displayed on the screen of the POS terminal. This will not impact significntly on the usability of the solution.

Another option to involve manual intervention is to mandate that the user provides confirmation before finishing the transaction. The mobile phone can

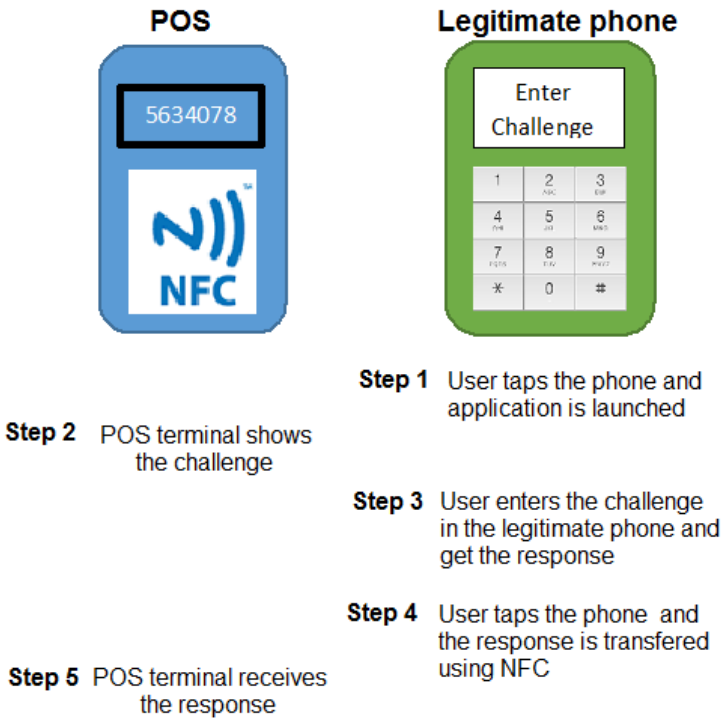


Figure 6.2: Manual Challenge Response protocol

display the itinerary containing the details of the items to purchase and user can manually confirm the same. However, the challenge in this approach is that when the user confirms the itinerary on the phone, it will send the a specific command to the POS terminal so that it knows that user has manually provided the confirmation. This command can be captured and replayed to spoof the POS terminal. This challenge is not applicable in the case of manual challenge response protocol as the response is random and unique for every transaction which makes replaying inefficient.

## 6.3 Side channel challenge response protocol

As discussed in the previous section, the challenge-response is performed between the POS terminal and mobile phone using the NFC channel. In mobile phones, there are additional channels which can be utilized for transferring the informa-

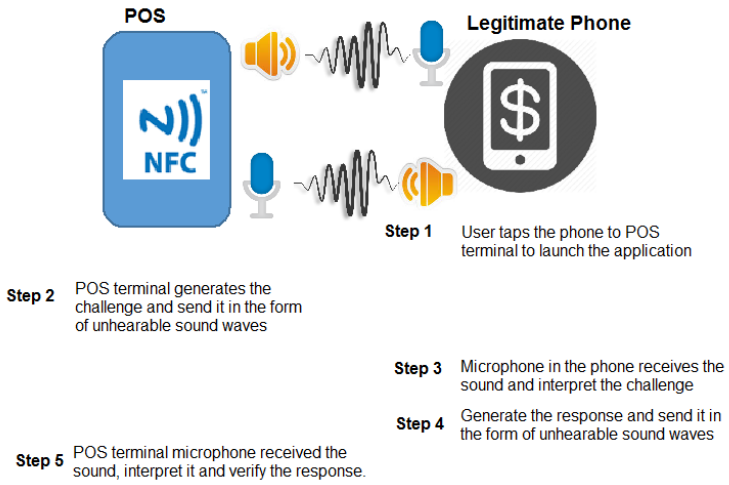


Figure 6.3: Side Channel Challenge Response protocol

tion. One of the channels is sound. One unique authentication mechanism was developed by SlickLogin[1]. It provides sound-based authentication for a web-based login system. When a user needs to login to a web service on a computing device like a laptop, the SlickLogin generates a unique code which is transferred from the laptop speakers to the mobile device in the form of sound waves. These sound waves cannot be heard by human ears. These sound waves are then interpreted by the application on the phone and sent to the authentication server to complete the verification process. This approach makes sure that the device registered by the user is physically present while logging into a web service.

A similar approach can be implemented for mobile payments to provide protection against relay attack. When the mobile phone is tapped on the POS terminal, the challenge is generated and transferred to the phone in the form of sound waves. The application on the phone will receive and interpret it. It will generate the response and will transfer it back to the POS terminal in the form of sound waves. The relaying of sound waves might also be a possibility but it will not be efficient as relaying the sound waves will introduce random noise which will allow the application on the phone to discard the challenge. The sound

[1] `http://www.slicklogin.com/index.html`

waves along with the noise will corrupt the challenge. Therefore, it will not be interpreted by the application. In addition to this, the sound waves used can be restricted to short distant sound waves. Sound waves can also be copied and relayed which will introduce additional time delay while making a transaction. This additional delay can be identified and alerted to discard the transaction. Figure 6.3 shows the overview of the steps involved in this protocol.

The use of additional channels such as sound will ensure that the authentic mobile phone is actually close to the POS terminal. The main challenge of this approach would be to have an additional device which can generate and receive sound waves in the POS terminal. However, it is practically feasible to implement this approach. It would not only prevent a relay attack but would also provide additional authentication. In terms of usability, this approach does not need any user input, which will enhance the overall user experience.

# Chapter 7

# Proof of Concept

This chapter discusses security mechanism implemented in HCE to prevent from device skimming and relay attack. It provides an explanation about the proof of concept showing the flaw in the design of the implementation of security mechanism. It discusses the different experiments performed to verify if the disabling of the NFC controller and the application processor together with the screen is a sufficient countermeasure against attackers that want to perform malicious transactions. Moreover, it discusses the experiments performed on Android 4.4.2 and 5.1.1. Later, a wireless charger is used as a mole to skim the target mobile device, which can be extended to perform a relay attack. Figure 7.1 shows an overview of this chapter.

## 7.1 Relay Attack on HCE

With the introduction of HCE in Android 4.4, it became possible to emulate cards without any need of physical SE. HCE is implemented as a service in Android so that it always runs in the background. This allows an attacker to skim the device and interact with the application implementing the HCE service if NFC is enabled on the mobile device. This skimming of the device might allow attacker to perform a relay attack. To prevent this, an additional security mechanism was implemented in Android 4.4 and later, disabling the NFC controller as well as the

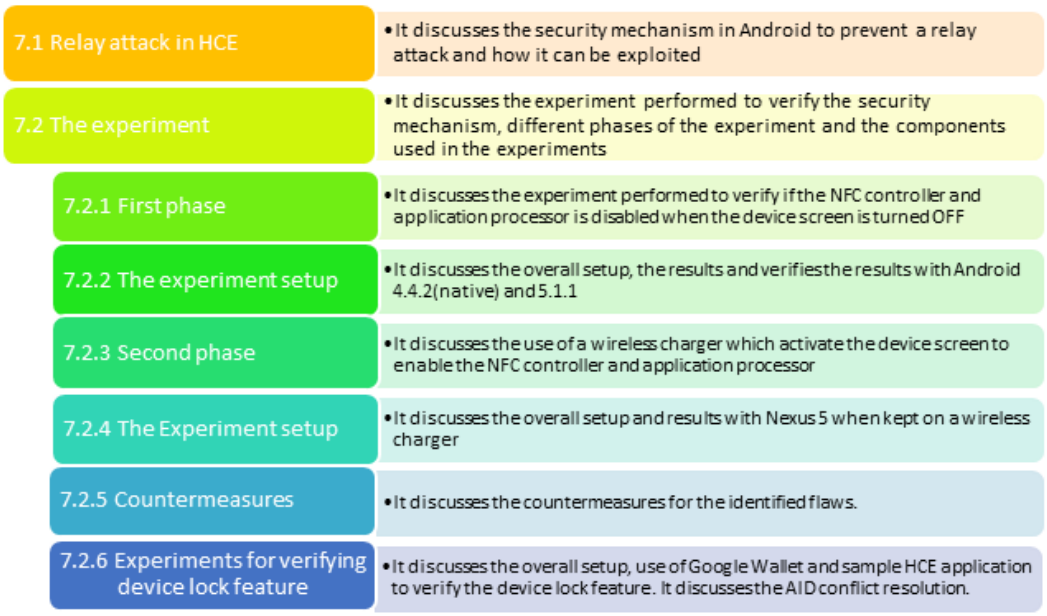


Figure 7.1: An overview of Chapter 7

application processor when the mobile device screen is turned off[1]. As a result, the HCE service will not work. This is explained in detail in Section 4.2.

As discussed earlier in Section 4.3.1.1, it is still possible to perform hardware-based relay attack. The attack scenario discussed in the Section 4.3.1.2 will allow an attacker to enable the device screen for a small time frame. This time frame might be sufficient to perform a relay attack.

## 7.2 The Experiment

The goal of the proposed experiment is to bypass the security mechanism, i.e. disabling the NFC controller as well as the application processor when the mobile device screen is turned off, to perform a relay attack by forcefully enabling the device screen using a wireless charger. To perform the relay attack, the whole experiment is divided into two phases.

[1] `https://developer.android.com/guide/topics/connectivity/nfc/hce.html#ScreenOffBehavior`

The first phase verifies if the NFC controller and application processor are disabled when the device screen is turned off. If the result is positive, the next step is to verify when the device screen is enabled by the wireless charger, if the NFC controller and application processor get enabled or not. If the verification is negative, the next step is intead to verify if the NFC reader can communicate with the HCE service.

The second phase is to verify if the Android operating system performs any checks before enabling the NFC controller and application processor, when the device screen is turned on. This phase also verifies if Android operating system performs any checks i.e. whether the device screen is enabled by pressing any button on the mobile device or not.

### 7.2.1 First Phase

In the first phase, it will be verified if the NFC controller and application processor are disabled when the device screen is turned off. The experiment setup includes following equipment:

- Arduino Uno Board
- Seeed Studio NFC Shield v2.0
- Samsung Galaxy S4 (Samsung custom Android 4.4.2 build)
- Arduino IDE
- A Laptop
- HCE Sample Android Application
- Seeed Studio PN532 library implementing the HCE

#### 7.2.1.1 Arduino Uno Board

Arduino Uno[1] is a microcontroller board based on the ATmega328. It has 14 digital input/output pins, 6 analog inputs, a 16 MHz ceramic resonator, a Universal Serial Bus (USB) connection, a power jack, an In-circuit Serial Programming (ISCP) header, and a reset button. It can be powered on using the USB connector or using an Alternating Current (AC)-to-Direct Current (DC) power adapter or battery. It allows to mount multiple shields like a NFC shield (which provides NFC capabilities to act as a reader or a tag) or a WiFi shield (which allows Arduino board to connect to the Internet).

#### 7.2.1.2 Seeed Studio NFC Shield v2.0

The NFC Shield v2.0 provided by Seeed Studio[2] is compatible with Arduino Uno and can be mounted on it. It has an NFC controller, PN532, handling wireless communication at 13.56MHz to read and write a tag or implement point to point data exchange between the shield and a smartphone. It also contains an NFC antenna which allows the device to act as an NFC reader.

#### 7.2.1.3 Samsung Galaxy S4

Galaxy S4 is one of the smartphones provided by Samsung with NFC capabilities. The Samsung Galaxy S4 comes with Android 4.2.2 (Jelly Bean) during the launch of the mobile device[3]. The version used in the experiment is Android 4.4.2 as it supports HCE (Build number: KOT49H.I9505XXUGNG8). Samsung Galaxy S4 also supports wireless charging[4]. Figure 7.2 shows the Samsung Galaxy S4 used.

#### 7.2.1.4 Arduino IDE

Arduino IDE is an open source development environment provided by Arduino. The provided IDE makes it easy to write code and upload it to the board. It

[1] http://www.arduino.cc/en/Main/ArduinoBoardUno

[2] http://www.seeedstudio.com/wiki/NFC_Shield_V2.0

[3] http://www.samsung.com/global/microsite/galaxys4/

[4] http://www.samsung.com/uk/consumer/mobile-devices/smartphones/smartphone-accessories/EP-WI950EWEGWW

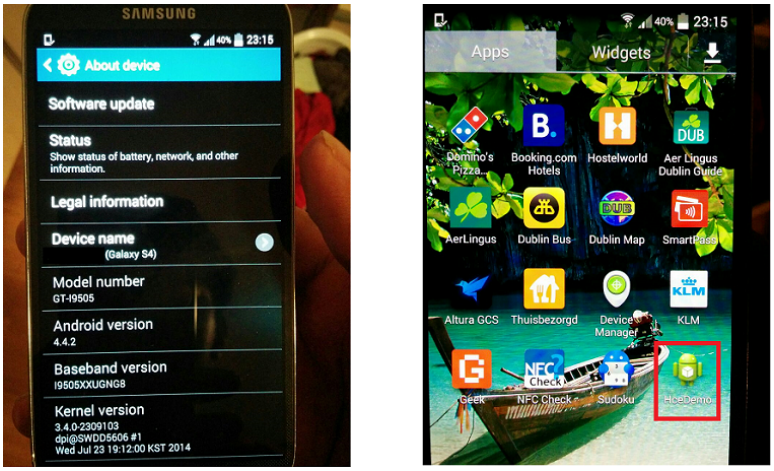


Figure 7.2: Samsung Galaxy S4 with Android 4.4.2

runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software[1]. The version used is 1.6.3.

#### 7.2.1.5 A Laptop

A laptop is used to run the Arduino IDE and upload the code to the Arduino Uno. The one used in the experiment is laptop running Microsoft Windows 8.1.

#### 7.2.1.6 HCE Sample Android Application

An Android application is needed to implement the HCE service. The sample Android application used in the experiment is developed by Adrian Stabiszewski. The reason of choosing this application is that it was already tested with Arduino and PN532 library. It is available on Github[2]. In HCE, the Android application is divided into two categories: Payment and Other[3]. The application used is set to *Other* category. Later, the category is modified to *Payment* for some experiments. It processes APDUs as defined in the ISO/IEC 7816-4. Due to unavailability of any real payment application in the Netherlands, a sample HCE

[1] `http://www.arduino.cc/en/Main/Software`
[2] `https://github.com/grundid/host-card-emulation-sample`
[3] `http://developer.android.com/guide/topics/connectivity/nfc/hce.html`

application is used. It will be interesting to use real payment applications such as Google Wallet but it will not matter much since the flaw is in the design of the implementation.

#### 7.2.1.7 Seeed Studio PN532 library implementing HCE

The NFC library provided by Seeed Studio for PN532 is available on Github[1]. It provides support for different interface like I2C[2], SPI[3] and HSU[4]. This library provides support for communication with Android devices (Android 4.4+). It also provides a working example using this library to communicate with the HCE service and processes APDU based on ISO/IEC 7816-4.

### 7.2.2 The Experiment Setup

The goal of this experiment is to verify if the NFC controller and application processor is disabled when the mobile phone screen is turned off. Figure 7.3 shows the initial set up of the experiment.

To perform the experiment the following steps are executed:

- The NFC shield is mounted on the Arduino Uno. The Seeed Studio PN532 library is installed on the laptop. The Arduino Uno is connected to the laptop. Once connected, the working example from Github is opened in the Arduino IDE. The code is compiled and uploaded on the Arduino Uno. Figure 7.4 shows the compiled code and "Done uploading" message.
- The HCE sample application is installed on the Samsung Galaxy S4. NFC is enabled in the phone settings.
- The serial monitor window is opened in Arduino IDE. Once opened, the NFC antenna on the NFC shield waits for an ISO14443A card. (Android

[1] https://github.com/Seeed-Studio/PN532
[2] http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html
[3] http://www.arduino.cc/en/Reference/SPI
[4] https://github.com/elechouse/PN532

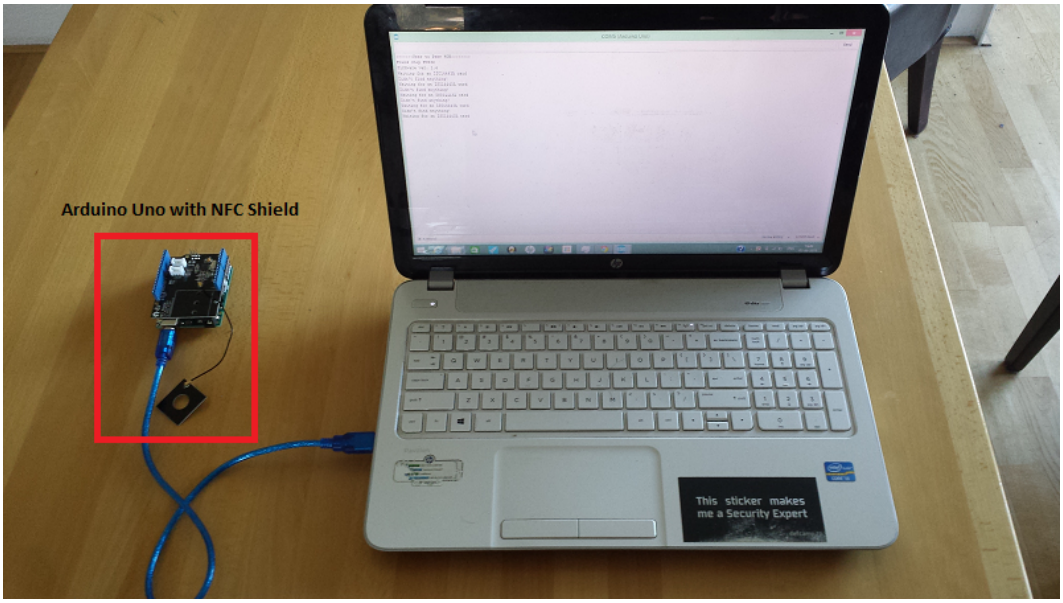


Figure 7.3: Initial Setup

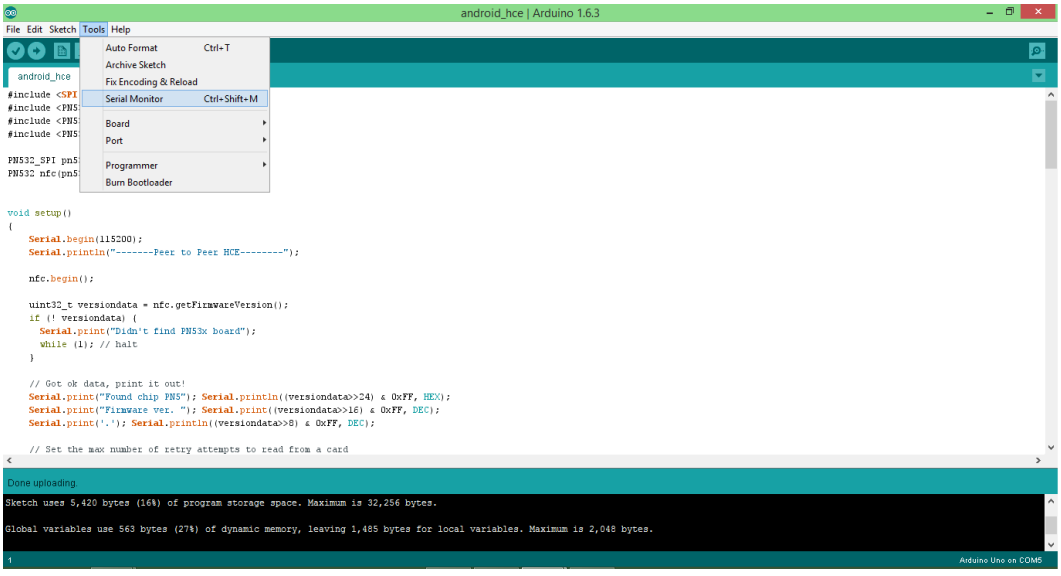


Figure 7.4: Arduino IDE

HCE provides support for ISO14443A and ISO14443B cards[1]). This card is currently emulated in the sample HCE application on the phone.

[1] `https://developer.android.com/guide/topics/connectivity/nfc/hce.html`

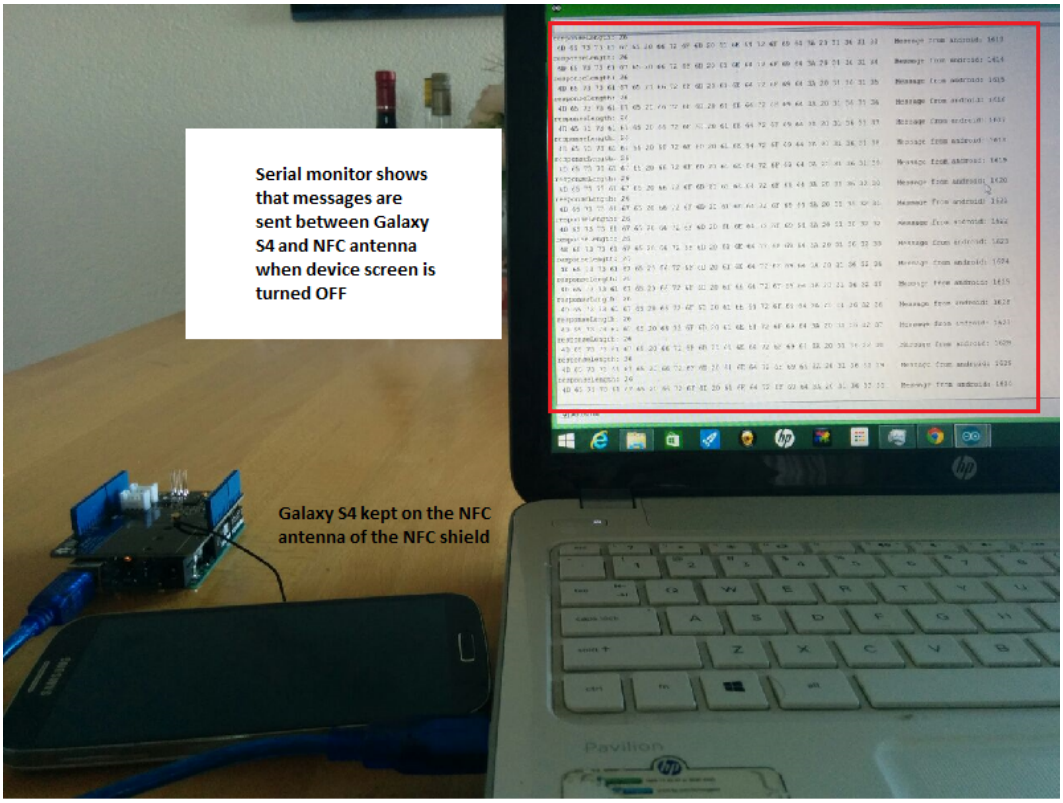


Figure 7.5: The overall setup

- The device screen of Galaxy S4 is disabled. Galaxy S4 is kept on the antenna of the NFC shield so that it is in close proximity. Figure 7.5 shows the overall setup.

#### 7.2.2.1 Result

The result of phase one of the experiment is as follows: In Android 4.4.2 (build number: KOT49H.I9505XXUGNG8), the NFC controller and application processor is not disabled when the mobile phone screen is turned off. The antenna of the NFC shield is able to communicate with the Android application implementing HCE.

Figure 7.6 shows that the NFC shield is waiting for the card. Once the Galaxy S4 is kept close to the NFC antenna of the shield, the serial monitor shows that

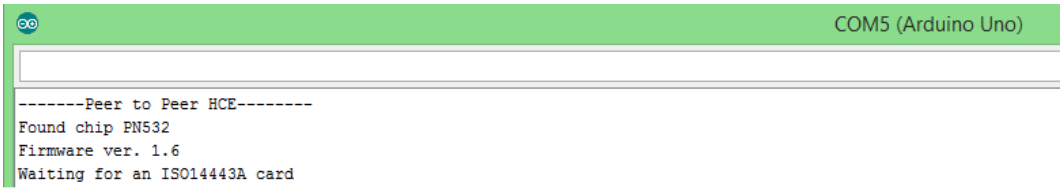


Figure 7.6: Serial monitor output

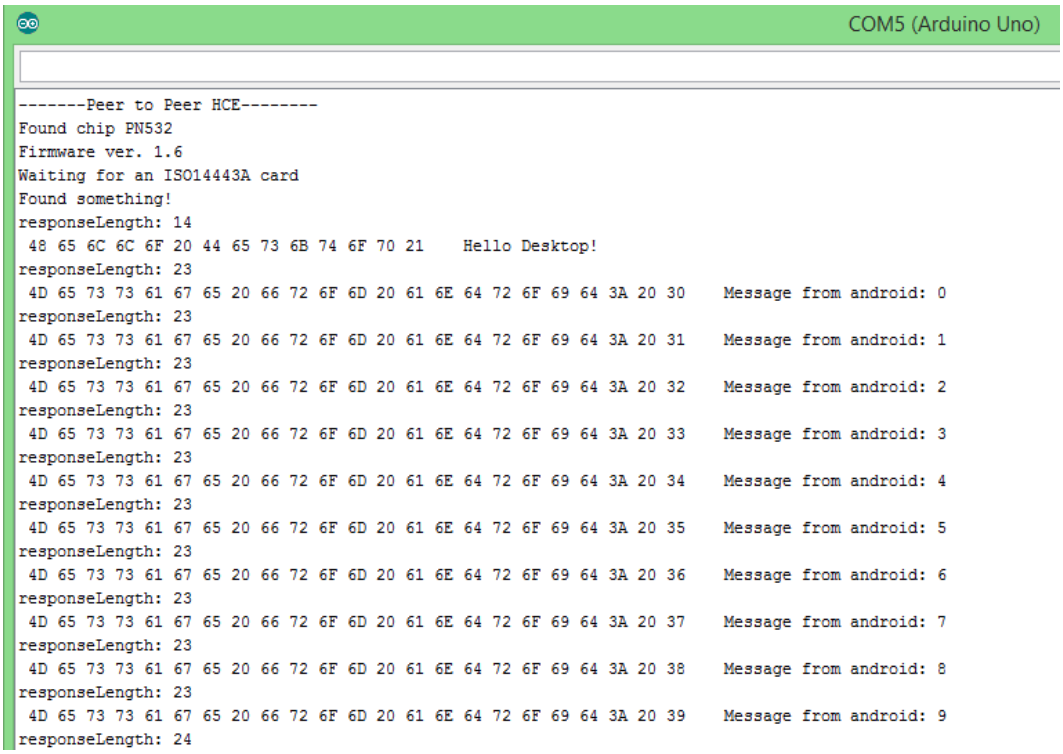


Figure 7.7: Serial monitor output showing communication with android HCE app

it is communicating with the HCE service on the phone and shows a message "Hello Desktop". Figure 7.7 shows the output. In addition to this, Figure 7.8 shows the part of the source code of the android application implementing HCE. It shows that when the application receives the first APDU, it sends message "Hello Desktop". Later, when the application receives further APDUs, it sends messages with a counter.

```java
package de.grundid.hcedemo;

import android.nfc.cardemulation.HostApduService;
import android.os.Bundle;
import android.util.Log;

public class MyHostApduService extends HostApduService {

        private int messageCounter = 0;

        @Override
        public byte[] processCommandApdu(byte[] apdu, Bundle extras) {
                if (selectAidApdu(apdu)) {
                        Log.i("HCEDEMO", "Application selected");
                        return getWelcomeMessage();
                }
                else {
                        Log.i("HCEDEMO", "Received: " + new String(apdu));
                        return getNextMessage();
                }
        }

        private byte[] getWelcomeMessage() {
                return "Hello Desktop!".getBytes();
        }

        private byte[] getNextMessage() {
                return ("Message from android: " + messageCounter++).getBytes();
        }

        private boolean selectAidApdu(byte[] apdu) {
                return apdu.length >= 2 && apdu[0] == (byte)0 && apdu[1] == (byte)0xa4;
        }

        @Override
        public void onDeactivated(int reason) {
                Log.i("HCEDEMO", "Deactivated: " + reason);
        }
}
```

Figure 7.8: Source code of the sample HCE app. Refer Appendix A

#### 7.2.2.2 Experiment with Android 4.4.2 on Nexus 7

The same experiment was performed on native Android 4.4.2 (build number: KOT49H) which is provided by Google. The results show that the NFC controller and application processor are disabled when the mobile phone screen is turned off. This suggests that the flaw discussed above is present only in custom build versions of Android 4.4.2.


Figure 7.9: Experiment with Android 5.1.1

#### 7.2.2.3 Experiment with Android 5.1.1 on Nexus 5

The same experiment was performed on Android 5.1.1 (build number: LMY48B) using Google Nexus 5. The result completely differs from the ones using the custom build of Android 4.4.2. In Android 5.1.1, the NFC controller and application processor are disabled when the mobile phone screen is turned off. When the screen is turned on, the NFC controller and application processor are enabled. Figure 7.9 shows the results. On the left side, Nexus 5 phone is kept on the NFC antenna with the screen disabled and on the right side, the serial monitor shows that no card has been found.

Therefore, the experiment results show that in the native Android builds (the ones we checked), the security mechanism is implemented. The flaw is present in the custom build of Android 4.4.2 by Samsung. This lays the foundation to move to phase 2 of the experiment.

### 7.2.3 Second Phase

This phase will verify when the device screen is enabled, whether the NFC controller and application processor are enabled or not. The result for Android 4.4.2 (build number: KOT49H.I9505XXUGNG8) shows that the NFC controller and application processor are enabled even if the device screen is off. Therefore, there is no need to verify further. In the second phase, Android 5.1.1 will be tested.

There are two modifications in the overall setup when compared to the phase one. A wireless charger is used in addition to the other components and instead of the Samsung Galaxy S4, a Google Nexus 5 with Android 5.1.1 was used.

**Wireless Charger** The wireless charger uses an electromagnetic field to transfer energy between two objects. Energy is sent through an inductive coupling to an electrical device, which can then use that energy to charge batteries or run the device[1]. The reason of using a wireless charger in the attack scenario is to enable the display screen of the device. When a mobile device is kept on the wireless charger, it enables the device screen and shows a notification that the device is charging. In addition to this, when the wireless charger is removed, the device screen is again switched on and shows a notification that the device has stopped charging.

**Google Nexus 5** Nexus 5[2] has NFC as well as in-built support for wireless charging. The Android version used in the experiment is 5.1.1. Figure 7.10 shows the phone used. Figure 7.11 shows the initial experiment setup for second phase.

### 7.2.4 The Experiment Setup

In this setup, the Nexus 5 is kept on the NFC antenna of the shield. On the left side, the device screen is turned *on* and on the right side, the serial output shows

[1] `http://goo.gl/1OnZc1`
[2] `http://www.google.com/intl/ALL/nexus/5/`

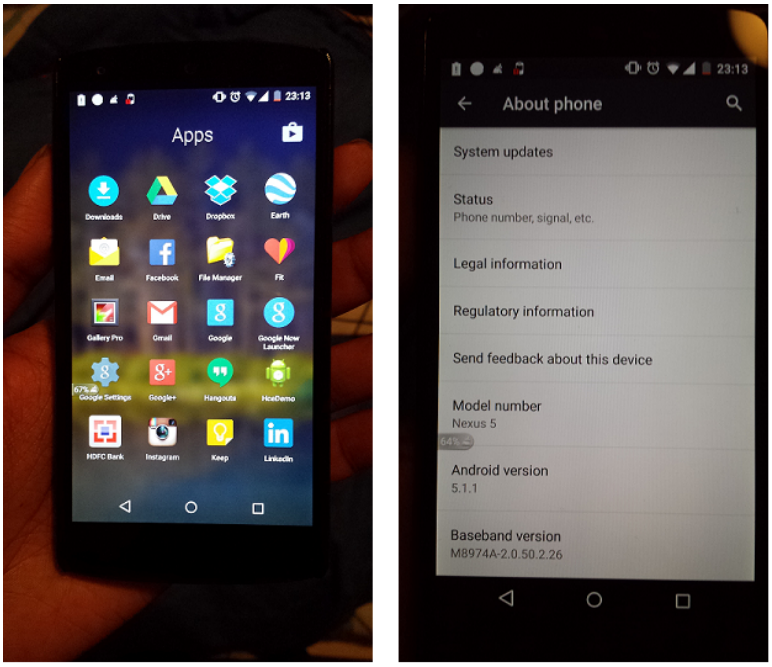


Figure 7.10: Google Nexus 5 with Android 5.1.1

that it is able to receive the messages.

In the next step, the wireless charger is powered *on*, the antenna of the NFC shield is kept on the surface of the wireless charger and the Nexus 5 is kept on the top of the NFC antenna. This way, Nexus 5 has contact with both i.e. wireless charger as well as NFC antenna. Figure 7.12 shows this setup.

#### 7.2.4.1 Result

The result of this experiment is that when the Nexus 5 is kept on the wireless charger, the device screen is turned *on*. The serial output result shows that the message is transferred between the Nexus 5 and the NFC antenna. Therefore, once the device display is active, the NFC controller and application processor get enabled (If the device screen is manually disabled when the communication is already happening, with little delay the communication is dropped as the NFC controller gets disabled). Unfortunately, the communication between the Nexus 5 and NFC antenna is not stable. This might be because of the interference be-

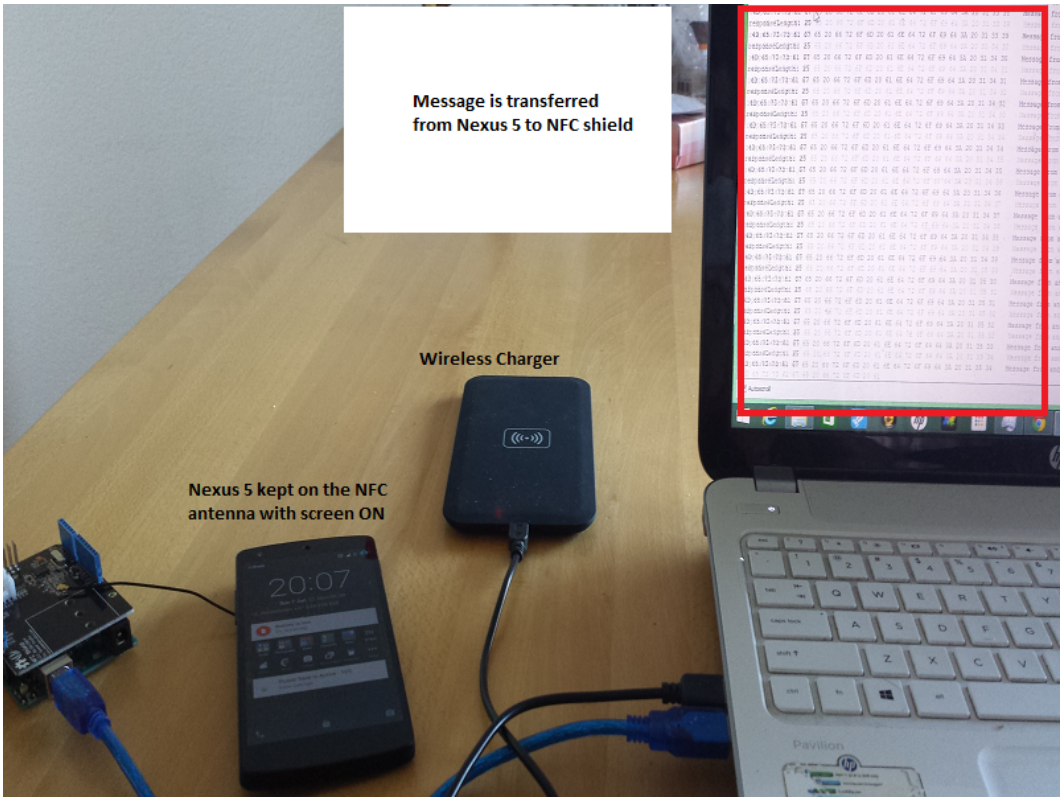


Figure 7.11: Initial setup with Google Nexus 5 and Wireless Charger

tween the electromagnetic induction generated by the wireless charger and the NFC antenna. There are multiple commercial products available which have NFC and wireless charger embedded in one device like Asus NFC EXPRESS 2 and Wireless Charger[1].

This unstable behaviour can be resolved by powering *off* the wireless charger. The device display screen not only gets active when the wireless charger is powered *on*, but also when the wireless charger is powered *off*. In this case, there is no electromagnetic interference caused, which allows the device display screen to be active and the message transfer between the Nexus 5 and NFC antenna starts without any problem. Figure 7.13 shows the results. On the left, a wireless charger is powered *off* which turns the device display screen *on*. On the

[1] `https://www.asus.com/Motherboards/NFC_EXPRESS_2_Wireless_Charger/`

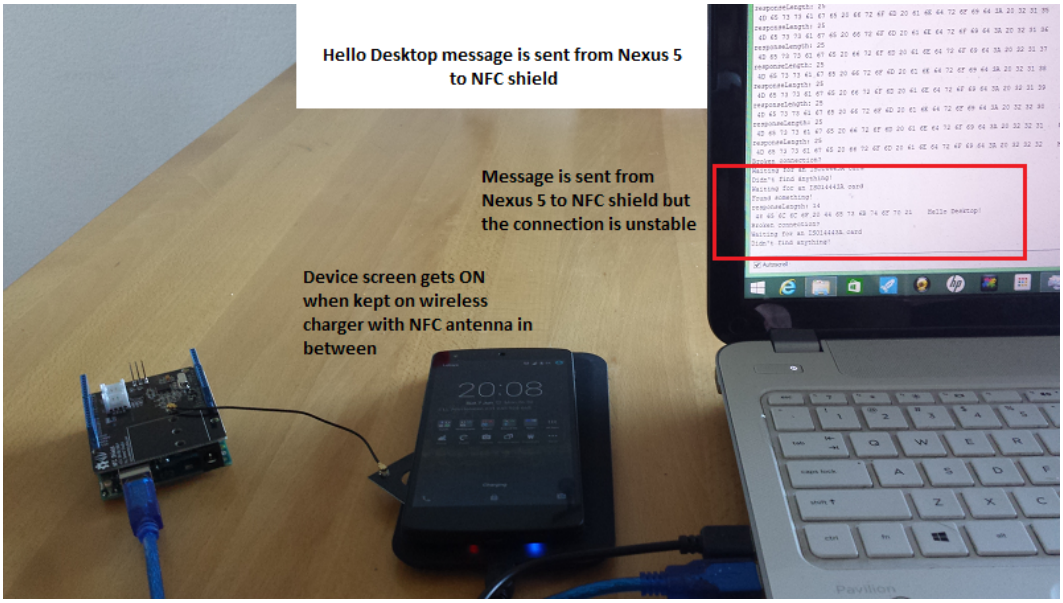


Figure 7.12: Google Nexus 5 is kept on the Wireless Charger along with the NFC antenna

right side, the serial monitor shows that messages are getting transferred without any interference. However, the message transfer is limited by the time frame for which the device screen is active. This time frame is configurable in the Settings user interface. By default, the screen time out is set to 30 seconds in Android[1] 5.1.1, which means device screen is turned *off* automatically when there is no interaction between the device and the user for 30 seconds.

The time frame of 30 seconds is enough for performing an NFC transaction. In case of NFC based payment in *Transport For London*[2], transaction time is close to 500 milliseconds. While in case of Google Wallet and Apple Pay, it is between 1 to 2 seconds[3]. If the delay time caused due to relaying the APDUs between *Mole* and *Proxy* is included, still 30 seconds will be enough to perform the transaction.

[1] `http://www.androidcentral.com/android-101-how-change-screen-time-out-length`
[2] `http://www.eetimes.com/author.asp?doc_id=1322320`
[3] `http://www.engadget.com/2014/10/29/week-apple-pay-google-wallet/`

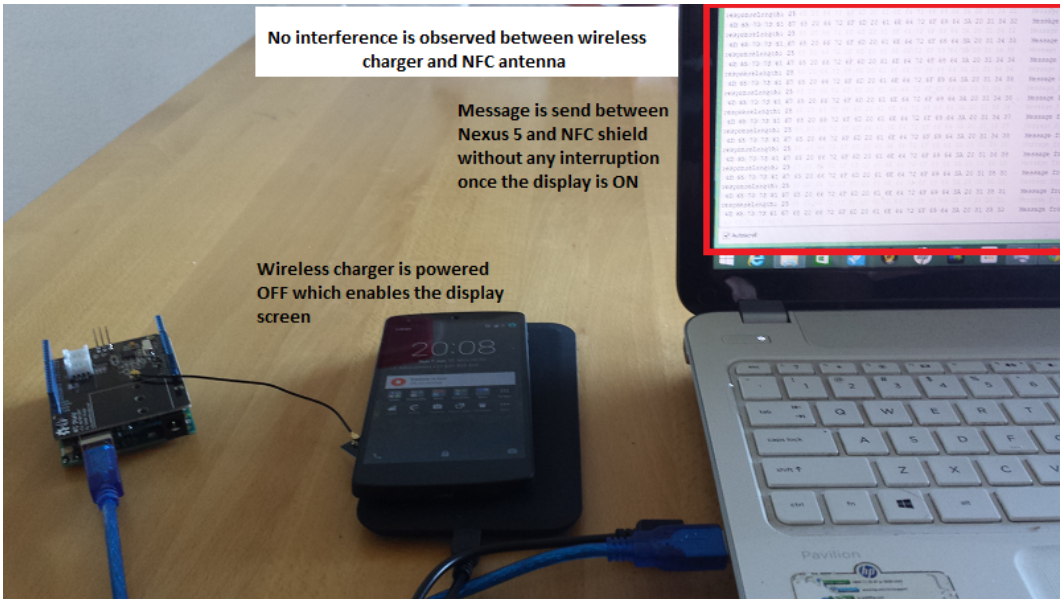


Figure 7.13: Google Nexus 5 kept on the Wireless Charger(powered OFF) along with NFC antenna

A limitation of this approach is that Android provides a functionality which forces the user to unlock the device for initiating a transaction. This functionality can be configured by setting the attribute *android:requireDeviceUnlock* to true in the *apduservice.xml* file. By default, the attribute is set to false. Therefore, it depends upon the implementation of the application. As the survey conducted by Consumer Report in 2014 states 34 percent of the US smartphone users do not enable device screen lock[1]. Apart from this, even if the attribute *android:requireDeviceUnlock* is set to true, there are still chances that users can keep their phones on the wireless charger when the device is unlocked.

### 7.2.5 Countermeasures

The identified flaw in the design of the implementation of the security mechanism can be fixed by performing a check on the event, which is causing the device screen to be enabled. This check needs to be performed before the NFC controller and

[1] `http://goo.gl/Tdm15g`

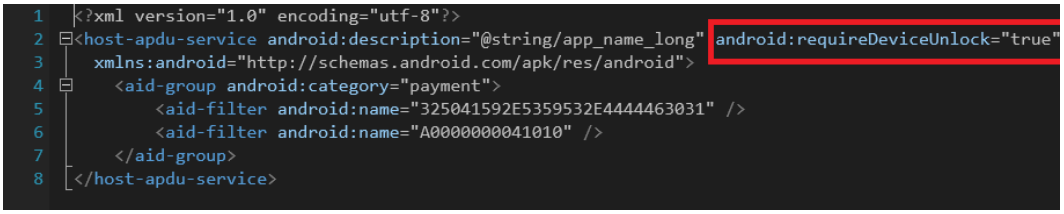


Figure 7.14: hce_aids file showing requireDeviceUnlock attribute is true in Google Wallet

application processor are turned *on*. It needs to ensure that they are turned *on* only when the power button is pressed. In addition to this, the countermeasures discussed in section 6.1.2 will also provide additional security mechanism for protection against relay attacks.

### 7.2.6 Experiments for verifying device lock feature

The goal of this experiment is to verify if the device is locked, it's screen is turned on and *android:requireDeviceUnlock* is set to true in the application, is it still possible for the NFC reader to communicate with the HCE service.

#### 7.2.6.1 The Experiment Setup

The setup used for this experiment is same as the previous one. However, the mobile device used is Nexus 7[1] running native Android 4.4.2 (KOT49H). Apart from the sample HCE application, Google Wallet[2] (Version: 9.0-R206-v12) is used. The reason of choosing Google Wallet for this experiment is that *android:requireDeviceUnlock* is already set to true. This was revealed by decompiling the apk file. Figure 7.14 shows the xml file specifying the same.

#### 7.2.6.2 The Experiment

The Seeed Studio PN532 library implementing the HCE is modified so that it can communicate with Google Wallet. The AID "325041592E5359532E4444463031"

[1] https://www.asus.com/Tablets/Nexus_7/

[2] https://play.google.com/store/apps/details?id=com.google.android.apps.walletnfcrel&hl=en

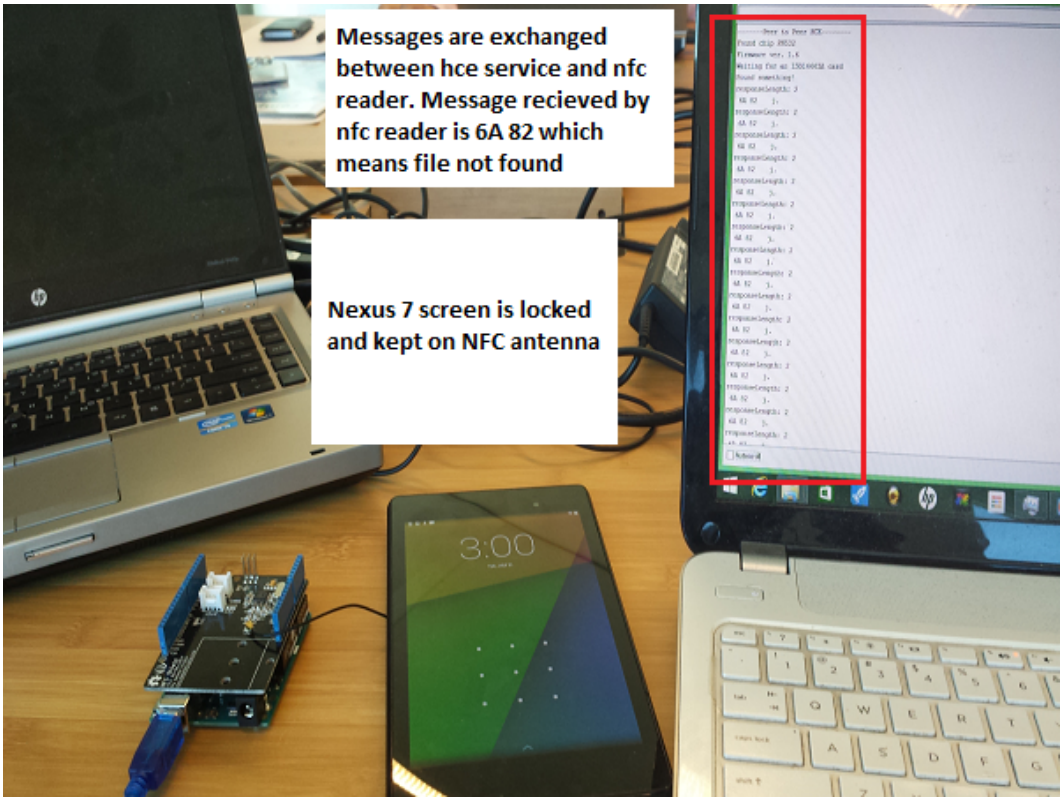


Figure 7.15: Google Wallet communicates with the NFC reader when the device is locked

which corresponds to Google Wallet is configured in the PN532 library. Figure 7.15 shows the overall setup. Google Wallet is installed in Nexus 7 and is running. NFC is enabled which allows Google Wallet to communicate with the NFC reader. The device screen is enabled and is in locked state. However, the output on serial monitor of Arduino IDE shows that HCE service corresponding to Google Wallet AID is found and it responds with APDU "6A 82". This APDU means "FILE NOT FOUND"[1]. The result shows that when the device is locked and *android:requireDeviceUnlock* is set to true some exchanges of APDUs are taking place. The message "FILE NOT FOUND" might be sent because Google Wallet is not configured and it does not have the necessary configuration files.

The same experiment is repeated using the sample HCE application used

[1] `https://www.eftlab.com.au/index.php/site-map/knowledge-base/118-apdu-response-list`

```
C:\thesis\apktool>apktool b host-card-emulation-sample
I: Using Apktool 2.0.0
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...

C:\SignApk>java -jar signapk.jar certificate.pem key.pk8 host-card-emulation-sample.apk host-card-emulation-signed.apk
```

Figure 7.16: Use of apktool and SignApk

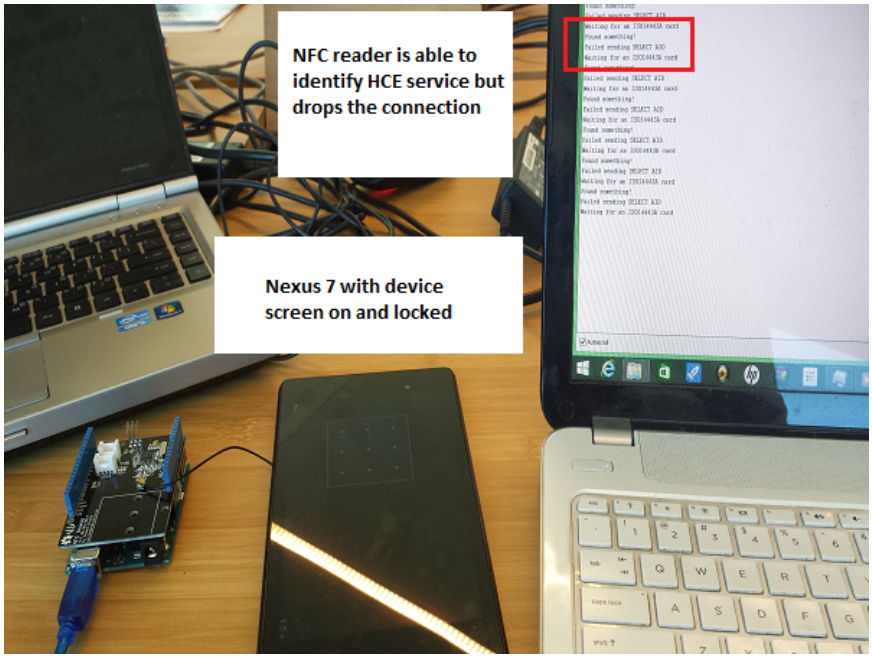


Figure 7.17: Sample HCE application with requireDeviceUnlock set to true and Nexus 7 screen is on and locked

earlier. *android:requireDeviceUnlock* is modified to true and recompiled using apktool[1] and the new apk is signed using SignAPK tool[2]. Figure 7.16 shows the commands executed for recompiling and signing the apk file.

In case of sample HCE application, the result is different. The output on the serial monitor shows that the NFC reader is able to identify the HCE service, but later it drops the connection. Figure 7.17 show the results.

In the last experiment, both sample HCE application and Google Wallet are

[1] `https://ibotpeaches.github.io/Apktool/`
[2] `https://code.google.com/p/signapk/`

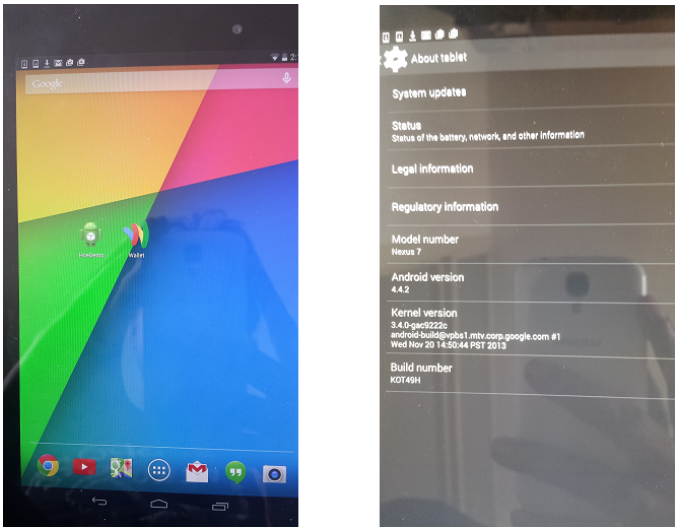


Figure 7.18: requireDeviceUnlock attribute is set to false in the sample HCE application and true in Google Wallet

installed on Nexus 7 (Native Android 4.4.2). Figure 7.18 shows that both applications are installed. The goal of this experiment is to verify if two applications are installed with same AID, one with *android:requireDeviceUnlock* set to true and other to false, does it create any confusion in AID conflict resolution. Attribute *android:requireDeviceUnlock* is set to false in the sample HCE application and true in Google Wallet. Both applications are configured to same AID i.e. "325041592E5359532E4444463031". When the device screen is turned on and kept in locked state, the output on the serial monitor shows that NFC reader is able to communicate to both applications. Figure 7.19 show the results.

#### 7.2.6.3 Results

The results of the experiments show that device lock feature depends on the implementation of the application. In the case of Google Wallet, the APDUs were exchanged and communication was established. However, in sample HCE application, there was no response from the application and the communication was immediately dropped. The last experiment shows that the AID conflict

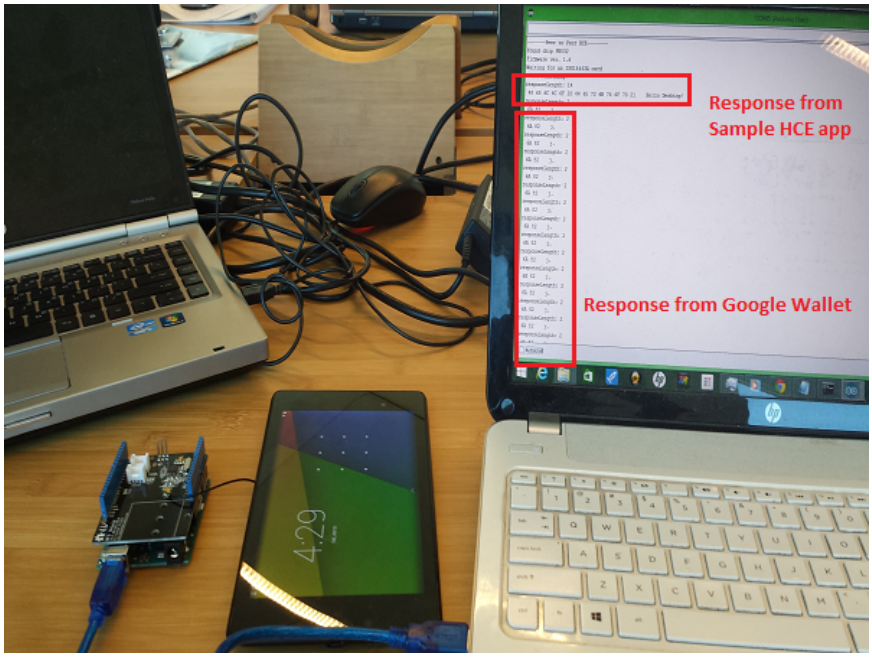


Figure 7.19: Serial monitor shows that NFC reader is able to communicate to both applications

resolution does not work as mentioned in Android documentation[1].

[1] https://developer.android.com/guide/topics/connectivity/nfc/hce.html#AidConflicts

# Chapter 8

# Future Work and Conclusions

This chapter provides a summary of the research results discussed in this thesis. Section 8.1 offers an overview of the thesis. Section 8.2 provides answers to the research questions and presents the conclusions. Section 8.3 provides an overview of future work to extend the results of this thesis.

## 8.1 Overview

Mobile payments have evolved from mobile banking to contactless payment which uses radio communication technology. NFC-enabled mobile devices emulate contactless cards either by using hardware-based SE or software-based, i.e. HCE. Both have their own advantages and disadvantages. We provided a detailed comparison between the different forms of SE, i.e. UICC, Micro-SD or eSE and SE on the cloud. The comparison was done using security, re-usability and standardization as three key attributes. Based on the overall analysis, UICC was considered as a best option among others.

The term HCE was coined in 2012 by SimplyTapp, and this technology was later supported by Android from version 4.4 and above. Android 4.4 also introduced a security mechanism which turns off the NFC controller and application processor when the device display screen is disabled. This protects the mobile device from skimming. We found that this security mechanism was not working

in Android 4.4.2 (Build number: KOT49H.I9505XXUGNG8). We tested with Android 4.4.2 (Build number:KOT49H) and this security mechanism was working properly. We also tested the latest release of Android, 5.1.1 (build number: LMY48B) and confirmed that this security mechanism was working in this build. We presented a flaw in the design of the implementation which is used in this security mechanism in Android 5.1.1. The identified flaw was that Android does not verify the event which enables the display screen before enabling the NFC controller and application processor. Any event such as receiving a notification or text message enables the display screen, which in turn enables the device to be skimmed by a NFC reader if it is kept in close proximity. However, it will be practically difficult to skim a mobile device when its screen is enabled due to a received message or notification as the attacker needs to be physically close to the target device. We presented a proof-of-concept in which a wireless charger, along with a NFC reader, was able to skim the mobile device implementing the HCE service. The proof-of-concept can be further extended to perform a relay attack on the payment solutions implementing the HCE. We also described different attack vectors such as a man-in-the-middle attack and denial-of-service attack which were specifically targeted to the HCE service implemented in Android.

Vodafone launched NFC SIM cards in the Netherlands in April 2015. We provided an analysis of the different components involved in the Vodafone NFC SIM card-based payment solution i.e. Vodafone SmartPass, Vodafone wallet and Vodafone NFC SIM card. We suggested different attack vectors for this payment solution which included spoofing and relay attack.

Different countermeasures like distance or time-bound protocols have been implemented in mobile payment solutions to provide protection against relay attacks. However, they have limitations which affect their efficiency. We proposed two countermeasures for relay attacks which were based on a challenge response protocol. The first countermeasure proposed was the manual challenge response protocol which introduces manual intervention and forces users to manually enter the response for the challenge in the POS terminal. Another countermeasure proposed uses channels like sound waves to perform the challenge response which

forces the actual device to be physically close to the POS terminal while making a payment.

## 8.2 Conclusion

This section provides the details of the findings specific to each research question.

**First Research Question** In terms of security, re-usability and standardization, which form of SE i.e. UICC, Micro-SD, eSE and cloud-based SE is better?

In terms of security, re-usability and standardization, UICC is the better option. It provides secure storage space for deploying payment applications and their corresponding data. The access control is restricted by the MNOs using cryptographic keys and creating different security domains. It uses TSM and OTA mechanisms to manage applications remotely. It can also be re-used if the user changes his mobile device. In terms of availability of standards, UICC has been avaolable for some time and has evolved, as have the standards. Other forms of SE lacks in one of the three attributes. For example, eSE lacks in re-usability as it is embedded inside the mobile device and cannot be removed by the user. Cloud-based SE security relies on the controls implemented by the cloud service provider. Different cloud service providers might implement different controls which might correspond to different security levels. Micro-SD-based SE lacks in standards as well as access control mechanisms.

**Second Research Question** What are the different attack vectors and security controls in mobile payment systems based on HCE?

Based on our analysis of the available documentation of Android HCE and reverse engineering the apk of some of the HCE-based mobile payment applications, we proposed different attack vectors. A hardware-based relay attack which uses the wireless charger as a mole allows the device screen to be enabled, which in turn enables the NFC to perform the relay attack. Another attack vector is a software-based relay attack which removes the dependency of a dedicated piece

of hardware to skim the device. A malicious application with required system privileges can intercept the traffic between the HCE service and the operating system. Man-in-the-middle is another attack vector which relies on the flaw in the implementation of SSL/TLS to intercept the traffic between the mobile device and cloud servers. The final attack vector we proposed is the denial-of-service attack in which we found multiple flaws such AID conflict, single point of failure of routing table, etc. Android HCE has implemented two security mechanisms, *BIND_NFC_SERVICE* which allows only the operating system to communicate with HCE service and the screen off as well as device lock feature which protects the device from skimming by disabling the NFC controller when the device screen is off or locked. Using a hardware-based relay attack, we suggested a way to bypass the second security feature and find a flaw in the design of the implementation. With the help of proof of concept, we proved that it is practically possible to perform device skimming with the use of a wireless charger.

**Third Research Question** What are the different attack vectors and security controls in an NFC SIM card-based mobile based payment system?

Based on the analysis of available documentation for the Vodafone NFC SIM card payment solution, we proposed two attack vectors. One is a spoofing attack in which an attacker can trick the NFC SIM card in the target device and can add money to his SmartPass prepaid account using the wallet stored in the target NFC SIM card. However, the limitation of this approach is that the identity of the attacker can be revealed by verifying the SmartPass transaction history details. To overcome the limitation, we proposed the hardware-based relay attack. Using the wireless charger as a mole, the relay attack can be performed on the Vodafone NFC SIM card-based payment solution. The security control implemented is the use of PIN codes if the transaction amount is more than 25 euros.

**Fourth Research Question** Are existing security controls against relay attacks adequate or, if not, can we propose enhanced ones?

We proposed two countermeasures which can make it more difficult for the

attacker to perform a relay attack. Both are based on the challenge response protocol. The first is a manual challenge response protocol which asks the user to enter the challenge manually in the payment application. This ensures that there is manual intervention involved so that the user is always aware when a transaction is made. This approach might impact the usability aspect of the mobile NFC-based payment system. To remove this limitation, the second approach uses an additional channel like sound waves to perform the challenge response. This additional channel will make it more difficult for the attacker to relay the sound waves along with messages on the NFC channel. It might be possible for an attacker to capture the sound waves and then relay them, but it would introduce an additional time delay which could be identified and the transaction could be terminated. The device lock feature in Android also prevents a relay attack but it depends whether it is enabled in the application and is implemented correctly.

## 8.3 Future Work

This section provides details about possible future work.

### 8.3.1 Test the new attack vectors proposed

As a part of this thesis, multiple new attack vectors have been proposed for HCE-based mobile payment solutions. A software-based relay attack can be tested by using a malicious application which has the required system privileges to intercept the traffic between the operating system and HCE service. The man-in-the-middle attack discussed in this thesis relies on the implementation of SSL/TLS. It will be interesting to verify the solutions which use HCE and cloud-based SE for this attack. Denial-of-service is another attack vector which is proposed. It would be interesting to verify and extend many of these propsed approaches.

### 8.3.2 Attacks on NFC SIM card

With the NFC SIM card, it would be interesting to see if there is any dependency on the application which can restrict relay attacks as discussed in this thesis. In

the case of the Vodafone NFC SIM card, the transaction can be conducted even if the battery of the mobile device is dead. This suggests that the NFC SIM card can directly interact with the POS terminal. This provides an opportunity for device skimming as well as performing a relay attack.

### 8.3.3 Other possibilities of hardware-based relay attacks

There are possibilities of using other hardware devices similar to the wireless charger to extend the proof of concept discussed in this thesis. Devices like Music dock stations or NFC EXPRESS 2 & Wireless Charger from ASUS[1] are some of the other options. It will also be interesting to extend the relay attack to wearable devices such as the Apple Watch which supports NFC-based payments. The limited hardware capability of these wearable devices when compared to mobile devices offer the opportunity to uncover weak security implementations. For instance, the Apple watch relies on the heart-beat sensor, fingerprint scanner and PIN codes while performing an NFC-based transaction. Therefore, it would be interesting to see if it is possible to skim the watch to perform a relay attack.

### 8.3.4 Countermeasures to relay attacks

It will also be interesting to implement the challenge response protocols proposed in this thesis as countermeasures to the relay attack and to verify their effectiveness. There are opportunities to identify other channels than sound to perform a challenge response, such as Bluetooth, light waves or radio waves.

In the end, this thesis attempted to contribute towards making NFC-based payment systems more secure by identifying different attack vectors. The comparison between different forms of SE will help companies to choose the right solution for their payment solution. The proof-of-concept emphasizes that it is not only important to implement different innovative security mechanisms but also to consider the design of the security mechanisms.

[1] `https://www.asus.com/Motherboard-Accessories/NFC_EXPRESS_2_Wireless_Charger/`

# Appendix A

This chapter provides the source code of the HCE sample application used in this thesis. The complete source code is available at GitHub[1].

*IsoDepAdapter.java*

```
package de.grundid.hcedemo;

import java.util.ArrayList;
import java.util.List;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;

public class IsoDepAdapter extends BaseAdapter {

        private LayoutInflater layoutInflater;
        private List<String> messages = new ArrayList<String>(100);
        private int messageCounter;

        public IsoDepAdapter(LayoutInflater layoutInflater) {
                this.layoutInflater = layoutInflater;
        }

        public void addMessage(String message) {
                messageCounter++;
                messages.add("Message [" + messageCounter + "]: " + message);
                notifyDataSetChanged();
        }

        @Override
        public int getCount() {
                return messages == null ? 0 : messages.size();
        }

        @Override
        public Object getItem(int position) {
                return messages.get(position);
        }
```

[1] https://github.com/grundid/host-card-emulation-sample/blob/master/src/de/grundid/hcedemo/

```
        @Override
        public long getItemId(int position) {
                return 0;
        }

        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
                if (convertView == null) {
                        convertView = layoutInflater.inflate(android.R.layout.simple_list_item_1,
parent, false);
                }
                TextView view = (TextView)convertView.findViewById(android.R.id.text1);
                view.setText((CharSequence)getItem(position));
                return convertView;
        }
}
```

*IsoDepTransceiver.java*

```
package de.grundid.hcedemo;

import java.io.IOException;

import android.nfc.tech.IsoDep;
import android.util.Log;

public class IsoDepTransceiver implements Runnable {


        public interface OnMessageReceived {

                void onMessage(byte[] message);

                void onError(Exception exception);
        }

        private IsoDep isoDep;
        private OnMessageReceived onMessageReceived;

        public IsoDepTransceiver(IsoDep isoDep, OnMessageReceived onMessageReceived) {
                this.isoDep = isoDep;
                this.onMessageReceived = onMessageReceived;
        }

        private static final byte[] CLA_INS_P1_P2 = { 0x00, (byte)0xA4, 0x04, 0x00 };
        private static final byte[] AID_ANDROID = {
        (byte)0xF0, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06 };

        private byte[] createSelectAidApdu(byte[] aid) {
                byte[] result = new byte[6 + aid.length];
                System.arraycopy(CLA_INS_P1_P2, 0, result, 0, CLA_INS_P1_P2.length);
                result[4] = (byte)aid.length;
                System.arraycopy(aid, 0, result, 5, aid.length);
                result[result.length - 1] = 0;
                return result;
        }

        @Override
        public void run() {
                int messageCounter = 0;
                try {
                        isoDep.connect();
                        byte[] response = isoDep.transceive(createSelectAidApdu(AID_ANDROID));
                        while (isoDep.isConnected() && !Thread.interrupted()) {
                                String message = "Message from IsoDep " + messageCounter++;
```

```
                                response = isoDep.transceive(message.getBytes());
                                onMessageReceived.onMessage(response);
                        }
                        isoDep.close();
                }
                catch (IOException e) {
                        onMessageReceived.onError(e);
                }
        }
}
```

*MainActivity.java*

```
package de.grundid.hcedemo;

import android.app.Activity;
import android.nfc.NfcAdapter;
import android.nfc.NfcAdapter.ReaderCallback;
import android.nfc.Tag;
import android.nfc.tech.IsoDep;
import android.os.Bundle;
import android.widget.ListView;
import de.grundid.hcedemo.IsoDepTransceiver.OnMessageReceived;

public class MainActivity extends Activity implements OnMessageReceived, ReaderCallback {

        private NfcAdapter nfcAdapter;
        private ListView listView;
        private IsoDepAdapter isoDepAdapter;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                listView = (ListView)findViewById(R.id.listView);
                isoDepAdapter = new IsoDepAdapter(getLayoutInflater());
                listView.setAdapter(isoDepAdapter);
                nfcAdapter = NfcAdapter.getDefaultAdapter(this);
        }

        @Override
        public void onResume() {
                super.onResume();
                nfcAdapter.enableReaderMode(this, this, NfcAdapter.FLAG_READER_NFC_A |
NfcAdapter.FLAG_READER_SKIP_NDEF_CHECK,
                                null);
        }

        @Override
        public void onPause() {
                super.onPause();
                nfcAdapter.disableReaderMode(this);
        }

        @Override
        public void onTagDiscovered(Tag tag) {
                IsoDep isoDep = IsoDep.get(tag);
                IsoDepTransceiver transceiver = new IsoDepTransceiver(isoDep, this);
                Thread thread = new Thread(transceiver);
                thread.start();
        }

        @Override
        public void onMessage(final byte[] message) {
                runOnUiThread(new Runnable() {
```

```
                        @Override
                        public void run() {
                                isoDepAdapter.addMessage(new String(message));
                        }
                });
        }

        @Override
        public void onError(Exception exception) {
                onMessage(exception.getMessage().getBytes());
        }
}
```

*MyHostApduService.java*

```
package de.grundid.hcedemo;

import android.nfc.cardemulation.HostApduService;
import android.os.Bundle;
import android.util.Log;

public class MyHostApduService extends HostApduService {

        private int messageCounter = 0;

        @Override
        public byte[] processCommandApdu(byte[] apdu, Bundle extras) {
                if (selectAidApdu(apdu)) {
                        Log.i("HCEDEMO", "Application selected");
                        return getWelcomeMessage();
                }
                else {
                        Log.i("HCEDEMO", "Received: " + new String(apdu));
                        return getNextMessage();
                }
        }

        private byte[] getWelcomeMessage() {
                return "Hello Desktop!".getBytes();
        }

        private byte[] getNextMessage() {
                return ("Message from android: " + messageCounter++).getBytes();
        }

        private boolean selectAidApdu(byte[] apdu) {
                return apdu.length >= 2 &&
                apdu[0] == (byte)0 && apdu[1] == (byte)0xa4;
        }

        @Override
        public void onDeactivated(int reason) {
                Log.i("HCEDEMO", "Deactivated: " + reason);
        }
}
```

# Appendix B

This chapter provides the source code of the Arduino library for PN532 which implements the HCE used in this thesis. The complete source code is available at GitHub[1].

*Android HCE example*

```
% ----------------------------------------------------------------------------

#include <SPI.h>
#include <PN532_SPI.h>
#include <PN532Interface.h>
#include <PN532.h>

PN532_SPI pn532spi(SPI, 10);
PN532 nfc(pn532spi);


void setup()
{
    Serial.begin(115200);
    Serial.println("-------Peer to Peer HCE--------");

    nfc.begin();

    uint32_t versiondata = nfc.getFirmwareVersion();
    if (! versiondata) {
      Serial.print("Didn't find PN53x board");
      while (1); // halt
    }

    // Got ok data, print it out!
    Serial.print("Found chip PN5"); Serial.println((versiondata>>24) & 0xFF, HEX);
    Serial.print("Firmware ver. "); Serial.print((versiondata>>16) & 0xFF, DEC);
    Serial.print('.'); Serial.println((versiondata>>8) & 0xFF, DEC);

    // Set the max number of retry attempts to read from a card
    // This prevents us from waiting forever for a card, which is
    // the default behaviour of the PN532.
    //nfc.setPassiveActivationRetries(0xFF);

    // configure board to read RFID tags
```

[1] https://github.com/Seeed-Studio/PN532/tree/master/PN532

```
    nfc.SAMConfig();
}

void loop()
{
  bool success;

  uint8_t responseLength = 32;

  Serial.println("Waiting for an ISO14443A card");

  // set shield to inListPassiveTarget
  success = nfc.inListPassiveTarget();

  if(success) {

     Serial.println("Found something!");

    uint8_t selectApdu[] = { 0x00, /* CLA */
                             0xA4, /* INS */
                             0x04, /* P1  */
                             0x00, /* P2  */
                             0x07, /* Length of AID  */
                             0xF0, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, /* AID defined on Android App */
                             0x00  /* Le  */ };

    uint8_t response[32];

    success = nfc.inDataExchange(selectApdu, sizeof(selectApdu), response, &responseLength);

    if(success) {

      Serial.print("responseLength: "); Serial.println(responseLength);

      nfc.PrintHexChar(response, responseLength);

      do {
        uint8_t apdu[] = "Hello from Arduino";
        uint8_t back[32];
        uint8_t length = 32;

        success = nfc.inDataExchange(apdu, sizeof(apdu), back, &length);

        if(success) {

          Serial.print("responseLength: "); Serial.println(length);

          nfc.PrintHexChar(back, length);
        }
        else {

          Serial.println("Broken connection?");
        }
      }
      while(success);
    }
    else {

      Serial.println("Failed sending SELECT AID");
    }
  }
  else {

    Serial.println("Didn't find anything!");
  }

  delay(1000);
```

```
}

void printResponse(uint8_t *response, uint8_t responseLength) {

   String respBuffer;

    for (int i = 0; i < responseLength; i++) {

      if (response[i] < 0x10)
        respBuffer = respBuffer + "0"; //Adds leading zeros if hex value is smaller than 0x10

      respBuffer = respBuffer + String(response[i], HEX) + " ";
    }

    Serial.print("response: "); Serial.println(respBuffer);
}

void setupNFC() {

  nfc.begin();

  uint32_t versiondata = nfc.getFirmwareVersion();
  if (! versiondata) {
    Serial.print("Didn't find PN53x board");
    while (1); // halt
  }

  // Got ok data, print it out!
  Serial.print("Found chip PN5"); Serial.println((versiondata>>24) & 0xFF, HEX);
  Serial.print("Firmware ver. "); Serial.print((versiondata>>16) & 0xFF, DEC);
  Serial.print('.'); Serial.println((versiondata>>8) & 0xFF, DEC);

  // configure board to read RFID tags
  nfc.SAMConfig();
}
```

# References


[1] Host-based card emulation. `http://developer.android.com/guide/topics/connectivity/nfc/hce.html`. Accessed on 19th June 2015. ix, 27, 28, 30

[2] Chan, jien-tsai, and wuu yang, advanced obfuscation techniques for java bytecode. journal of systems and software 71.1: 1-10, 2004. 23

[3] Reveilhac, marie and pasquet, marc, promising secure element alternatives for nfc technology. In *Near Field Communication, 2009. NFC'09. First International Workshop on*, pages 75–80. IEEE, 2009. ix, 41, 43

[4] Armoogum, sandhya, and asvin caully, obfuscation techniques for mobile agent code confidentiality. journal of information & systems management 1.1, 83-94, 2011. 23

[5] Globalplatform value proposition for remote post-issuance secure access modules (sam) managementcloud-based nfc mobile payments. `http://www.globalplatform.org/documents/globalplatform_remote_sam_white_paper_nov2011.pdf`, 2011. Accessed on 19th June 2015. ix, 37

[6] Roland, michael, josef langer, and josef scharinger, practical attack scenarios on secure element-enabled mobile devices. near field communication (nfc), 2012 4th international workshop on. ieee, 2012. ix, 21, 22, 48

[7] Atul verma, get into the zone: Building secure systems with arm® trustzone® technology, 2013. ix, 19, 20

[8] Trusted labs, security evaluation of trusted execution environments: Why and how, 2013. 20

[9] Emv payment tokenisation specification – technical framework, version 1.0. http://www.emvco.com/specifications.aspx?id=263, 2014. Accessed on 19th June 2015. ix, 16, 17

[10] Globalplatform made simple guide: Secure element. http://www.globalplatform.org/mediaguideSE.asp, 2014. Accessed on 19th June 2015. 20, 32

[11] Stronger security and mobile payments. dramatically faster and cheaper to implement. http://www.pymnts.com/wp-content/uploads/2014/03/Loop-MobileTokenization-final4.pdf, 2014. Accessed on 19th June 2015. 22

[12] Trusted execution environment - device specifications. http://www.globalplatform.org/specificationsdevice.asp, 2014. Accessed on 19th June 2015. ix, 20, 21

[13] Smart Card Alliance. Mobile/nfc security fundamentals. http://www.smartcardalliance.org/resources/webinars/Secure_Elements_101_FINAL3_032813.pdf, 2013. Accessed on 19th June 2015. 41

[14] Atmel. Requirements of iso/iec 14443 type b proximity contactless identification cards. http://www.atmel.com/images/doc2056.pdf, 2005. Accessed on 19th June 2015. ix, 26, 28

[15] Jackson Beale. Paypal payflow payment gateway diagram. https://www.pinterest.com/pin/211035932512582530/, 2014. Accessed on 19th June 2015. ix, 10

[16] Martijn Bolhuis. Using an nfc-equipped mobile phone as a token in physical access control. http://essay.utwente.nl/65419/1/thesis_nfc_martijn_bolhuis_final.pdf, 2014. Accessed on 19th June 2015. ix, 26, 27

[17] SD card. Sd association introduces smartsd memory cards with swp interface for nfc. https://www.sdcard.org/press/SD_Association_

Introduces_smartSD_Memory_Cards_with_SWP_Interface_for_NFC_FINAL_6-4-2013.pdf, 2013. Accessed on 19th June 2015. 42

[18] Sarah Clark. Simplytapp proposes secure elements in the cloud. http://www.nfcworld.com/2012/09/19/317966/simplytapp-proposes-secure-elements-in-the-cloud/, 2012. Accessed on 19th June 2015. 45

[19] Francisco Corella and Karen Lewison. A comprehensive approach to cryptographic and biometric authentication from a mobile perspective. http://pomcor.com/whitepapers/CryptographicAuthentication.pdf, 2013. Accessed on 19th June 2015. 11

[20] PCI Security Standard Council. Requirements and security assessment procedures v3.1. https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-1.pdf, 2015. Accessed on 19th June 2015. ix, 12

[21] Chris Cox. Trusted service manager: The key to accelerating mobile commerce. http://www.firstdata.com/downloads/thought-leadership/fd_mobiletsm_whitepaper.pdf, 2009. Accessed on 19th June 2015. ix, 34

[22] emarketer analyst. Us mobile payments to top $1 billion in 2013. http://www.emarketer.com/Article/US-Mobile-Payments-Top-1-Billion-2013/1010035, 2013. Accessed on 19th July 2015. 2

[23] VISA Inc. Visa to enable secure, cloud-based mobile payments. http://investor.visa.com/news/news-details/2014/Visa-to-Enable-Secure-Cloud-Based-Mobile-Payments/default.aspx, 2014. Accessed on 19th June 2015. 45

[24] ECMA International. Near field communication wired interface (nfc-wi), standard ecma-373. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-373.pdf, 2012. Accessed on 19th June 2015. 37

[25] Eduard Kovacs. Tokenization: Benefits and challenges for securing transaction data. http://www.securityweek.com/tokenization-benefits-and-challenges-securing-transaction-data, 2014. Accessed on 19th June 2015. 16

[26] MasterCard. Secure elements. https://mobile.mastercard.com/Partner/MobilePayPass/SecureElements. Accessed on 19th June 2015. ix, 36, 38, 39

[27] MasterCard. Mastercard to use host card emulation (hce) for nfc-based mobile payments, 2014.
http://newsroom.mastercard.com/press-releases/mastercard-to-use-host-card-emulation-hce-for-nfc-based-mobile-payments/
Accessed on 19th June 2015. 45

[28] Pierluigi Paganini. Near field communication (nfc) technology, vulnerabilities and principal attack schema, 2013.
http://resources.infosecinstitute.com/near-field-communication-nfc-technology-vulnerabilities-and-principal-attack-schema/
Accessed on 19th June 2015. x, 47

[29] Olivier Potonniée and Ming Yin. Secure element api. http://opoto.github.io/secure-element/, 2014. Accessed on 19th June 2015. 27

[30] Pardis Pourghomi, Gheorghita Ghinea, and UK Uxbridge. Cloud-based nfc mobile payments, 2013. 39

[31] Ricardo J. Rodríguez and Pepe Vila. Practical experiences on nfc relay attacks with android: Virtual pickpocketing revisited. http://conference.hitb.org/hitbsecconf2015ams/wp-content/uploads/2014/12/WHITEPAPER-Relay-Attacks-in-EMV-Contactless-Cards.pdf, 2014. Accessed on 19th June 2015. 67

[32] Tokenization Taskforce PCI Security Standards Council Scoping SIG. Pci dss tokenization guidelines. https://www.pcisecuritystandards.org/

documents/Tokenization_Guidelines_Info_Supplement.pdf, 2011. Accessed on 19th June 2015. ix, 18, 19

[33] George Wallner. Stronger security and mobile payments-dramatically faster and cheaper to implement. http://www.pymnts.com/wp-content/uploads/2014/03/Loop-MobileTokenization-final4.pdf, 2014. Accessed on 19th June 2015. ix, 17, 18, 19

[34] Tamara Wilhite. Differences in a b for iso 14443. http://www.ehow.com/info_10004163_differences-b-iso-14443.html, 2014. Accessed on 19th June 2015. 26