NTNU – Trondheim
Norwegian University of
Science and Technology

Relay attacks of NFC smart cards

Xiqing Chu

Master in Security and Mobile Computing
Submission date: June 2014
Supervisor: Colin Alexander Boyd, ITEM

Norwegian University of Science and Technology
Department of Telematics

NTNU – Trondheim
Norwegian University of
Science and Technology

Relay Attack of NFC Smart Card

Xiqing Chu

Submission date: June 2014
Responsible professor: Colin Boyd, ITEM, NTNU Tuomas Aura, Aalto University
Supervisor: Sandeep Tamrakar, PhD Researcher, Aalto University

Norwegian University of Science and Technology
Department of Telematics

Title: Relay Attack of NFC Smart Card

Student: Xiqing Chu

Problem description:

This thesis would study and understand ticketing system based on DESFire cards. Student would investigate whether replay attack using two proxy devices is possible or not. After establishment of attack, student observes what would be the consequences of the replay attack on public transportation system and the valuable assets attackers can profit on during the attack.

Student would develop Android programs on NFC based phones to perform research. Testing environment would be built on legacy Java code. Possible attacks experiment would be carried on lab ticketing system. Bases on the findings from research, student will try to recommend solutions to improve the ticket protocol so that this system is unaffected by the relay attack or at least minimize the profit that attacker can get.

Responsible professor: Colin Boyd, ITEM, NTNU Tuomas Aura, Aalto University

Supervisor: Sandeep Tamrakar, PhD Researcher, Aalto University

Abstract

Contactless smartcards are used for various security-critical applications, such as identification and access, ticketing, and payment. Smart cards are vulnerable to a relay attack where the card is read e.g. from the victim's pocket and used remotely for unauthorized purposes. Some common card technologies are also vulnerable to man-in-the-middle attacks where the attacker manipulates the communicated data, typically to the card holder's advantage. This thesis project has two goals: (1) to implement relay attacks against NFC smart cards, such as transport tickets based on Mifare DESFire EV1 tehcnologies, and (2) to investigate how time measurements could be used to detect and prevent such attacks. We have utilized off-shelf smart phone NFC features and built a prototype of NFC smart card relay attack for experiments. We also conducted time measurements on both direct DESFire APDU transactions and transactions through relay attack channels. Finally we proposed counter-measures to relay attack such as timing on each transaction and check UID of NFC tags in transaction.

Preface

During the past two years, NordSecMob international master program offered me a splendid opportunity to discover the western world. I am glad I have achieved a successful student life and this program broadened my horizon. The valuable experience in the past two years will company me to the future.

Special thanks goes to my instructor Sandeep, professor Tuomas from Aalto University and professor Colin from NTNU. Sandeep as an instructor offered me numerous amounts of help during the exploration of this thesis. He gave me advice on both life and academic research.

Life will go on.

June 30, 2014

Espoo, Finland

Contents

List of Figures

List of Tables

Abbreviations and Acronyms

3K3DES	Triple Key, Triple DES Encryption
ATS	Answer to Select
APDU	Application Protocol Data Unit
FWT	Frame Waiting Time
HCE	Host Card Emulation
MITM	Man-in-the-middle Attack
NFC	Near Field Communication
OS	Operating System
POS	Point of Sale
PC	Personal Computer
PCD	Proximity Coupling Device
PICC	Proximity Integrated Circuit Card
RFID	Radio Frequency IDentification

Chapter 1
Introduction

Near Field Communication (NFC) technology exists in our daily life and has integrated into many aspects of our activities(See Chapter 2). Originally developed as a subset of RFID technology, NFC features a low range but high frequency wireless communication between two devices. Contactless smart cards, or say, NFC smart cards are nowadays used in many applications including ticketing system like bus tickets, payment systems like NFC embedded Master card[All07] and even identification cards which can unlock the door. Already there have been plenty of researches on the security features of NFC applications[KW05, HB06, OP11, SVV10, MLKS08, Mul09].

The emergence of NFC technology is far earlier and tools to operate NFC are dedicated devices from vendors. In the past, business solution programmers or researchers have the resource to explore the pattern of NFC through dedicated devices and proprietary protocols. Normal users possessing NFC cards can only check their information of cards and read or write information through official provided service points. For example, the bus card users can only check their balance in cards via top-up machines. However, the situation has changed since smart phones with NFC chips come into common user's life. A normal user has no knowledge of NFC specifications can read/write his card and even customize his NFC tag with the help of smart phone applications.

With the unveiling of the NFC technology, a piece of hardware enabled NFC features can easily perform the following two important tasks in NFC, *read/write* mode and *card emulation* mode. As the NFC technology is labeled as a low distance and encryption protected communication, we are not the first one to perform relay attack[Han05]. However, this thesis might be the first to discover the possibility to make a feasible relay attack by taking advantage of NFC feature on off-shelf Android[1] smart phones.

[1] http://www.android.com/

1.1 Problem statement

> *John leaves home in the morning with his bus travel card on the desk with a smart phone attached to it. When he catches up the bus and trying to pay for the trip, instead of showing his travel card, he shows another smart phone from his pocket to the reader. Surprisingly, the reader accepts it and beeps as normal. The bus driver cannot believe his eyes and thinks it must be some dark magic. What he does not know is that John's card at home updates the value simultaneously already via cellular network by the smart phone put on it. John doesn't cheat, all he does is a simple relay attack on NFC technology.*

Our goal in this thesis research is to prevent relay attack on NFC smart cards, Forcing real contactless smart card to be physically presented to service points, e.g. bus card to bus card reader. Traditionally an attacker who is performing a relay attack needs to sit in between two communication parties. As we already know the specification of NFC is to communicate in an extremely low distance by 1-10cm, attackers cannot derive a possible relay attack without enlarging this distance. With the help of NFC hardware operating in card emulation mode, we can already impersonate the interface of a real card[Han05]. Connected via a third channel, another piece of NFC hardware can read/write information accordingly to the real card somewhere else. Thus, the attacker can fully hijack the communication between reader and card and sniff the traffic.

As nowadays NFC chips are becoming a standard part in smart phones, it lowers the difficulty to get a useful device to perform card emulation. Our research would use a modified Android operating system to control the NFC chip in smart phone over certain level. Standard Android room does not provide us APIs to emulate NFC tags freely, only modified Android can do emulation 4. One smart phone is used to impersonate the card and the other is going to read/write transactions to the real card. Transactions are recorded for further understanding. Communication between two smart phones is via Wi-Fi or Bluetooth channel. This communication channel covers a far distance of 46m indorr or 92m outdoor if using Wi-Fi 802.11, and 31m for Bluetooth 2.1[FP05].

By extending existing library of Android smart phone SDK via Java Reflection, we build useful prototype applications to facilitate our relay attacks. The documentations of predecessors are studied to better our understanding of protocols.

As part of the evaluation, we choose one type of NFC smart cards — Mifare DESFire EV1 as target NFC system. A demo reader program is written to normally

interact with DESFire cards. We record each transaction and command between reader and card as well as time for completing each transaction. After inserting our relay attack devices in between, we evaluate the impact of delay in transactions introduced by the relay attack. And from the data collected we evaluate the relay attack channel delay on original transaction. Together with existing smart card protocols we propose our strategy to tackle with this type of relay attack.

During the implementation and real field demonstration, we also study the failures of attack and security mechanisms we can put forward to prevent emulation based relay attacks.

1.2 Structure of the Thesis

This thesis consists of eight parts.

From Chapter two we present the background knowledge of NFC and related terminologies. Useful previous studies and some projects which lay down the fundamental of our research are revisited. Important terminologies are also introduced to reader of this article.

Chapter three would have an in-depth look of Mifare DESFire EV1 smart card features. Protocol level details are covered in this chapter. These protocols are important for later-on implementation.

Chapter four would be a bird-eye view of our NFC relay attack design. We analyze some prerequisite before implementation. We also try to optimize our architecture to achieve fast speed of information exchange between devices. Chapter five is rather standalone part; it describes the implementation details of relay attack. Programming codes snippets and other dependencies are put in this chapter. As a demo, work flows of the relay attack and screen shots are also included.

In chapter six and seven we collect corresponding performance and evaluation metrics and start to explore the impact we introduced. A comparison between different transactions, different data communication channel is taken place. The comparison of time of transactions between original card and emulated card is well discussed. Also we would talk about little the future of relay attack and prevention of it.

Chapter Eight is the conclusion of this thesis.

Chapter 2
Background and Previous Studies

Near Field Communication (NFC) is a technology which enables low-range, low-power, lightweight information exchange between devices. As a subset of RFID technology, it is composed of a set of specifications and protocol stacks brought up front by NFC Forum[Def06].

NFC roots deeply in our daily lives as it is considered an important part of great evolution: Internet of Things (IoT)[AIM10]. NFC technology features a touch paradigm which is easy to use and intuitive[JTSM07], thus it is considered to be one of the enabling technologies of IoT[AIM10]. Users generally touch one device to another to activate NFC service and exchange information. Three operating modes are available: *read/write* mode, *card emulation* mode and *peer-to-peer* mode. Among those three modes, *read/write* mode is the common scenario where ordinary users of NFC enjoy the convenience of NFC. A user can use a safe NFC key to lock or unlock his door and authenticate him to the access control system[MLKS08]. Bus cards with NFC embedded chips can keep values and credit information which is used for transactions[RKSH07]. Payment cards with NFC feature offer users another option than contact-based bank cards[MB09]. NFC tags contain structured information that can be used as portable information storage, e.g. business cards which can be read by scanning it with NFC readers. NFC is also used as a part of the biometric passports to store private information about an individual[AM10].

Although in our daily scenario NFC is easy to use and all operations on NFC tags happen in a contactless yet invisible manner, our interests in security of NFC technology rises. As Kfir Z et al. has pointed out in paper [KW05], relay attack on NFC has a potential to become a popular attack in recent years and can cause potential financial loss. As the tools for investigating NFC becomes more advanced and easy to get, we want to explore the security limits of NFC and especially in our topic, relay attack on NFC.

In the **section 2.1** we would introduce the basic definitions, special features and

standards in NFC and NFC operation modes. In the **section 2.2** we discuss about a feature of NFC—Frame Waiting Time— and the related previous studies which lay down the foundation of our relay attack research.

2.1 Near Field Communication

In this section we introduces the fundamental knowledge of NFC to our audience. The topics discussed here cover from NFC standards and specifications to more practical aspects such as NFC operation modes and using smart phone as a NFC tool. Some tool libraries are also covered here as we bring together the necessities of practical programming of NFC smart cards.

2.1.1 NFC Definition and Smart Cards

NFC technology is driven and standardized by NFC Forum to promote the interoperability of NFC devices and services. This alliance is initially formed up by Nokia, Philips and Sony in the early year of 2004. The NFC protocol stacks, architecture and message exchange format are brought forward and defined in the following years until 2010. NFC Data Exchange Format (NDEF) is one of those outcomes of NFC specifications.[Def06]

The biggest distinguish feature that separates NFC and other communication technology such as Bluetooth[1] is that NFC is only meant for low-band width information exchange. As a subset of RFID technology (Radio Frequency IDentification), NFC only focuses on the low-range communication. To summarize the characteristics of NFC, it is a standard which operates on frequency 13.56 MHz and has a close distance range of 1-2cm. Its target is to exchange lightweight information between two devices at a transmission speed of 106, 212 or 424 kbps.[Wan06] [COO13]

NFC devices in communication are two parties both in compliance with NFC standards. One party is the initiator and the other is defined as recipient. Usually the case is that the recipient is a NFC smart card and initiator is a NFC reader. The initiator is idle until a passive recipient enters the communication area and is selected to communicate with. The initiator creates an electromagnet field to power the powerless recipient and starts the data exchange. Once transmission is done, as soon as the recipient is out of communication range, the initiator is returned to idle again.

Smart card is one of the most commonly seen NFC tags (recipient) in the use cases[Ort06]. It has an onboard file system and micro-possessing chip embedded to it. It has a size of regular bank card and does not contain a power source. [HB06]

[1] http://www.bluetooth.com/

[KW05]. A microprocessor unit together with a file system form up an access control which limits the access of data on smart card to only authorized users. Some vital data can be set as read-only and be visited by authorized users with correct key. As it is targeted at becoming a portable computing device with small amount of data storage, smart card is extremely suitable for ticketing system and crediting system. For example, bank cards, bus travel cards and identity cards at access control points. Credit data stored on smart cards can be rewritten and the structure of file is flexible and thus it is widely used in commercial systems[KW05]. As the smart cards evolve in generations, some of the old types are already compromised such as MIFARE DESFire D40 (MF3ICD40)[OP11]. In this thesis our studying target is Mifare DESFire EV1 smart card, the successor of DESFIRE D40. It is a mature smart card type and its technical details would be covered in **Chapter 3**.

2.1.2 NFC Standard Protocols

Although the smart card implementation details remain in proprietary to vendors (such as DESFire), the standard of NFC protocol stack is open and widely studied. NFC enabled devices shall respect and fully implement the requirements in order to be compatible with devices from different manufactures.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) have documented standards for NFC. In our thesis study, the most relevant components are protocol standards for NFC enabled devices. These are parts of the NFC features in modern NFC equipped smart phones. From top to bottom, ISO 7816-4, ISO14443-4, ISO14443-3 type A and B, ISO 14443-2 and ISO 14443-1 protocols are used to form whole protocol stack of NFC enabled smart phones. All details of ISO 14443 protocol suits can be found in the article [CH07]. This article is for both entry level readers and experts. Here we give out some summarize of four layers of protocol stack in our context.

From the below of the protocol stack, ISO 14443-1 describes the physical characteristics of contactless integrated circuit cards. These cards are called Proximity Integrated Circuit Chip (PICC) in a standard term. Above of it is the ISO 14443-2 protocol defines the behavior of radio frequency signal interface. It defines the power and the frequency to conduct NFC transactions.

Starting from ISO 14443-3 level, it divides into two different flavors, `type A` and `type B`. Each type provides initialization phase of NFC communication with different modulation methods and different code schemes. Thus devices which are implemented for one type of protocol cannot successfully communicate with the other type. However, despite the differences, the transmission protocol of ISO 14443-4 can be established on either of them.

ISO 14443-4 transmission protocol is the main protocol layer where exposed to NFC business logic programmers on Android smart phones nowadays[CKP+12]. It features data block exchange and multi-activation mechanisms to enable successful transmitting of messages between PICC and PCD (Proximity Coupling Device, also known as card reader). The visible appearance of messages transmitted is in bytes thus is quite raw.

Above the transmission protocol layer, there is a final layer of ISO 7816-4 where the Application Protocol Data Unit (APDU) is defined. This layer helps to encapsulate and abstract the message format and thus the programmers are less troubled with exploring raw data in bytes.

2.1.3 NFC Operating Modes

One NFC enabled device is a certain kind of device which can perform NFC transactions with the other NFC device. One Nexus S phone with NXP PN544 chip[Sem10] can be considered as an appropriate NFC enabled device. As for the art status quo, NFC enabled devices shall be able to perform all or one of the following three operating models: *read/write mode, peer-to-peer mode* and *card emulation mode*[COO13].

- *Read/write mode* is the most commonly seen existing mode. An active reader initiates a session to exchange data with a passive tag. The passive tag itself is powerless and thus requires the reader to provide driven energy in an electromagnet field. The active reader extracts useful information through the session and creates/updates data on the tag. For example, a bus terminal reader makes transactions with a user travel card.
- *Peer-to-peer mode* is a data exchange communication mode exists largely between two NFC enabled devices. It is a bidirectional channel which both parties can initiate requests and retrieve information through the channel. The applicable scenario is like this: two smart phones use NFC to exchange parameters about Bluetooth settings and then establish Bluetooth channel afterwards.
- *Card emulation mode* is the main focus in our thesis. One NFC device operates in this mode is tempting to emulate a passive tag. As a passive tag it can be read and write like a normal NFC tag. This emulation is often divided into two different flavors if the NFC device is a smart phone: emulation with a Secure Element (SE)[SVV10] or direct emulation with client applications. The SE in a smart phone is not easy to be tempered with and provides virtual smart card functionality. If directly emulating a tag with client application, programmers have to manually manage the ISO 14443-4 layer commands and secure the critical information stored on the phone. The direct emulating approach gives

us large freedom to derive a practical relay attack on NFC applications. The phone emulating a passive tag can absorb and analyze the message that card reader sends to it. It can also forward the messages to another device through a second channel and make real transactions with a real smart card. This emulated NFC tag is the vital element to the process of relay attack.

2.1.4 Mobile Phone and NFC

With the converge of phones and PDAs, no longer are the mobile phones specialized to only making phone calls and text messaging. Smart phones, with a growing number and a growing functionality, have become personal data processing hubs[ZN06]. New services emerge in the market as smart phones adopt technologies such as Wi-Fi, Bluetooth, GPS[2] and NFC.

However, the role for smart phone to play is far more than personal computing. Smart phone is a ubiquitous input device for personal daily life that gives user a unified access point to different services[BBRS06]. Home appliances, vender machines, and other new services use smart phone as input. Storing with personal information, smart phone has become the true centralized service hub for a personal consumer.

As we have covered in previous section and introduction, mobiles phones with a suitable NFC chip can provide a user with experience of difference NFC related services. A user can check remaining value and travel history in a NFC bus card from a smart phone. He/She can also purchase a concert ticket by scanning NFC tags at ticketing office. These features utilized the *read/write* mode of NFC in smart phone. Furthermore, credential information can be stored on a smart phone so that it turns out to be a personal bank card. Whenever the user wants to purchase with bank card, he can instead let the POS (Point of Sales) machine to scan his smart phone[DM08]. This feature utilized the *card emulation* mode of NFC in smart phones.

The flexible NFC features in smart phone open the gate of booming business around NFC-centered services and applications. However, numerous security issues also been pointed out in recent study including spoof of tag content, deny of service and our thesis topic, relay attack. These attacks are not only target at NFC-subsystem of smart phone, and also target at weak applications built on top of NFC[Mul09].

In our research of this thesis, thanks to the advancement of smart phones in recent years, we implemented a prototype of relay attack within two off-shelf market smart phones. Only minimum software changes are made to API and original operating system(See chapter 5). As the relay attack hardware (smart phone) and software (smart phone apps) become more available to end users, our audience can realize the

[2] http://www.gps.gov/

urgent is not only addressing the issue of relay attack of NFC, but also the preventing of such attack.

2.2 Relay attack and Frame Waiting Time

In this section we go through the previous studies about the vulnerability of NFC especially in the context of relay attack. We point out the imperfect implementing of ISO 14443 protocols and how it fails to detect relay attack in practical scenario. We would introduce the Frame Waiting Time, a useful parameter and its role to play in preventing relay attacks on NFC.

2.2.1 Relay Attack on NFC

Sportiello et al. [SC13] describes a long distance of relay attack on NFC card which gives out a solution that a reader can establish a transaction in a controlled channel by relay attacker with a NFC card in a long distance. Within specially designed hardware, they can route the APDUs between the card and reader in a long distance. Thus the NFC smart card is no longer required to be proximate enough to reader, but geologically locates in any area.

Issovits et al. [IH11] have conducted a study on the weaknesses on ISO 14443 protocol stack. They implemented a prototype of RFID relay attack using an off-shelf mobile phone and a specially designed RFID emulator tag. The study suggests two countermeasures to relay attack: a protocol binding of distance detection of NFC communication or additional checks on initial parameters in an encrypted way. However, these amendments need to modify existing protocol and may break the existing services already in use.

Jason Reid et al. [RNTS07] suggests a distance-bounding protocol not only on NFC but on all RFID based contactless smart cards. It utilized a symmetric key encrypted protocol to protect the information from altering by attacker and uses a timing mechanism to detect if the channel is of high latency. This novel approach also introduces new elements to the existing protocol if we want to implement it. Thus this can also break down the existing services.

2.2.2 Relay Attack on NFC with Smart Phone

Roland et al.[RLS12] have conducted a research targeting at NFC Secure Elements on mobile phone. They propose to conduct a relay attack by implanting a malicious application on user smart phone to forward NFC transactions out of Secure Elements and to another location located on a global accessible place. This article inspires us about forwarding the NFC transactions outside the smart phone itself to give attacker

more flexibility of placing the actual attack logic, and more complex operations on original NFC transactions.

Markantonakis et al. [Mar12] have successfully conducted a practical relay attack with Nokia cell phones with Bluetooth feature to forward APDUs in between thus forms up a relay attack on NFC smart cards. In the article author implemented a prototype to justify his arguments. This article inspires us the feasibility of using NFC enabled smart phones to conduct a relay attack. Our research is based on the same concept but instead, we are seeking for a more general approach of using Android based cell phones and applications. Thus lower the complexity of building attacking app and also gives more freedom of choosing suitable attacking devices.

2.2.3 Frame Waiting Time and Relay Attack

As we are not only target at lowering the complexity of NFC relay attacks, we also want to explore the future of preventing relay attacks on NFC. As the existing ISO 14443 series protocols are widely used in contactless smart cards nowadays and we want to utilize this protocol instead of building up a new protocol. New protocol can result in incompatibility of existing services and redesign of end point hardware.

Frame Waiting Time (FWT) has come into our sight as it is already a specification in ISO 14443 protocols. It describes the waiting time that reader can expect of NFC response from card before this reader considers that transaction is invalid. After sending the request from the reader, a card shall respond quicker than the FWT. As described in reference[Han05], a FWT in ISO 14443-4 specification is calculated by formula:

$$FWT = (256 \times 16/fc) \times 2^{FWI} \tag{2.1}$$

Right now the *fc* of DESFire EV1 card (A typical NFC smart card, see **chapter 3**) has a fixed value of 13.56×10^6 and FWI integer with a value decided in ATS (Answer to Select) response in initialization phase (ISO 14443-3). The default ATS of a DESFire real card is 0x08 and thus, the FWT is a value of 77.33ms.

As the FWT limits the time the reader expect for a response. It requires a relay attack to complete in a certain amount of time. As we know that relay attack using two smart phones at least requires forwarding of messages on a second communication channel and that could make the relay attack detectable if the NFC response is slower than usual(high latency). We will discuss in practical implementation, to which extend can attacker accelerate the progress. We also point out how a system designer of NFC applications using FWT to prevent most relay attacks in Chapter 7.

Chapter 3
DESFire EV1 Specifications

This chapter serves as a bridge between abstract protocol definition and implementation of NFC attacks. As the subject of our experiment is Mifare DESFire EV1 smart card, we describe the characteristics and specification of this type of card in this chapter. As a mature NFC smart card after its predecessor DESFire D40[OP11], DESFire EV1 card brings security and usability to a new level. The target of this chapter is to cover the most relevant areas of DESFire EV1 in the context of relay attack. We here cover the topics of card identification, file system, key management, authentication process and finally, we gives out examples of typical DESFire transactions.

Much of the information about Mifare DESFire comes from officially published document[NXP12]. Information about DESFire smart card itself is covered by experiments, open source community online documents and its predecessor DESFire D40[OP11]. The example of transaction is produced by experiments we conducted on real card.

3.1 DESFire EV1 Card Identification

As the document[NXP12] states, before we enter the ISO 14443-4 protocol for transactions, DESFire smart card goes through a standard ISO 14443-3 (type A) anti-collision and card selection phase (card identification). This phase is aiming at exchanging critical information for establishing transmission phase, and separating and choosing correct card for transaction. Holder of the card may have presented to the reader a wrong card or a card with wrong physical parameters; these can all be detected by ISO 14443-3 phase and it prevents further wrong steps from happening.

As we summarize from the document[NXP12], the card identification process of ISO 14443-3 shall go through the following steps illustrated in the Figure 3.1. Here we identify the smart card reader as PCD and smart card as PICC(See **Chapter 2**.

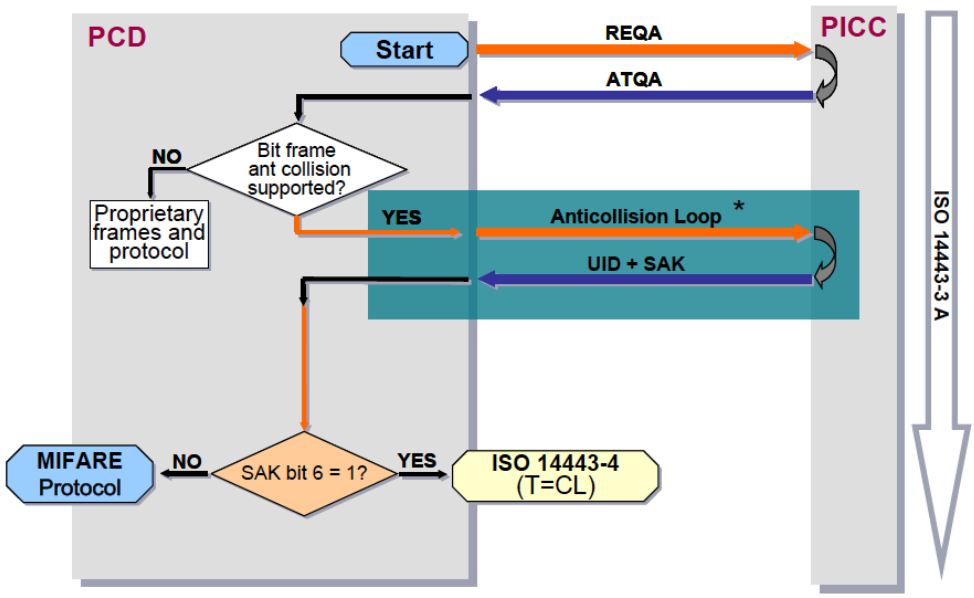

Figure 3.1: ISO 14443-3 Card Identification Process[NXP12].

PCD is in idle state waiting for PICC to approach it. It sends out REQA (REQ type A) command to poll for information from any PICC that comes close enough. As a PICC comes near PCD, it immediately responses with ATQA (ATQ type A). This ATQA contains anti-collision information but the content is generally ignored by PCD, see [NXP12]. Then it enters an anti-collision loop where the SAK value and UID value of a card is checked to distinguish different type of cards. After checking the SAK value, ATS is checked and we now can enter the ISO 14443-4 transmission. Definition and functionality of ATQA, SAK, UID and ATS are covered in Table 3.1

Different smart cards have different parameters. Emulated smart card has different parameters to real cards. Thus the behavior of reader can vary because of the card type. SAK bit 3 indicates the length of cascaded UID bit, if it is set to 0, it is complete (7 byte UID). If it is set to 1, it is not complete. SAK bit 6 indicates which protocol to be used for transmission. It is set to 1 to indicate that transmission protocol is ISO 14443-4 compliant; otherwise it is a proprietary protocol. ATS value is by default 0x80 according to ISO 14443. But it can be customized which states in the NXP document[NXP12]:

Name	Definition
ATQA	Answer to Request type A.
SAK	Bit 3 : indicates the length of UID. Bit 6 : indicates the ISO 14443-4 compatibility.
UID	Unique identifier of PICC.
ATS	Answer to Select. Can be customized. Default : 0x80

Table 3.1: ATQA, SAK, UID and ATS in Card Identification

As the ATS of different MIFARE ICs can be customized, it is certainly not advisable to rely on the ATS to differentiate the IC type. NXP advises to keep the default value of the ATS to avoid any privacy attack based on the information in ATS.

We here present two examples about the detailed difference about a real card and an emulated card in the phase of card identification(captured from experiments):

```
Random DesFire Card
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 03  44
      UID (NFCID1): 04  6f  0f  c2  80  26  80
     SAK (SEL_RES): 20
              ATS: 75  77  81  02  80

Nexus S
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 00  04
      UID (NFCID3): 08  c3  21  4e
     SAK (SEL_RES): 60
              ATS: 78  33  88  00
```

Figure 3.2: ATQA, SAK, UID and ATS for Real Card and Emulated Card

Here the ATQA is generally ignored by reader, but the UID is definitely different for real card and Nexus S phone emulated card. Nexus S has a random and shorter version of UID (4 bytes). A DESFire EV1 real card has a unique 7-byte UID which

has been burnt into card while being manufactured (This special feature is discussed in **section 5.4**). SAK for DESFire EV1 real card is `0x20`, which indicates it is compliant with ISO 14443-4 transmission protocol and ATS is customized to `0x75 0x77 0x81 0x02 0x80`. For Nexus S emulated card, ATS is `0x78 0x33 0x88 0x00`. Both ATS values are different from default value `0x80`.

3.2 DESFire EV1 File System

Our research project requires us to establish a prototype of reading and writing on a DESFire EV1 card. Thus the internal file system and access control on the DESFire contactless card is of vital importance to our discussion. Both authentication key and critical information are stored on the on-chip persistent memory of DESFire EV1 card and protected by access control. This section will bring knowledge of DESFire EV1 file system to our audience.

DESFire EV1 smart card has a non-violate 4kbyte memory which can endure 100,000 cycles of reading/writing. The memory accessing is fast with 2ms of write time (1ms erase, 1ms write). Data stored on the card can retain for over 10 years. [NXP12]

On each DESFire EV1 card, 28 applications can exist simultaneously. They are numbered in a form of 3-byte AID (Application ID). For each application, 16 files can be created. Furthermore, each application has a vector of 14 different keys (numbered from 0 to 13). These keys can be used to indicate the different access rights to each file and file operations (read, write, delete).

Name	**Definition**
Application	28 in each card. Distinguish by 3-byte AID.
File	16 files each application.
Key	14 keys each application. For access control purpose.

Table 3.2: DESFire EV1 Smart Card Structure

There are different file types existing on the DESFire EV1. Standard data file, which stores unformatted user data. Backup data file, which stores unformatted data and integrated with backup mechanism. Value file stores a 32-bit integer. Linear record file stores information about structured user data. And finally cyclic record file, a file can store structured data and it overrides old data when space limit is reached. Each file has a encoded identifier as in table 3.3.

Name	Encoded Identifier
Standard Data File	0x00
Backup Data File	0x01
Value File	0x02
Linear Record File	0x03
Cyclic Record File	0x04

Table 3.3: DESFire EV1 Smart Card File Types

As each file on the DESFire smart card is protected by an access control system, reading and writing to the files need to verify corresponding keys. During key authentication, a temporary key is generated to encrypt the transmission. We cover this knowledge in the following section.

3.3 Authentication and Key System

In the previous section we have mentioned that accessing contents on DESFire EV1 smart card needs to pass an access control system. The accessing control system exists on three level, `PICC level (card level)`, `application level` and `file level`. The keys stored on DESFire card would determine that if the reader has certain rights to read/write content or delete/create application.

3.3.1 Operations and Authentication

Command	Description
Create Application	Creates new applications on the PICC.
Delete Application	Permanently deactivates applications on the PICC.
Get Applications	Returns the Application IDentifiers of all active applications on a PICC.
Select Application	Selects one specific application for further access.
FormatPICC	Releases the PICC user memory.
Get Version	Returns manufacturing related data of the PICC.

Figure 3.3: PICC level commands

On PICC level, a reader can try to create/delete application and even format PICC (format whole card to factory setting). These operations are critical thus shall check PICC master key before the reader can take actual action.

Command	Description
Get FileIDs	Returns the File IDentifiers of all active files within the currently selected application.
Get FileSettings	Get information on the properties of a specific file.
Change FileSettings	Changes the access parameters of an existing file.
Create StdDataFile	Creates files for the storage of plain unformatted user data within an existing application on the PICC.
Create BackupDataFile	Creates files for the storage of plain unformatted user data within an existing application on the PICC, additionally supporting the feature of an integrated backup mechanism.
Create ValueFile	Creates files for the storage and manipulation of 32bit signed integer values within an existing application on the PICC.

Figure 3.4: Application level commands

On application level, different files can be created and deleted. Each application has an application master key. If a reader wants to do mentioned operation in table 3.4, it shall possess a correct application master key and pass the authentication check.

Command	Description
Read Data	Reads data from Standard Data Files or Backup Data Files.
Write Data	Writes data to Standard Data Files or Backup Data Files.
Get Value	Reads the currently stored value from Value Files.
Credit	Increases a value stored in a Value File.
Debit	Decreases a value stored in a Value File.

Figure 3.5: File level commands

On file level, multiple operations can be performed. Read/write and credit values are all subject to the file key authentication. Application can choose from key vector (contains 14 keys) which key is used for write and which key is used for read. Reader shall authenticate itself with corresponding key before the action can be taken.

3.3.2 Authentication and Session Key

The purpose for authentication with key is not only to grant authorization to specific actions, moreover, it lets the reader and smart card to establish a session key which is used for encrypting transmission. Without the session key, though the attacker can captured the traffic of transaction, he is not able to interpret the content. The encryption method available for choosing is DES, AES and 3K3DES (triple key, triple DES). For DES and AES, 16-byte length of key is required. Only first 8 bytes from 16 bytes is used for DES. For 3K3DES, a 24-byte key length is required. But as we reuse part of the key, an actual 16-byte long key is used in practice[NXP12].

On PICC level, we negotiate a proper session key before the transmission section starts. DESFire card (PICC) always performs encryption and reader (PCD) always performs decryption. The data to be encrypted is filled with 0x00 to meet a length requirement. If the data is 0x00, it is filled with a 0x80 byte, then 0x00 to meet the requirement. All encryption is performed in CBC(Cipher Block Chain) mode.

First, the reader initiate a command to indicate that it wants to enter the authentication steps. It wants to authenticate with the PICC card master key `K`. Choosing method is either DES or 3DES. Then card prepares a random number `rndA` (8 bytes) and encrypts it with `K` with agreed encryption cipher.

$$m = Encrypt_K(rndA) \tag{3.1}$$

Upon receiving the encrypted message `m`, reader decrypts it with PICC master key `k` stored in its memory (maybe different from `K`, thus the following authentication will fail) and shifts result left for 8 bits. The shifted result is represented as `rndA'`. Then the reader creates another 8 byte random number `rndB` and connect `rndA`,`rndB` into `rndC`. The reader decrypts the `rndC` with `k` and sends back.

$$\begin{aligned} &Decrpyt_k(m) = rndA \\ &rndA' =<< rndA \\ &rndC = rndB|rndA' \\ &then, p = Decrypt_k(rndC) \end{aligned} \tag{3.2}$$

The card receives `p` and encrypts it. It checks if the `rndA'` is correct shifted result of `rndA`. If not correct, the reader possessed PICC master key `k` is not correct. Thus,

the reader is not authorized. If the authorization is successful, the card will shift the `rndB` left by 8 bits to become `rndB'` and encrypts it and sends it back.

$$\begin{aligned} &Encrypt_K(p) = rndC \\ &rndB' = << rndB \\ &q = Encrypt_K(rndB') \end{aligned} \tag{3.3}$$

Upon receiving the encrypted `q`, the reader would decrypt and check if the `rndB'` is the correct shifting of `rndB`. If correct, then the smart card possesses the same PICC master key as the reader does. Mutual authentication completes.

The session key is generated from `rndA` and `rndB`. It is composed of first 4 bytes of `rndA`, first 4 bytes of `rndB`, last 4 bytes of `rndA` and last 4 bytes of `rndB`.

$$key = rndA(part1) + rndB(part1) + rndA(part2) + rndB(part2) \tag{3.4}$$

3.4 An Example of DESFire Operation

In this section we give out an example of DESFire EV1 smart card transaction. The transaction is recorded fully from ISO 14443-4 transmission level and is in raw bytes. We document each transaction command with human readable comments. The whole transactions are performed on a blank DESFire card. This example first authenticates the reader and grants authority to create and application. Then the reader selects the application and creates a new backup file. Finally it appends information the newly created file then commits the transaction.

```
Authenticat PICC master key with DES method.
>> 90 5a 00 00 03 00 00 00 00 (SELECT_APPLICATION)
<< 91 00 (OPERATION_OK)
>> 90 0a 00 00 01 00 00 (AUTHENTICATE_DES_2K3DES)
<< 94 9b 00 c0 c4 df 34 12 91 af (ADDITIONAL_FRAME)
>> 90 af 00 00 10 3a 3c 51 6a fe 37 ba ce 6a 9d b1 e1 d1 78 7c 57 00 (MORE)
<< 14 5a 92 49 c9 2d 64 be 91 00 (OPERATION_OK)
```

Figure 3.6: Authenticate PICC master key.

This piece (Figure 3.6) of trace demonstrates the authentication for PICC master key. The choosing method for encryption is DES. The marks of >> is the request

command send out by PCD and commands start with << is the response from PICC.

The PCD starts with the `select application` command which selects the `0x00 0x00 0x00` application which is the PICC card itself. So the authentication master key is PICC master key. Then the PICC responds with `0x91 0x00` to indicate that PCD can proceed. PCD chooses the authentication cipher as DES (on the third line of trace). The PICC does mathematical operation described in equation 3.1 and sends encrypted message to PCD (marked as ADDITIONAL FRAME). The PCD decrypt message and recalculate rndC according to our operation described in equation 3.2 and send them to PICC. Finally the PICC returns information described in equation 3.3 and finish the authentication.

```
Create an application.
>> 90 ca 00 00 05 77 de 50 0b 45 00 (CREATE_APPLICATION)
<< 91 00 (OPERATION_OK)
>> 90 5a 00 00 03 77 de 50 00 (SELECT_APPLICATION)
<< 91 00 (OPERATION_OK)
>> 90 1a 00 00 01 00 00 (AUTHENTICATE_3K3DES)
<< 81 99 9a 6d 70 14 64 a8 5e 98 55 90 1b 71 f8 a5 91 af (ADDITIONAL_FRAME)
>> 90 af 00 00 20 eb 2f 97 b0 d7 eb 78 1d 3a 0d e6 83 79 a2 db cf 8b
   d7 3f cd fe d8 be 61 49 c1 ce 49 b9 ab 4b f9 00 (MORE)
<< 24 6a 9f 3d 90 b7 82 95 a1 1f eb b8 6c dc 49 b1 91 00 (OPERATION_OK)
```

Figure 3.7: Create New Application on PICC.

After successful authentication, PCD now has the power to create applications on PICC. In the trace (Figure 3.7), PCD first create an application with blank 0x00 keys. This key filled with 0x00 can be used for authenticate with 3K3DES method. Then in order to gain power to create files inside application, the PCD again authenticate itself to the application. As we can see the authentication process is just following the same pattern as authentication of PICC master key, the length of APDUs transferred is longer. This is because we used 3K3DES as a chosen cipher, which requires longer block of data.

In trace (Figure 3.8), PCD first selects the already created application then authenticates itself by 3K3DES method with default key (bytes of 0x00). After gaining the power to the application, it creates a backup data file.

The final trace is writing data to backup data file. After the authentication to the application, PCD gets the specific backup data file settings and write data in

```
Create a backup data file under application.
>> 90 5a 00 00 03 77 de 50 00 (SELECT_APPLICATION)
<< 91 00 (OPERATION_OK)
>> 90 1a 00 00 01 00 00 (AUTHENTICATE_3K3DES)
<< 74 f6 85 e1 26 a4 6e 34 e3 29 a8 f3 ec 48 e4 9f 91 af (ADDITIONAL_FRAME)
>> 90 af 00 00 20 09 e9 1e c0 c4 e2 1c dc f8 c8 87 89 20 94 e9 fd 9d ac
   7f f1 0d d3 68 d4 fd 86 a7 05 03 34 07 5c 00 (MORE)
<< a5 3c 81 ea f4 1a 82 48 09 7a 3f 11 f2 f9 4a 5e 91 00 (OPERATION_OK)
>> 90 cb 00 00 07 06 01 20 00 ff 00 00 00 (CREATE_BACKUP_DATA_FILE)
<< 7a e8 f5 8c 47 ba d6 82 91 00 (OPERATION_OK)
```

Figure 3.8: Create a Backup file under New Application on PICC.

```
Write data to backup file, then commit transaction.
>> 90 5a 00 00 03 77 de 50 00 (SELECT_APPLICATION)
<< 91 00 (OPERATION_OK)
>> 90 1a 00 00 01 02 00 (AUTHENTICATE_3K3DES)
<< 48 16 01 6f 8f b1 0f d6 d5 e5 9a 4c 8f cc c5 77 91 af (ADDITIONAL_FRAME)
>> 90 af 00 00 20 5a 54 1b b4 72 13 1d 42 af d2 f4 7f d3 0c 13 bf ff 46
   cf fb 91 36 20 aa 94 a3 01 90 88 ac ec 40 00 (MORE)
<< ca d1 b5 99 4a ee 01 e6 b7 84 a3 f0 8a 14 f9 64 91 00 (OPERATION_OK)
>> 90 f5 00 00 01 06 00 (GET_FILE_SETTINGS)
<< 01 01 20 00 ff 00 00 ce a5 37 0f 93 40 6f aa 91 00 (OPERATION_OK)
>> 90 3d 00 00 17 06 00 00 00 08 00 00 31 32 33 34 35 36 37 38 3a 72 ee
   f2 88 7e 5b eb 00 (WRITE_DATA)
<< 6a ce 6a bf 8e 96 d6 39 91 00 (OPERATION_OK)
>> 90 c7 00 00 00 (COMMIT_TRANSACTION)
<< 28 25 4d 84 89 1d 8e c7 91 00 (OPERATION_OK)
```

Figure 3.9: Write data to Backup Data File.

bytes to the file. After writing is complete, it commits the change and finish the transaction.

Chapter 4
Relay Attack on NFC applications

The *partial* goal of this thesis is to deliver a successful NFC relay attack system via off-stock market phones. The final goal is to study the relay attack by smart phones and propose methods to detect and prevent it. We use off-shelf Android smart phones as hardware and build applications on them. We install modified open source operating system Cyanogenmod 9[1] on them. We also using Java reflection technology to expose related NFC emulation APIs from the operating system. Within the powerful handset, we are able to emulate a practical tag at the same time communicate the data we received from real reader back to the smart phone which is operating in read/write mode, where the tag is actually read/written according the transaction.

This chapter consists of information and perquisites of derive a practical attack on NFC applications. We first discuss about the basic of general relay attacks, and then talk a little about the requirement of NFC relay attack. We explore in-depth about how far we can achieve via the existing operating system and hardware, and give out an overall design of attacking system.

4.1 Man-in-the-middle attack (MITM) and Relay attack

The very beginning of implementing this practical attacking system is to understand the fundamental concept of relay attack and its similarity to Man-in-the-middle (MITM) attack.

Man-in-the-middle attack, is often refer to a kind of attack that that a third party is interfering with the normal traffic between two parties to gain information about the communication. The attacker hijacks the communication channel and impersonates as the normal users. The entire communication is interrupted and legitimate users at end points still believe they are talking to each other without a

[1] http://www.cyanogenmod.org/

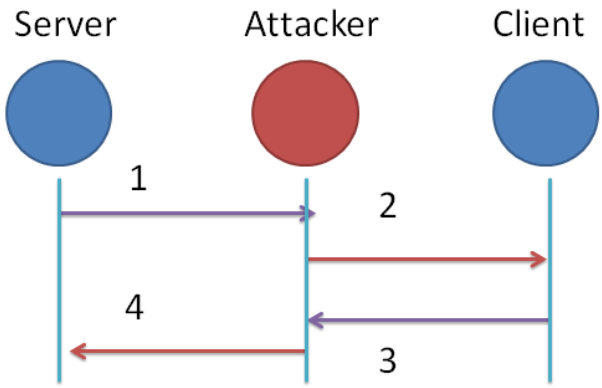

Figure 4.1: Man-in-the-middle attack workflow.

third party in the middle. The attacker must be able to achieve the following points to make a successful MITM attack:

- `Interrupt the whole communication between two legal parties.` This is achieved via hijacking communication, e.g. pretending to be the access point of Wi-Fi to hijack all internet traffic[ACL07]. This is not easy to achieve in telecommunications.
- `Impersonate as legitimate users in the conversation.` For simplicity, if there is a server-client mode service running, MITM attacker shall be able to impersonate both as client and server.
- `Reproduce exchanging information.` This step is the most difficult part. Usually modern communication protocol(such as SSL/TLS) is encrypted and needs authentication steps (to both end-users) to generate session key to encrypt the channel. Or the session key is generated based on a shared secret for both parties(such as NFC smart card[NXP12]). Without the session key, an attacker can do nothing with the actual content in fly. Nonetheless he inserts new, valid traffic in to communication, e.g. An attacker shall be able to have the symmetric encryption key to encrypt and decrypt traffic. Otherwise he becomes a dummy pipe during the attack.

As we can see from the illustration 4.1, the attacker will receive communication from server at step one and try to decrypt it and then encrypt it again to send out at step two. The same thing would happen if attacker receives information from

client on step three. However, if the attack fails to obtain the session key for original message encryption, he cannot fully understand traffic and reproduce correct message. In our very specific scenario NFC applications, unfortunately the communication can be encrypted by DES/AES/3DES. Some communications can be plain text or MACed messages thus attacker still can sniff them.

Here comes the relay attack, a close variant of MITM attack. In a "reader - attacker - card" system, an attacker can forward information without recalculate it. It is just like a router, forwarding IP packages without understanding the payload of it. But the router still can dump the traffic and study the traffic time pattern. Such kind of attacks is easy to implement as long as we can emulate the parties in the communication. Fortunately in our NFC cases, although the traffic has an option to be encrypted, it still lacks of a detecting system to find out the existence of possible relay attackers in the middle. As operating in a close distance as a design and pattern, NFC is totally vulnerable to relay attacks. The figure 4.2 is an example of design a two-phone solution to derive such kind of relay attack.

Figure 4.2: Design of Relay Attack against NFC tags.

Here a cell phone is used as a fake card (AsTag) to communicate with the real reader, using card emulation technology to emulate a smart card. Another cell phone is acting as a fake reader (AsReader) to communicate with the real card. The two cell phones are linked with a wireless channel (Wi-Fi or Bluetooth). Once the real reader initiates transactions, all the request messages are forwarded through wireless channel from AsTag to AsReader. AsReader conducts real transaction with smart card using the request messages. The transaction responses from real card are transmitting back to the real reader via the same channel. This relay attack makes the relay attacker transparent to both NFC reader and real card.

4.2 Requirements of Smart Phone NFC Relay Attack

This section we discuss the basic requirements if we want to make a NFC relay attack using off-shelf Android smart phones.

4.2.1 Phone-to-Phone Communication Channel

As we have discussed above in **chapter 1**, NFC relay attacks have to solve the problem of distance in the first place. Usually the card is read and written at a service point such as POS machine. If you have a wired communication cable between your attacking devices, it not only limited your attacking range but also make yourself more suspicious in public.

Fortunately, the modern Android smart phones are equipped with enough communication ability such as Wi-Fi or Bluetooth. This serves as a best invisible proxy channel. NFC transactions limit the length of payload by 253 bytes so the total on-fly traffic would be less than 300 bytes[NXP12]. Regardless of means of transmission, this payload will be quickly transmitted over Wi-Fi/Bluetooth. The design of communication results in three styles of link between our attacking devices:

- *Wi-Fi directly link.* In this mode one cell phone is acting as a router and the other as a regular Wi-Fi client. Information is passed without going through a third router.
- *Bluetooth directly link.* In this mode two cell phones are connected via Bluetooth in pairs. Information is also passed without a third party.
- *Mobile Broad Band links.* In this mode two cell phones will both sends information to a remote server on the internet acting as an exchange server. Communication is indirect but has the most flexibility. Also this channel is provided by a reliable telecommunication company.

The obvious advantage of first two methods is they are controllable in terms of time delay. Communication time is highly dependent on the Wi-Fi and Bluetooth technology itself and operating system stacks on each phone. However, the shortcoming is the devices must be configured and paired before they can be used. Also the distance of attack is limited by the max Wi-Fi or Bluetooth transmission range between two devices.

The third option is less reliable. As we rely on a third party telecommunication service, delay time is uncertain. Also it requires a reflection server operating constantly on the remote, this will increase the delay in the communication as described in figure 4.3.

4.2.2 NFC Read/Write, Emulation Mode

As we have mentioned in **chapter 1** and **chapter 3**, nowadays a smart phone is as a powerful personal computing base.Popular smart phones are equipped with

Figure 4.3: NFC Relay Attack via Mobile Broad Band.

NFC chips and the APIs for those smart phone platform enable the programmer to easily implement their logic. So these smart phones become more suitable for conducting NFC relay attack. Take Android smart phones as an example, although different vendors has different models, most of them have NFC chips and a unified API interface exposed by Android operating system. Read/Write of NFC bus cards is no longer bound to specific top-up machine as a user can easily check his bus card balance on special designed app by tapping his card on the back of Android smart phone. But how to emulate a card is not introduced in official Android version until recently. Recently on Google Nexus 5 phone released an Android version 4.4 which can enable programmers to write applications which emulates a NFC card (Host Card Emulation).

As the figure 4.4 shows the design of an Android native application to use host card emulation (HCE) feature is highly controlled by Google Android operating system. It only exposes the interface of ISO 7816 application level APIs. Right now among official releases only KitKat 4.4 operating system has the feature and during the time this thesis is writing, Android devices with 4.4 version of operating system update just emerged for several months. Many old devices cannot receive offical KitKat 4.4 update from Google. However this HCE idea has become popular already in the modified Android operating system for several years. According to Cyanogenmod 9 operating system (a modified Android) documentation, it supports ISO-DEP specification (based on ISO 14443-4) and upper layer. Especially, the operating system supports the NFC-A (ISO 14443-3 type A) protocol, which the DESFire EV1 uses**chapter 3**.

When the HCE device is approaching a reader , Cyanogenmod operating system will generates an intent and invoke the NFC app designed by us. This app emulates the interface as a DESFire EV1 card. Requests from reader are sent directly from Android operating system to NFC app. These requests are then transmitted to another smart phone which is interacting with a real card. While functionalities are described simply in design, how to let it cooperate with the communication channel is difficult during implementation. We have to carefully optimize the application structure to accelerate app logic speed and achieve low latency of our relay attack. The trasmission of NFC messages to Wi-Fi/Bluetooth channel shall be efficient and error tolerated. We also have to make the program code separate and components de-coupled so maintenance of code is easier.

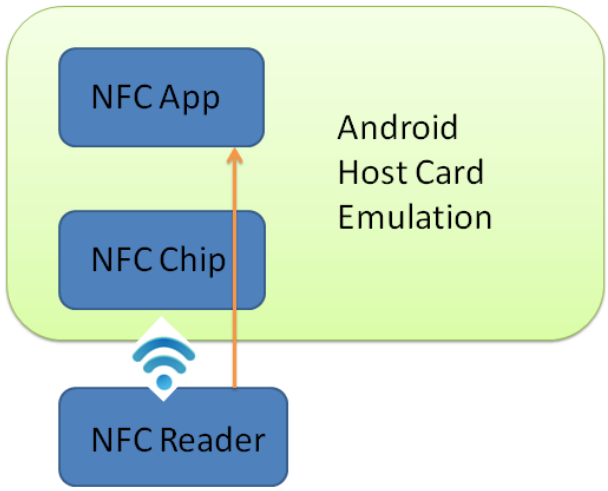

Figure 4.4: Host Card Emulation of NFC on Android.

4.2.3 Block Communication between Reader and Card

The nature of NFC is a contactless transmission protocol in short range only by 1-10cm, we can achieve blocking communication by attaching attacker's AsReader phone directly on the NFC card to avoid other devices reading it. Furthermore, put the NFC card away from reader and attach another AsCard phone to its reading surface. By the design of Android operating system, only one intent is triggered if we are doing NFC transaction. We design our application to be triggered and keeps foreground running during the attack. This forces the communication between the reader and card to go through our attack devices only.

4.3 Solution Design Components

In this section we will cover the general choices of components to build up feasible relay attack software architecture on AsReader phone and AsCard phone. These softare components form Android apps to run on the phones. The detailed implementation codes are covered in Chapter 5. Here we introduce the components on high level and justify our choices.

- *Wi-Fi and Bluetooth direct links.* As we have discussed, using a third router or mobile broad band would significantly increase the potential delay time in transmitting the data.

- *Cyanogenmod version 9* operating systems instead of Android 4.4 KitKat. For the art right now, Android KitKat 4.4 is only available for newest devices (like Nexus 5). Cyanogenmod 9 has a better compatibility with our old experimenting devices at hand — Nexus S and it has been tested. Furthermore, Cyanogenmod 9 enables us to manipulate on ISO 14443-4 level and we can observe raw APDUs from there. Android 4.4 KitKat only allows programmer to visit ISO 7816 level, which is a level higher than ISO 14443-4.

- *Control communication from ISO 14443-4 transmission level.* Leave the lower level to the operating system to deal with. ISO 14443-3 anti-collision is performed by under-lying operating system. Our app program logic starts to forward actual APDUs on the ISO 14443-4 level.

- *Device Role Divide.* One smart phone is called AsCard, it is in card emulation mode and expose its interface as Mifare DESFire EV1 to the real reader. Another smart phone is called AsReader, it is in read/write mode and gets transaction instructions from the AsCard phone over Wi-Fi or Bluetooth. AsReader phone receives the requests and deals real transactions with a DESFire EV1 Card. No third device is required.

The above first point and last point is easy to understand in our scenario. Only two phones are involved and we choose a wireless channel of Wi-fi/Bluetooth to link them up. But the second and third are related to physical constrains we have. Cyanogemod version 9 operating system is a moddification of Android open source project. It already introduced API level host card emulation although quite primitive, limited only to ISO 14443-4 level (no above layer ISO 7816 support). The development progress is also painful. It also requires the programmer to use Java reflection technique to wrap around the corresponding APIs(these APIs are not part of the Android Development SDK release). However, the raw APIs on ISO 14443-4 level gives us great chance to truly look at the content of original NFC

requests/responses in bytes. Also Cyanogenmod 9 is a tested operating system version which is compatible with our hardware existing in laboratory — Nexus S smart phones.

In the third decision we decided to forward NFC commands on ISO 14443-4 level. It is because the lower level ISO 14443-3 anti-collision is not exposed to client application in Android even in modified Android. And since lower level ISO 14443-3 level provides anti-collision and activation at the beginning of the conversation phase, we have no choice but rely on underlying operating system to handle it then take the control over once it is done. This can cause a difference of parameters in **chapter 3, figure 3.2**. From ISO 14443-4 level, our app logic begins the monitoring/forwarding on real NFC transaction commands.

Chapter 5
Implementation

This chapter covers the procedure how to build up a useful prototype that enables the relay attack on NFC smart cards. This prototype demonstrates the general design we discussed in section 4.3. Also some details of programming code are covered in this chapter.

The whole prototype consists of three major parts. One Java application is in charge of performing basic transactions including reading, writing files on card and erasing and reset the card. This program exists on a regular PC; the libnfc library enables it to control a dedicated NFC reader to perform those actions. The other two parts are two different Android programs runs on two separate smart phones. They share the main software architecture but with different functionality. One smart phone operates in host card emulation mode, we call it `AsCard`. The other one operates in regular read/write mode just like a regular NFC reader. We call this one `AsReader`. Together AsCard and AsReader are connected via Bluetooth or Wi-Fi channel to achieve internal communications.

5.1 Development Tools and Dependencies

For the basic developer machine, a Ubuntu 12.04 LTS is with Java 6 is required (Used to develop NFC apps on Nexus S smart phones). There should also be a PC running on Java 6 with a specialized reader for NFC attached to it (Used to develop DESFire EV1 smart card read/write application). This PC shall also install a package of libnfc Java library. In addition, there would be two already rooted Android smart phones with NFC features enabled. These two phones are installed with Cyanogenmod 9 operating system. These requirements are described in table 5.1

Another important tool we use here is the ACR122u NFC reader. This is the standard reader to support Mifare DESFire EV1 smart card. Within the libnfc library enabled inside server, the Java program on the server can fully control the

Machine	OS	Library	IDE
Dev Machine	Windows 7	Java 6, Android SDK	Eclipse
Server Machine	Ubuntu 12.04	Java6, libnfc	Eclipse

Table 5.1: Developer Machine Settings

reader. Libnfc library is an open source libray aiming at operating existing NFC devices. It supports many languages including Java[1]. ACR122u reader operates in 13.56MHz frequency and is in compliance with ISO 14443 NFC protocol standards. It both supports ISO 14443 type A and B cards. But for our experiment DESFire EV1 cards, only type A is needed.

In order to make the relay attack, we need two off-shelf smart phones which can run Android system. Google Nexus S is used. Although it is not the newest one in the market, it features enough battery power to do the experiment and is available in laboratory. See table 5.2 for supported features about this phone.We will have to both re-install them into Cynogenmod 9 operating system. The modified system exposes NFC host card emulation feature which cannot be accessed in regular Android operating system. See **Chapter 4**. Although we are using default Android SDK (Which means we cannot call Host Card Emulation APIs in Cyanogenmod directly), But we can walk around it by Java Reflection technique.

Model	NFC	Bluetooth/WIFI	OS
Nexus S	NXP PN544	Supported	Cyanogenmod9

Table 5.2: Nexus S phone with Cyanogenmod 9

5.2 Architecture Implementation

Before we get into the detail design of this prototype, we would like to remind our readers about the final goal we want to achieve in this picture 5.1.

As we can see in our laboratory environment, we have a machine which runs a simple read/write ticketing NFC system. It connects to the ACR122u reader for providing NFC operations. Each time the transaction is either triggered manually or by Java program runs on it. The attacker is responsible for placing the relay attack phones between real DESFire EV1 card and real reader. The smart phone attached to the real card is called `AsReader` as we have mentioned earlier. The other one which operates in card emulation mode is impersonating as a fake card, thus, we call

[1] http://nfc-tools.org/

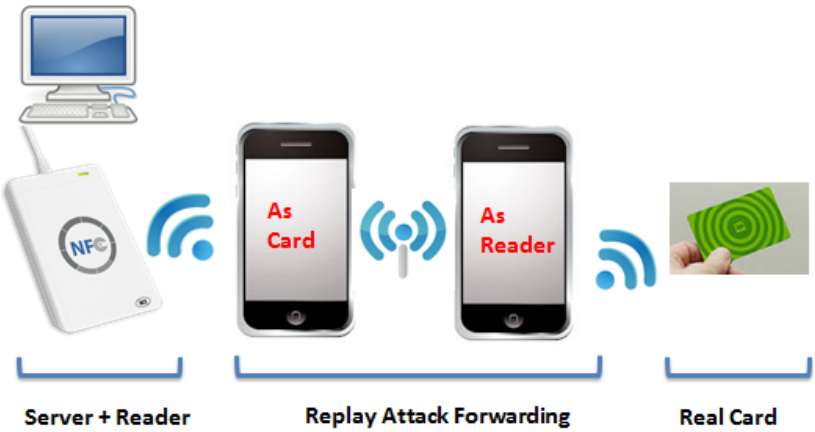

Figure 5.1: Prototype Overview of Relay Attack.

it `AsCard`. To make a connecting between the two cell phones, We use Bluetooth 2.1 or Wi-Fi (802.11b).

Section 5.2.1 discusses about the setup of Java Read/write program. Section 5.2.2 is the design of host card emulation phone program including its optimized architecture. Section 5.2.3 is the implementation details of reader/writer phone program.

5.2.1 Ticket Reader Application

The NFC read/write component is rather simple in logic than the smart phone apps. It has a main purpose to control the ACR122u reader which has been attached to it via USB port and performs NFC transactions. The stack of operating system and the connection between our programs to the rest components are illustrated in Figure 5.2.

Daniel[And13] in his thesis has wrapped up the libnfc library and provides us a convenient API interface to make NFC transactions on DESFire EV1 card, our design of Java program logic is much simpler. As the primary goal is to successfully operate the information of the card and make NFC transactions, we here gives one example how to authenticate and format the Mifare DESFire EV1 card.

The Java Code snippet for authentication and then format the card is as following Figure 5.3:

Here we assume the card primary key is a 8 byte of 0x00 array and the au-

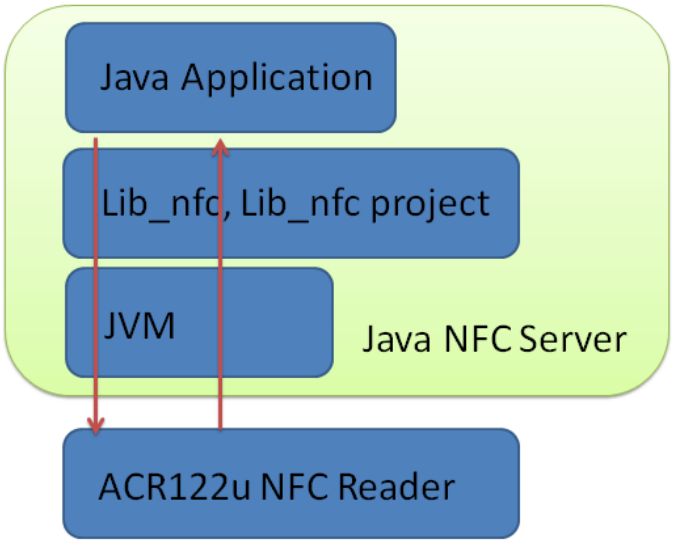

Figure 5.2: Java NFC program structure (on PC).

thentication method is DES encryption. We first wait for a DESFire EV1 card and then connect to the card. We select the default card application (which is the authentication of whole card key), and then perform actual authentication. After we have done the authentication, we issue the format card command immediately, then disconnect.

On the ISO 14443-4 type A transmission layer, the actual APDU bytes that are transferred are looks like below code block Figure 5.4:

Here are some important bytes which lead to our attention. Starting the third line from `"0x90 0x0a"`, the reader application tries to authenticate to card. Any line starts with symbol `"»"` is the outgoing APDUs from NFC reader. The line starts with symbol `"«"` is the APDU that DESFire EV1 card answers.

As our audience can see, with the three lines below, the card and reader exchange information to authenticate the reader and the card as described in section 3.3.2to make sure the reader has a valid PICC master key to the card. This encryption is non-repeatable because its process involves random numbers (see section 3.3.2). After the mutual authentication is done, `"0x90 0xfc"` command line is issued for the sake for wiping the card entirely. If during the process the authentication is failed, the whole transaction would be terminated.

```
package nfcjlib.extra;

import java.io.UnsupportedEncodingException;

public class ExampleFormatCard {
    public static void main(String[] args) throws UnsupportedEncoding
        DESFireEV1 desfire = new DESFireEV1();
        desfire.connect();
        // clear card
        desfire.selectApplication(new byte[] {0x00, 0x00, 0x00});
        desfire.authenticate(new byte[8], (byte) 0x00, KeyType.DES);
        desfire.formatPICC();

        desfire.disconnect();
    }
}
```

Figure 5.3: Format Card, Java code based on libnfc.

```
>> 90 5a 00 00 03 00 00 00 00 (SELECT_APPLICATION)
<< 91 00 (OPERATION_OK)
>> 90 0a 00 00 01 00 00 (AUTHENTICATE_DES)
<< 10 d1 17 ce 14 47 e6 29 91 af (ADDITIONAL_FRAME)
>> 90 af 00 00 10 98 19 f5 cf 99 16 bc b9 4b 5e 3e b3 b5 2b 6e 7f 00 (MORE)
<< e9 0d 77 eb 47 aa 33 52 91 00 (OPERATION_OK)
>> 90 fc 00 00 00 (FORMAT_PICC)
<< 63 00 (OPERATION_OK)
```

Figure 5.4: Format Card, Actual APDU been Exchanged

We have more examples in the Chapter 6. These examples are written as part of the timer framework which we can measure the round trip time in different situations. The timer framework means we insert Java code before and after NFC operations like read/write to detect round trip time for each command to be executed. To summarize, we measured the time for mutual authentication with DES, 3K3DES, AES are measured as well as reading and writing different length of data. The reading and writing process is covered with scenario which includes plain text, MACed messages or fully encrypted messages. We carefully measured the delay we introduced to the system when we implant our relay attack devices. Please move on to Chapter 6 for more details.

5.2.2 Host Card Emulation App

As our audience may wonder, within a standard Android SDK(which lacks of functionality of Host Card Emulation feature provided by Cyanogenmod 9), how can we expose these Host Card Emulation feature APIs?

The answer is *Java Reflection*. By using Java reflection, we can explore and wrap around the functionality which is only accessible at run time. Thus, we here first create a *TagWrapper* Class which looks exactly like regular NFC class interface. The only difference here is we are no longer reading or writing to the NFC cards, instead we are doing it to the NFC readers.

As we see from the class diagram Figure 5.5, TagWrapper class gives out same interface as *connect(), reconnect(), transceive(byte[])*. This interface has the same syntax as the regular NFC operations to DESFire EV1 Cards. The only difference here is they are targeted at PCD reader devices rather than NFC cards. The function *transceiver(byte[])* will send out byte[] array to reader and get the response as return. Just like we send out byte[] array to NFC cards and expect responses.

The main architecture of the Host Card Emulation (HCE) app is built using a loosely coupled design. As we are dealing with at least two communications: NFC communication from NFC reader and Bluetooth/Wi-Fi communication from the other phone, making a single thread application becomes impossible. Nonetheless we should consider another thread which is default in Android, which is the UI thread.

As the HCE app gets the environment and then enabled NFC and Bluetooth/ Wi-Fi feature, it now waits for the NFC reader to approach for further transactions. But what if the paring phone on the other side of communication does not prepared to receive APDUs? What if the other phone is already dead? How to manage the lifecycle of both Android UI and NFC lifecycle? How to create efficiency during the transmitting and exchanging data?

These challenges are address by the following message system design in figure 5.6. The Android system launches us a default UI thread which is long live. We use it as a hub to spawn more threads. Namely NFC thread and Bluetooth/Wi-Fi thread.

NFC thread is in charge of listening to any NFC events triggered by NFC chips on the phone. Once there is an APDU comes in, it will quickly forward the information to UI thread, where the APDU is logged. NFC thread is also responsible for reporting the status of NFC events. Such events like reader are closed by force and reader is reconnected. Once these events are received, the corresponding flag in UI thread will be set, so UI thread can tell the other phone about the emergency situation via Bluetooth/Wi-Fi.

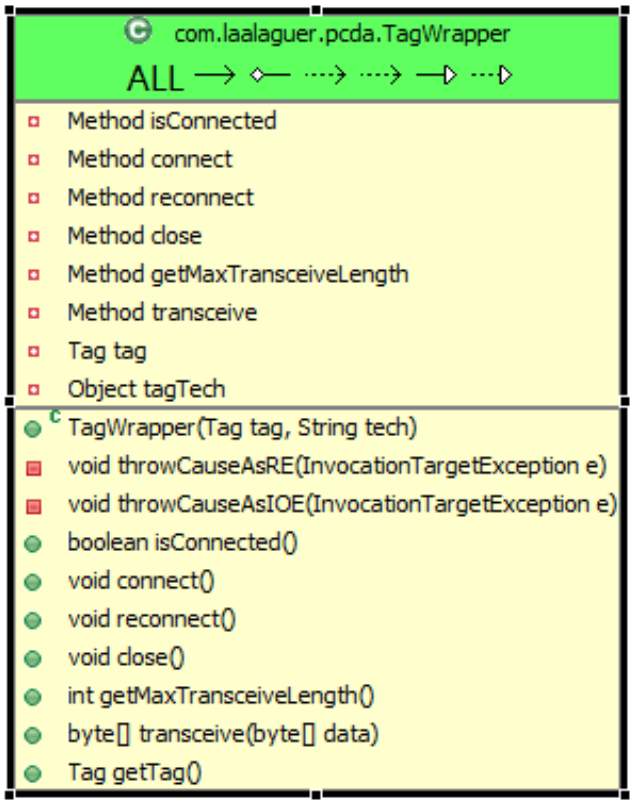

Figure 5.5: Tag Wrapper to enable Host Card Emulation

Bluetooth/Wi-Fi thread is another thread which similarly in charge of Bluetooth or Wi-Fi messages. This long live thread can monitor the traffic comes from the other phone thus servers as a tunnel between AsCard and AsReader phones. Also the system events like Bluetooth/Wi-Fi connection failure would be reported by this thread to UI thread. So the UI thread would know how to make corresponding activity fall down naturally.

As on the Bluetooth and Wi-Fi channel, we choose to implement a UDP socket. This is because UDP is easier to manage than TCP and much faster because there is no mechanism for retransmission. Unreliable it is, but it servers well in our close range communication.

The figure of 5.6 best demonstrates when the connections are all established, how

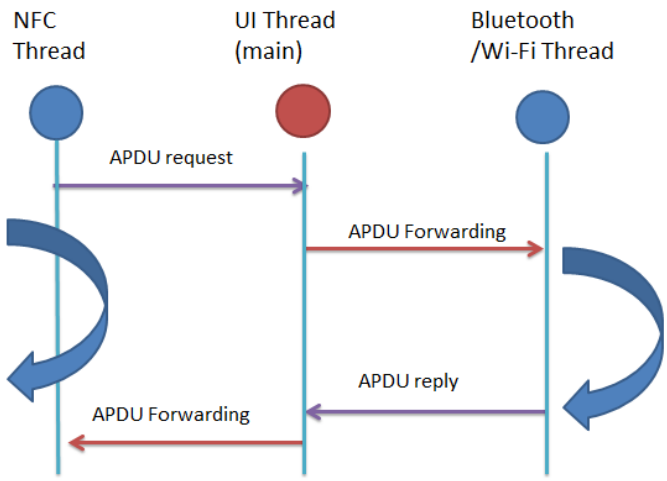

Figure 5.6: Three Threads Architecture of HCE app

is an APDU request from NFC reader been captured and forward to Bluetooth/Wi-Fi channel to deliver. Once the APDU response is prepared and sent back from the other phone, we can again monitor the content of APDU and forward it back to the NFC reader. The whole process is decoupled by using threads and asynchronies. Even if either NFC channel or Bluetooth/Wi-Fi device faces a failure, the system can handle it gracefully.

Some important class diagrams is included in figure 5.7 for interested readers. The AsCard Activity is the main UI thread created when launching the application. *BluetoothMessageService* is a class which in charge of Bluetooth connection. It will spawn corresponding threads used for listening and accepting sockets. *HCEDesfire-Service* is a class in charge of NFC module on the phone. It would spawn a thread *DummyReplyThread* to handle the APDUs to and from NFC reader. This thread will periodically empty and send the APDUs in the buffer to feed the NFC reader.

As this thesis is a research paper targeting at solving problems rather than discussing the solution method itself, we would not dive into the code design details. Documentation would be found in the original source code if reader is interested.

We display here is a figure 5.8 shows the important internal state of our HCE application, to summarize, the rows begin with BT is related to Bluetooth reading/writing. Once the flag is set, the program would know there are messages coming from or needs to be sent out via Bluetooth channel. The rows begin with HCE is

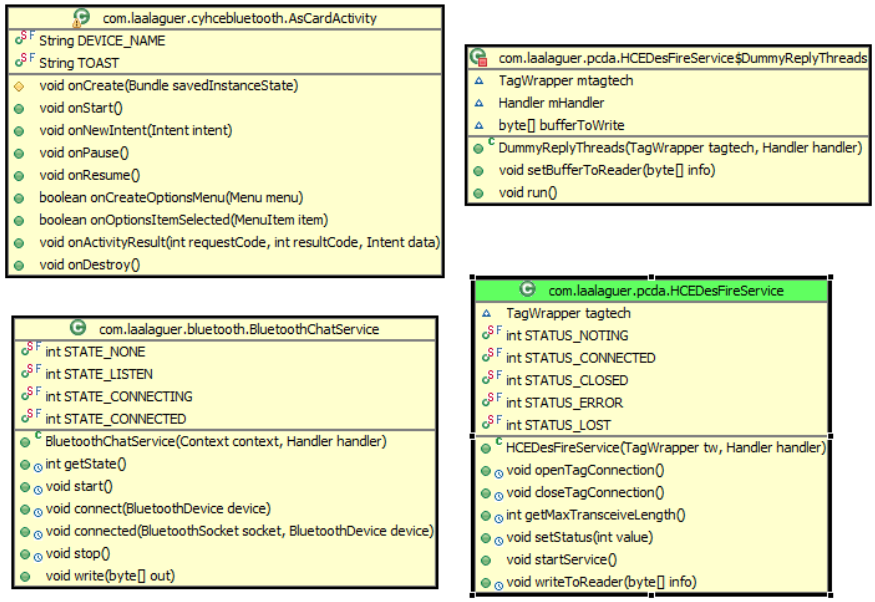

Figure 5.7: Important Class Diagram of HCE application

related to NFC host card emulation. Once the flag is set, we know that there would be NFC messages coming or the NFC reader has changed its status.

5.2.3 NFC Reader/Writer App

As we have gone through the design principles and architecture of Host Card Emulation app we built above in section 5.2.2, it is not difficult to image a similar structure app can be built to read/write real Mifare DESFire EV1 cards. It also has a similar life cycle and communication ability except that it comes along with a real NFC tag reading/writing interface targeting at real smart cards.

We here omit the description of all the same structure but keeps the most essential part of the code, the NFC read/write interface. As show in figure 5.9. It has public methods of *openTagConnection(), writeToCard(byte[])* to manipulate the APDUs transfer to and from card. It also has a *DummyReplyThread()* which is spawn by the UI thread to monitor the NFC card status. This thread has extremely similar functionalities with the same named thread in Host Card Emulation app.

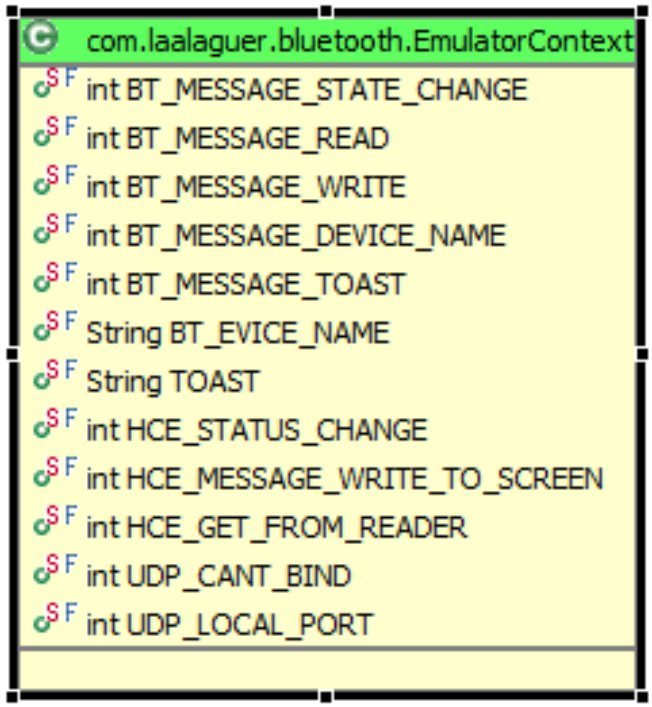

Figure 5.8: Internal Status of HCE application

5.3 Flow of Relay Attack

As we have all the components in hand, we start to pair them and derive a practical attack on the NFC smart card. The static set up would be show in figure 5.10.

Attack shall do the following things to successfully launch up a relay attack.

- *Launch the apps on both AsCard and AsReader smart phone.* Click on the already installed app icon to launch the app.
- *Enable the NFC feature and Wi-Fi or Bluetooth.* Remember to enable NFC and communication radio interface, otherwise the feature is not complete.
- *Connect two phones with Bluetooth or Wi-Fi.* Pairing your devices so they can talk to each other.

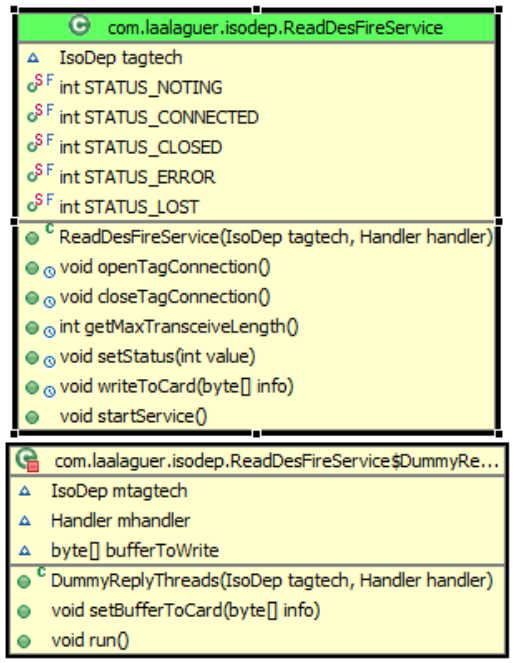

Figure 5.9: Tag Connection Class Diagram of Reader/Writer app

- *Attach the AsReader smart phone to the victim smart card.* Simply touch the card with smart phone. It will be captured by our app.
- *Attach the AsCard smart phone to the NFC reader.* Similar to the step above, but the target is the NFC reader we want to play with.
- *The Relay attack* automatically triggered when the reader start to make transactions.
- *Attacker hijacks the communication and can drop or analyze any APDUs on the fly.* Attacker can now instantly see the APDUs on the UI of each app. He can further saving them for analysis.

5.4 Problems and Optimization

During the implementation of prototype and the iteration of experiment-test-modification loop, we found out many difficulties lying in our path. Most of the problems related

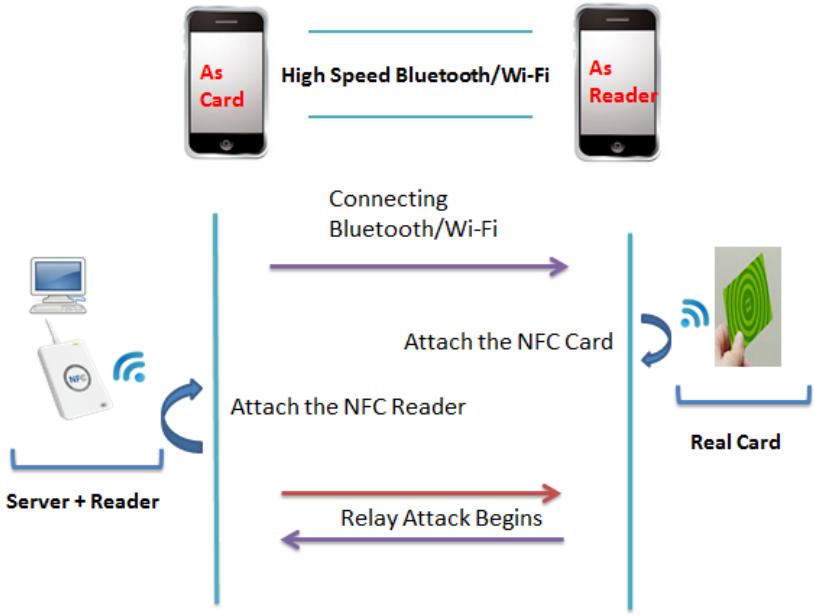

Figure 5.10: Tag Connection Class Diagram of Reader/Writer app

to software are solved and optimized and integrated to our app. But some problems lies in the hardware and driver itself.

Efficiency of relay attack and message buffer size. This problem is a obvious question. We try to decouple all the components and use thread to keep the application runs as fluently as possible. The Wi-Fi/Bluetooth service is not interfer with NFC transaction service. These two services are run in sepearate threads to make the app smooth. If we do not use threads, Wi-Fi/Bluetooth service may wait for transmission of data and delay the receiving of NFC requests or delivering of NFC responses. As we have the knowledge of Mifare DesFire EV1, the max transmission byte size is 253. Thus we have programmed a 300 bytes buffer for information exchange. We do not want to waste a longer buffersize to be transmitted via communication channel. As this buffer can be chopped into pieces on before Wi-Fi transmission and re-assembled on the other side. This will reduce the transmission speed.

Lost connection after Formatting card. This is a small flaw in our implementation but amended. When to Java server program tries to format the card, the card would do it then disconnect. Our phone was unaware of this automatic feature and still hang

there waiting for response. But this flaw has been fixed during the implementation.

Different UID of real Card and Emulated Card. According to our observation, the initial phase of wake-up and anti-collision is handled by Cyanogenmod operating system in a different way than real card. The protocol ISO 14443-3 describes the anti-collision phase shall detect where the UID of card is 7 bytes or 4 bytes. As we have known the Cyanogenmod can only controls the NFC chip to emulate a 4-byte UID. This feature can be best described in the figure 5.11. Far from now, it is fixed in the operating system driver and NXP PN544 chip. So we cannot overcome this.

```
Random DesFire Card
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 03  44
       UID (NFCID1): 04  6f  0f  c2  80  26  80
      SAK (SEL_RES): 20
                ATS: 75  77  81  02  80

Nexus S
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 00  04
       UID (NFCID3): 08  c3  21  4e
      SAK (SEL_RES): 60
                ATS: 78  33  88  00
```

Figure 5.11: UID of Emulated Card vs. Real Smart Card

Chapter 6
Evaluation and Experiments

As we have successfully implemented our relay attack on off-stock market smart phones, it is time to measure the impact of relay attack on original communication. This chapter describes the experiments we have conducted in order to discover the difference, errors we have introduced into the NFC communication model. Also the performance of our relay attack is also measured.

As we have explicitly stated in section 5.4, Emulated NFC tags(by phone) has a different UID and behavior in the ISO 14443-3 protocol level. Thus our emulated tag has a slightly different anti-collision phase than the common Mifare DESFire card. But the real world transaction activity is actually carried out after anti-collision phase and into ISO 14443-4 transmission protocol. On this level, our relay message is exactly the same to the original message. For this reason, the most experiments here we covered is focusing on the ISO 14443-4 level. The main metrics we focus on is delay time introduced by relay attack comparing to the original smart card.

This chapter will first introduce our experiment set up and talk about some error sources in delay time measurement. Tools and equipment we used to track the status of whole system. Next we will carefully run experiments over most possible DESFire commands and transactions to test the delay we introduced when plug in our relay attack. Finally would be a discussion on the data we collected.

6.1 Experimental Setup

Experiments are carried out in a scientific manner and we have to lay out a foundation for them. This section describes some basic set ups and also the error resources if we have such set up.

6.1.1 Tools and Environment

During the experiments, we choose Nexus S as our smart phone devices and both phones are installed with our attacker app. During experiments, the phones are placed one meter to each other on the desk.

Name	OS	Library	Reader
NFC Server	Windows 7	Java 6 JVM	ACR122u reader

Table 6.1: Java NFC Server Setting

Name	Connectivity	Distance	Feature
Nexus S	Bluetooth 2.1/Wi-Fi 802.11b	1 meter	NFC

Table 6.2: Nexus S phone settings

Name	Frequency	Functions	Protocol
Mifare DESFire Card	13.56Mhz	read/write, authenticate	ISO 14443 A

Table 6.3: DESFire EV1 features

Also we have a device Proxmark3 to sniffing the traffic and have a more specific round trip time measurement. Proxmark3 is developed as an open source hardware project to clone, read and sniffing the traffic of RFID tags. Our Mifare DESFire cards are acutally supported by Proxmark3. With the high frequency antenna connected, it can continuously sniff the traffic and mark down the time interval of APDU commands sending from reader and from card.

Another time measurement method is to measure the round trip time from Java reader program. The time measurement is subject to the actual accuracy of JVM implemented. However as the actual NFC smart card services are implemented with business logic inside various NFC applications on server, its time measurement also can be representative for our study.

6.1.2 Error Sources in Experiment

As we highly depend on the Proxmark3 to give out delay time information if we insert relay attack between the reader and DESFire EV1 card, the first errors source is the Proxmark3 hardware itself. It may record wrong messages or wrong times by accident. By the time now, the only method we have is to upgrade to newest stable version and do as much experiments as we can to eliminate unstable results.

The second error source is when we calculate the delay from Java NFC server application; we usually omit the accuracy of JVM time measurement. Also the ACR122u reader itself is a black box to experiments. Delay time measurement may also vary from time to time because operating system maybe busy running other processes. Thus the time measurement results from Java NFC server application can only be an indicator, rather than a serious data source.

6.2 Time Delay Measurements

6.2.1 Single Command Encryption Time of Real Card

```
>> 90 5a 00 00 03 00 00 00 00 (SELECT_APPLICATION)
<< 91 00 (OPERATION_OK)
>> 90 0a 00 00 01 00 00 (AUTHENTICATE_DES)
<< 10 d1 17 ce 14 47 e6 29 91 af (ADDITIONAL_FRAME)
>> 90 af 00 00 10 98 19 f5 cf 99 16 bc b9 4b 5e 3e b3 b5 2b 6e 7f 00 (MORE)
<< e9 0d 77 eb 47 aa 33 52 91 00 (OPERATION_OK)
>> 90 fc 00 00 00 (FORMAT_PICC)
<< 63 00 (OPERATION_OK)
```

Figure 6.1: Single Command authentication by DES example

Here we come back to the problem of exploring the processing time of encryption of messages on the DESFire card. Namely we have three choices, DES, 3K3DES (a variation of triple DES) and more advanced AES. For the DES and 3K3DES, each round only 8 bytes of key is used. But for AES, key length would be 16 bytes.

The best place to explore the encryption progress is showed again in figure 6.1. As we have discussed before, DESFire card has a mutual authentication progress which evolves encryption and decryption. Take DES for example. The trace lines begin with "0x10 0xd1" and "0xe9 0x0d" in figure 6.1 are actually card response APDUs. These responses are the result of DES encryption/decryption of 8 byte temporary secrets. As the processes of responses in these two steps are different, we call them step 1 and step 2 respectfully.

Table 6.4 best describes the different time we observed for the step 1 and step 2 phase of authentication. DES uses a 16-byte key but only 8 bytes are effective(as we mark as *), 3K3DES would use a 24-byte key as it needs three rounds. AES only needs 16-byte key but provides the same strong mechanism as 3K3DES.

As our audience may observe from the table, our testing results show that regardless of 8-byte or 16-byte encryption, all the three encryption mechanism can be finished in 2-3 ms. From the average time, DES is always fastest, following by AES. The slowest option is 3K3DES. But the advantage is really not so huge enough to catch up our attention.

Name	**DES**	**3K3DES**	**AES**
Key Size(byte)	16(*)	24	16
Step 1 (Avg) ms	2.34	2.76	2.66
Standard Deviation	0.04	0.05	0.03
Step 2 (Avg) ms	2.29	3.12	2.73
Standard Deviation	0.05	0.03	0.04

Table 6.4: Response Time for Different Encryption method on Real Card

6.2.2 Single Read Operation Time of Real Card

After we have known the time of processing encryption on 8 to 16 bytes of small data, we start to look at some more advanced operation or say, transaction on DESFire card. We hereby choose a standard backup file on DESFire card as our testing object. We have tried to read different length of data from it with different methods.

We choose to read from 8 bytes to 253 bytes (top limit of transceiver payload length) from a single backup file. Only backup file allows us to freely manage our own content structure. This reading process is either plain text, or MACed, or fully encrypted. There are also choices for encryption methods, like DES/3K3DES/AES are applied accordingly. Table 6.5 is the matrix we collected after we have carefully measured the response of DESFire card.

Size/Time(ms)	**DES**	**DES MAC**	**AES**	**AES MAC**	**3K3DES**	**3K3DES MAC**
8 byte	0.51	0.96	1.25	1.25	1.37	1.37
16 byte	0.54	1.22	1.63	1.58	1.51	1.64
32 byte	0.59	1.79	1.93	1.93	2.29	2.22
64 byte *	0.85	3.11	2.81	2.78	3.48	3.48
128 byte **	1.21	5.48	4.42	4.42	5.92	5.88
253 byte ***	1.91	10.37	7.51	7.49	10.41	10.38

Table 6.5: Response Time for Reading Data, Different Encryption methods on Real Card

These data are also illustrated in the figure 6.2 . To our surprise, short payload from 8 to 32 bytes are transferred in a single response and regardless of their ways of transmission, time for DESFire card to send out response is always around or less than 2ms.

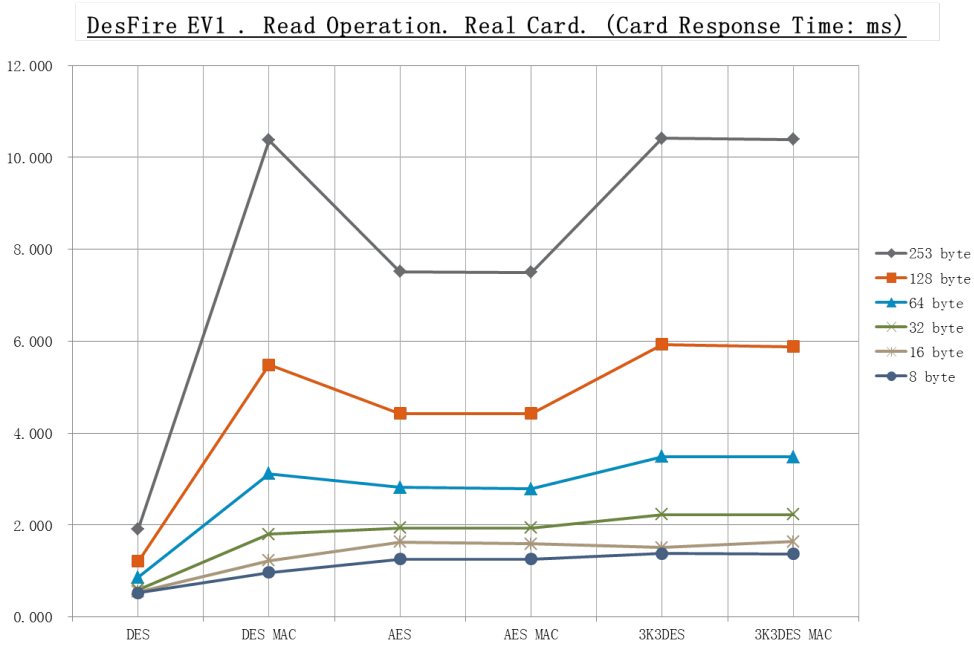

Figure 6.2: Response Time for Reading Data, Different Encryption methods, Real Card

Starting from 64 bytes to 253 bytes(as we marked as *,** and ***), data are transferred in 2 pieces, 3 pieces and 5 pieces. This is in compliance with ISO 7816 protocol specification about single APDU length. DESFire card will automatically split the answers into small trunks less than 64 bytes. Thus the respond time increases due to multiple transmissiones.

Among all the methods, Plain text method is always the easiest and fastest. DES with MAC option is as slow as 3K3DES encryption with MAC option. This makes us conclude that MAC calculation is more time consuming than simple rounds of DES calculation.

The AES encryption method is significantly faster than 3K3DES. It is the second

fast method next to plain text transmission. As AES provides the same encryption protection as 3K3DES, we strongly suggest our reader to use AES for the sake of saving time.

6.2.3 Relay Attack Delay Measurement (Sniffer)

After we have tested DESFire card basic transactions, now we collected enough data as benchmark of transaction timing. This time we investigate the time delay impact of different channels. As the nature of relay attack is inserting the attacker into the communication and forwarding messages, some delays are expected here. So the single command transmission time can no longer be near 2ms.

Channel/ Time(ms)	**DES Step1**	**AES Step1**	**3K3DES Step1**	**DES Step2**	**AES Step2**	**3K3DES Step2**
Real Card	2.34	2.65	2.75	2.28	2.72	3.12
Wi-Fi	67.91	71.59	72.63	73.54	74.03	74.03
Bluetooth	66.30	69.45	70.57	72.68	74.03	73.64

Table 6.6: Channel Delay created by Wi-Fi, Bluetooth

As we can see from table 6.6, the first line is the normal time for a DESFire card to respond to step 1 phase and step 2 phase of different authentication methods. Usually this phase only takes around 2-3ms as we have discussed above. However, by introducing two smart phones in between to do the relay attack, the single command response time increased into 60-70ms.

These delays are illustrated in figure 6.3 for a more direct and easy understanding. Although the Bluetooth 2.1 channel is slightly faster than Wi-Fi 802.11b channel, the worst case does not take more than 74.1ms. As far as our knowledge of DESFire EV1 specification, Frame Waiting Time is set to be 77ms as recommended, which means even the relay attack can survive the most strict time limitation in real environment.

Our delay in both Bluetooth channel and Wi-Fi channel in relay attack is acceptable. However, we have to mention that it is already close to the upper threshold, so further optimization is needed.

6.2.4 Relay Attack Delay Measurement (Java Application Side)

After we have a precise measurement with our tool Proxmark3 as sniffer, we return back to the Java NFC server side to have a more overall view of time delay. Despite each single APDU we sent out to execute our command, only several steps linked together can we perform meaningful operations and transactions. For example, server

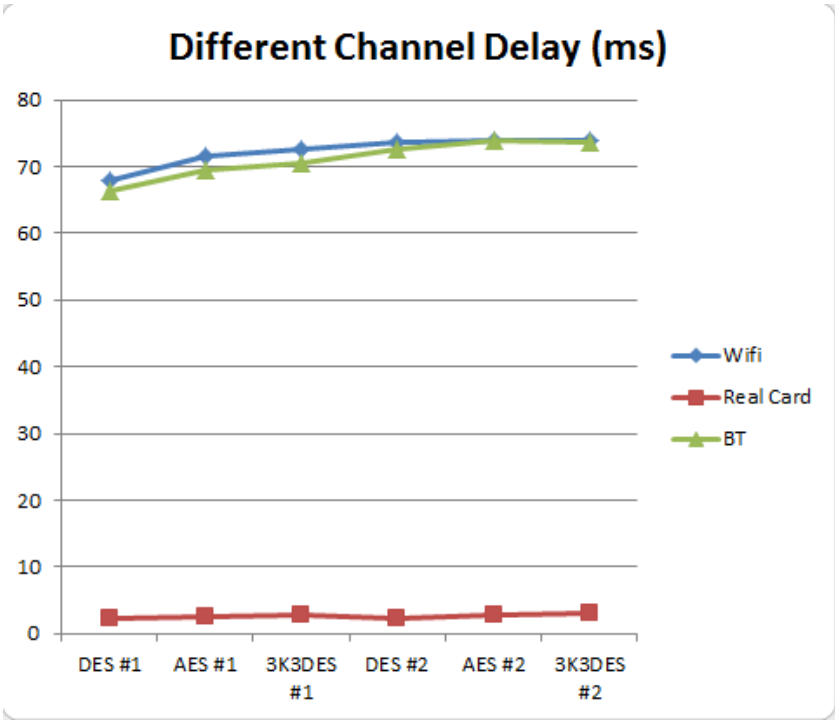

Figure 6.3: Different Channel Delay

side NFC application shall first authenticate itself then perform reading operation, or write to a file first then commit transaction to make the change permanent.

Thus we insert into our Java program time measurement codes to see the time delay between a real DESFire card and a relay attacked scenario. Here we choose only 32 byte small data for reading and writing purpose. All the file types are covered including value file, backup file, CRC file. The data can be seen in table 6.7.

To make it easier for reader to understand, we take the data and make a figure in 6.4. As we said in the beginning of this chapter, server side timing can never be accurate but gives a good, intuitive insightful look about what has happened from server observation. DESFire card makes a full-step transaction usually less than 150ms. In case it counters with CRC files, it takes much longer time.

Our Bluetooth or Wi-Fi relay channel takes much longer time than DESFire in terms of multiple-step transaction. The more transmission rounds a single transaction takes, the more we can see the delay. As we conclude here, the delay is accumulated

Name/Time(ms)	Real Card	Wi-Fi	Bluetooth
Create App	54.8	127.8	112.2
Select App	16	94.3	111.5
Authen App	51.4	223.9	202.9
Create CRC file	60	140.5	117.5
Create Value file	48	122.3	107.3
Create Backup file	52.4	131.5	112
Write CRC file	77	291.0	254.5
Commit CRC file	29	101.1	84.8
Write Backup file*	132.2	540.9	483.5
Commit Backup file	29.2	100.1	109.4
Read CRC file	212.6	932.7	672.5
Read Value file	37.4	208.0	197
Read Backup file	117.6	547.3	415.6

Table 6.7: Different execution time of transactions

and can be felt by server side programs easily. Taking reading backup file as an example. The direct command without relay attack only spent 117ms to finish this transaction, but if we insert relay attack phones with Wi-Fi and Bluetooth in between, it spent 547ms and 415ms respectfully. Server side business logic program can already suspect there is an abnormal behavior of client DESFire card.

6.3 Problems in Experiments

During our experiments, we have encountered several problems in collecting data.

- *Data corruption.* As our Proxmark3 sniffer has a limit ability, it sometimes cannot capture correct signal from high frequency antenna. Some bytes are lost. Especially when the buffer of sniffer is full, it starts to stop recording any new transactions. So some data we collected are corrupted. During the analysis, we find out more than 20
- *Limited memory.* Our Proxmark3 is powerful of sniffing, but it has a limited internal buffer. Usually it stops working halfway when we are doing heavy data transactions. Thus we have to stop manually from time to time to collect data and restart the Proxmark3.
- *Sniffing ability.* The Proxmark3 sniffer has a good performance in laboratory, but when it is taken out to be tested in real environment (bus card charging machine), it can only capture the transaction of deducting values. It cannot

capture a single signal when we are top-up our bus card credits. This might due to the frequency bus company used to top-up bus card is different from normal usage.

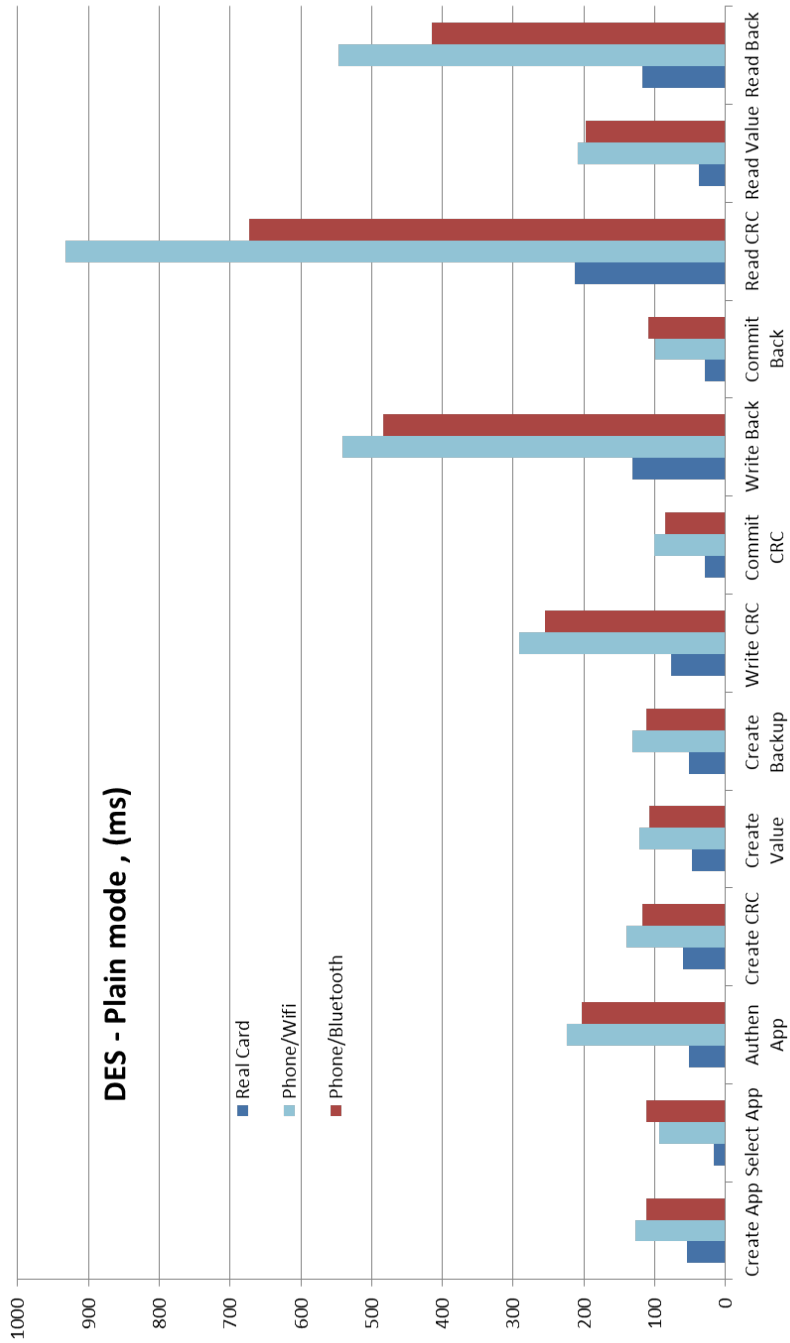

Figure 6.4: Delay of different operations from server side.

Chapter 7
Discussion

At this point, we have finished our experiments about our relay attack. Truly it succeeded in our laboratory environment and was an excellent proof of concept. However there are several things shall be taken into consideration. A single successful attack does not mean it can work everywhere. And also the relay attack itself has limitations. From those limitations, we can have effective preventions to stop smart phone based relay attacks.

7.1 Frame Waiting Time

According to ISO 14443 specifications, smart card and it pairing reader shall negotiate a proper Frame Waiting Time. This time window is the max waiting time for reader to expect card to finish a single command. In ISO 14443 protocol design, the default waiting time is 77ms(see section 2.2). As we can see our attack needs nearly 74ms to complete a single command, it would go exceeding this limit easily. However, in real life NFC applications, this time windows has not been properly implemented. In DESFire cards, ATS is not set to default value of 0x80 but instead set to `0x75 0x77 0x81 0x02 0x80`. See section 3.1. Thus this Frame Waiting Time is no longer calculated as 77.33ms. It depends how the programmer of NFC logic interprets it. But if the NFC program truly respects and implements the FWT, we suggest to enforce a smaller frame waiting time to eliminate more illegal relay attacks. Since the DESFire card can respond to reader in less than 77ms (See table 6.4, table 6.5), we suggest a smaller value shall be used. For example, any APDU response later than 50ms is considered invalid response. This FWT can at least elimiate our implementation of relay attack by smart phones.

7.2 Bluetooth vs Wi-Fi

As we have seen from the previous chapter, relay attacks in Bluetooth channel is always slightly faster than Wi-Fi channel. The difference is more obvious when

we execute multiple commands rather than single command. We have carefully measured the time at each point and conclude that under same software architecture, on Android Nexus S phone the Bluetooth stack is faster than Wi-Fi stack in our case. This may due to the Bluetooth chip or Wi-Fi chips capability, or may be the inside the implementation of operating system itself. Our further study would be comparing different communication channel efficiency under same software architecutre.

7.3 Cellular network

One of our goals in the very beginning of thesis is to do the relay attack in a much longer distance than Wi-Fi. To achieve this, we can put the traffic over cellular network (mobile broad band). However, as in theory it makes no difference to the experiments we have done right now, in practice user may suffer from potential traffic congestion and delay in cellular network. This may affect our user experience in a larger scale. We also have to point out that more delay means more rejection possibility from correctly implemented NFC applications, because they have a strict Frame Waiting Time.

7.4 Preventing Attack From The Beginning

As we have mentioned in the Chapter of Implementation, during the initial phase of activation and anti-collision, ISO 14443-4 protocol is used. And our smart phone emulated NFC cards has a significant shortcoming: 4-byte UID rather than regular 7-byte UID in a real card. This can be detected by NFC reader and thus we can reject the emulated NFC card at the very beginning. No further transactions are allowed as long as the UID is only 4 bytes long. But this may cause some new problem in the future. As we know that security can never depends on the UID because it can be read by everyone as plain text. Also advanced tools like Proxmark3 can easily spoof 7-byte UID. Thus this prevention is useless towards this type of hacking devices.

References

[ACL07] Marco Domenico Aime, Giogrio Calandriello, and Antonio Lioy. Dependability in wireless networks: can we rely on wifi? *Security & Privacy, IEEE*, 5(1):23–29, 2007.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[All07] Smart Card Alliance. Proximity mobile payments: Leveraging nfc and the contactless financial payments infrastructure. *Smart Card Alliance*, 2007.

[AM10] Adrian Atanasiu and Marius Iulian Mihailescu. Biometric passports (epassports). In *Communications (COMM), 2010 8th International Conference on*, pages 443–446. IEEE, 2010.

[And13] Daniel Andrade. Connecting nfc to the cloud, remote updating of smart cards. Master's thesis, Aalto University, 2013.

[BBRS06] Rafael Ballagas, Jan Borchers, Michael Rohs, and Jennifer G Sheridan. The smart phone: a ubiquitous input device. *Pervasive Computing, IEEE*, 5(1):70–77, 2006.

[CH07] Vipul Chawla and Dong Sam Ha. An overview of passive rfid. *Communications Magazine, IEEE*, 45(9):11–17, 2007.

[CKP+12] ByungRae Cha, DaeKyu Kim, SunMi Park, Jongwon Kim, and JaeHyun Seo. Concept design of micro payment model based on android nfc to reinvigorate traditional markets. In *Proceedings of the International Conference on Smart Convergence Technologies and Applications (SCTA?12)*, 2012.

[COO13] Vedat Coskun, Busra Ozdenizci, and Kerem Ok. A survey on near field communication (nfc) technology. *Wireless personal communications*, 71(3):2259–2294, 2013.

[Def06] NFC Record Type Definition. Nfc forum technical specification, 2006.

[DM08] Mohsen Darianian and Martin Peter Michael. Smart home mobile rfid-based internet-of-things systems and services. In *Advanced Computer Theory and*

Engineering, 2008. ICACTE'08. International Conference on, pages 116–120. IEEE, 2008.

[FP05] Erina Ferro and Francesco Potorti. Bluetooth and wi-fi wireless protocols: a survey and a comparison. *Wireless Communications, IEEE*, 12(1):12–26, 2005.

[Han05] Gerhard P Hancke. A practical relay attack on iso 14443 proximity cards. *Technical report, University of Cambridge Computer Laboratory*, pages 1–13, 2005.

[HB06] Ernst Haselsteiner and Klemens Breitfuß. Security in near field communication (nfc). In *Workshop on RFID security*, pages 12–14, 2006.

[IH11] Wolfgang Issovits and Michael Hutter. Weaknesses of the iso/iec 14443 protocol regarding relay attacks. In *RFID-Technologies and Applications (RFID-TA), 2011 IEEE International Conference on*, pages 335–342. IEEE, 2011.

[JTSM07] Päivi Jaring, Vili Törmänen, Erkki Siira, and Tapio Matinmikko. Improving mobile solution workflows and usability using near field communication technology. In *Ambient Intelligence*, pages 358–373. Springer, 2007.

[KW05] Ziv Kfir and Avishai Wool. Picking virtual pockets using relay attacks on contactless smartcard. In *Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference on*, pages 47–58. IEEE, 2005.

[Mar12] Konstantinos Markantonakis. Practical relay attack on contactless transactions by using nfc mobile phones. *Radio Frequency Identification System Security: RFIDsec*, 12:21, 2012.

[MB09] Michael Massoth and Thomas Bingel. Performance of different mobile payment service concepts compared with a nfc-based solution. In *Internet and Web Applications and Services, 2009. ICIW'09. Fourth International Conference on*, pages 205–210. IEEE, 2009.

[MLKS08] Gerald Madlmayr, Josef Langer, Christian Kantner, and Josef Scharinger. Nfc devices: Security and privacy. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 642–647. IEEE, 2008.

[Mul09] Collin Mulliner. Vulnerability analysis and attacks on nfc-enabled mobile phones. In *Availability, Reliability and Security, 2009. ARES'09. International Conference on*, pages 695–700. IEEE, 2009.

[NXP12] NXP. An10833 mifare type identification procedure. 2012.

[OP11] David Oswald and Christof Paar. Breaking mifare desfire mf3icd40: Power analysis and templates in the real world. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 207–222. Springer, 2011.

[Ort06] S Ortiz. Is near-field communication close to success? *Computer*, 39(3):18–20, 2006.

[RKSH07] Florian Resatsch, Stephan Karpischek, Uwe Sandner, and Stephan Hamacher. Mobile sales assistant: Nfc for retailers. In *Proceedings of the 9th international conference on Human computer interaction with mobile devices and services*, pages 313–316. ACM, 2007.

[RLS12] Michael Roland, Josef Langer, and Josef Scharinger. Relay attacks on secure element-enabled mobile devices. In *Information Security and Privacy Research*, pages 1–12. Springer, 2012.

[RNTS07] Jason Reid, Juan M Gonzalez Nieto, Tee Tang, and Bouchra Senadji. Detecting relay attacks with timing-based protocols. In *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 204–213. ACM, 2007.

[SC13] Luigi Sportiello and Andrea Ciardulli. Long distance relay attack. In *Radio Frequency Identification*, pages 69–85. Springer, 2013.

[Sem10] NXP Semiconductors. An 190810: Pn544 c2 antenna design guide. *Application Note, Rev*, 1, 2010.

[SVV10] Jarkko Sevanto, Petri Vesikivi, and Pekka K Viitaniemi. Phone with secure element and critical data, April 6 2010. US Patent 7,694,331.

[Wan06] Roy Want. An introduction to rfid technology. *Pervasive Computing, IEEE*, 5(1):25–33, 2006.

[ZN06] Pei Zheng and Lionel M Ni. Spotlight: the rise of the smart phone. *Distributed systems online, IEEE*, 7(3), 2006.