Security of NFC applications

THI VAN ANH PHAM

KTH Information and
Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

Aalto University
School of Science
Degree Programme in Security and Mobile Computing

Thi Van Anh Pham

Security of NFC applications

Master's Thesis
Espoo, June 30, 2013

Supervisors:	Professor Tuomas Aura, Aalto University
	Professor Gerald Q. Maguire Jr., KTH Royal Institute of Technology
Instructor:	Sandeep Tamrakar, M.Sc. (Tech.), Aalto University

Aalto University
School of Science
Degree Programme in Security and Mobile Computing

ABSTRACT OF MASTER'S THESIS

Author:	Thi Van Anh Pham		
Title: Security of NFC applications			
Date:	June 30, 2013	**Pages:**	xii + 88
Professorship:	Data Communication Software	**Code:**	T-110
Supervisors:	Professor Tuomas Aura Professor Gerald Q. Maguire Jr.		
Instructor:	Sandeep Tamrakar, M.Sc. (Tech.)		

Near Field Communication (NFC) refers to a communication technology that enables an effortless connection and data transfers between two devices by putting them in a close proximity. Besides contactless payment and ticketing applications, which were the original key drivers of this technology, a large number of novel use cases can benefit from this rapidly developing technology, as has been illustrated in various NFC-enabled application proposals and pilot trials.

Typical NFC-enabled systems combine NFC tags, NFC-enabled mobile phones, and online servers. This thesis explores the trust relationships, security requirements, and security protocol design in these complex systems. We study how to apply the security features of different types of NFC tags to secure NFC applications. We first examine potential weaknesses and problems in some novel use cases where NFC can be employed. Thereafter, we analyze the requirements and propose our system design to secure each use case. In addition, we developed proof-of-concept implementations for two of our proposed protocols: an NFC-enabled security-guard monitoring system and an NFC-enabled restaurant menu. For the former use case, we also formally verified our proposed security protocol.

Our analysis shows that among the discussed tags, the NFC tags based on secure memory cards have the least capability and flexibility. Their built-in three-pass mutual authentication can be used to prove the freshness of the event when the tag is tapped. The programmable contactless smart cards are more flexible because they can be programmed to implement new security protocols. In addition, they are able to keep track of a sequence number and can be used in systems that do not require application-specific software on the mobile phone. The sequence number enforces the order of events, thus providing a certain level of replay prevention. The most powerful type of tag is the emulated card since it provides a clock, greater computational capacity, and possibly its own Internet connection, naturally at higher cost of deployment.

Keywords:	NFC, security protocol, DESFire, Java card, card emulation, restaurant, vending machine, class attendance, security guard
Language:	English

Aalto-universitetet
Högskolan för teknikvetenskaper
Examensprogram för datateknik

SAMMANDRAG AV DIPLOMARBETET

Utfört av:	Thi Van Anh Pham		
Arbetets namn:			
Säkerheten för NFC-applikationer			
Datum:	Den 30 Juni 2011	**Sidantal:**	xii + 88
Professur:	Datakommunikationsprogram	**Kod:**	T-110
Övervakare:	Professor Tuomas Aura Professor Gerald Q. Maguire Jr		
Handledare:	Sandeep Tamrakar, M.Sc. (Tech.)		

Near Field Communication (NFC) hänvisar till en kommunikationsteknik som möjliggör en enkel anslutning och dataöverföring mellan två enheter genom att sätta dem i en närhet. Förutom kontaktlös betalning och biljetthantering ansökningar, vilket var den ursprungliga viktiga drivkrafter för denna teknik, kan ett stort antal nya användningsfall dra nytta av denna snabbt växande teknik, som har visats i olika NFC-aktiverade program förslag och pilotförsök.

Typiska NFC-applikationer kombinerar NFC-taggar, NFC-kompatibla mobiltelefoner och online-servrar. Denna avhandling utforskar förtroenderelationer, säkerhetskrav och säkerhetsprotokoll utformning i dessa komplexa system. Vi studerar hur man kan tillämpa de säkerhetsfunktioner för olika typer av NFC-taggar för att säkra NFC-applikationer. Vi undersöker först potentiella svagheter och problem i vissa nya användningsfall där NFC kan användas. Därefter analyserar vi de krav och föreslå vårt system design för att säkra varje användningsfall. Dessutom utvecklade vi proof-of-concept implementationer för två av våra föreslagna protokoll: en NFC-aktiverad säkerhet-guard övervakningssystem och en NFC-aktiverad restaurang meny. Dessutom, för fd bruk fallet, kontrollerade vi formellt vår föreslagna säkerhetsprotokoll.

Vår analys visar att bland de diskuterade taggar, NFC taggar som baseras på säkra minneskort har minst kapacitet och flexibilitet. Deras inbyggda trepass ömsesidig autentisering kan användas för att bevisa färskhet av händelsen när taggen tappas. De programmerbara beröringsfria smarta kort är mer flexibla eftersom de kan programmeras för att genomföra nya säkerhetsprotokoll. Dessutom kan de hålla reda på ett löpnummer och kan användas i system som inte kräver ansökan-specifik mjukvara på mobiltelefonen. Sekvensnumret framtvingar ordning av händelser, vilket ger en viss nivå av replay förebyggande. Den mest kraftfulla typen av taggen är den emulerade kortet eftersom det ger en klocka, större beräkningskapacitet, och möjligen sin egen Internet-anslutning, naturligtvis till högre kostnad för utplacering.

Nyckelord:	NFC, säkerhetsprotokoll, DESFire, Java kort, kort emulering, restaurang, varuautomat, säkerhetsvakt
Språk:	Engelska

Acknowledgements

I sincerely thank Professor Tuomas Aura for his constant feedback, discussions and his comments on my thesis. He has inspired me with his immense knowledge and experiences in the field.

I am grateful to Professor Gerald Q. Maguire Jr. at KTH Royal Institue of Technology for co-supervising this thesis, and giving me comments on different draft versions of the thesis.

I would also like to gratefully thank my instructor Sandeep Tamrakar for his guidance and comments. I also thank Correia Andrade Daniel for discussions with him about the DESFire tag authentication procedure.

Finally, I would like to thank my family for their unconditional love and supports.

Espoo, June 30, 2013

Thi Van Anh Pham

Abbreviations and Acronyms

AID	Application Identifier
CC	Capability Container
CPU	Central Processing Unit
DoS	Denial of Service
GCC	GNU Compiler Collection
GPS	Global Positioning System
HMAC	Keyed-Hashed Message Authentication Code
HR	Human Resources
HTTP	Hypertext Transfer Protocol
IC	Integrated Circuit
ID	Identifier
IMEI	International Mobile Station Equipment Identity
IMSI	International Mobile Subscriber Identity
JCRE	Java Card Runtime Environment
JCVM	Java Card Virtual Machine
MAC	Message Authentication Code
MB	Message Begin
ME	Message End
MIME	Multipurpose Internet Mail Extensions
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
NFCIP	Near Field Communication Interface and Protocol
PCD	Proximity Coupling Device
POS	Point Of Sale
RF	Radio Frequency
RFID	Radio Frequency Identification
SE	Secure Element
SMS	Short Message Service
SP	Service Provider
TLS	Transport Layer Security

TLV	Type - Length - Value
TNF	Type Name Field
UID	Unique Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus

Contents

List of Figures

Chapter 1

Introduction

Near Field Communication (NFC) is a short-range radio communication technology that promises to enhance our everyday tasks by the convenience of its “tap and go” principle [2], i.e. enabling users to simply touch two NFC-enabled devices together to establish a communication session between them, thus making applications and data exchange easy and convenient. The primary drivers promoting the adoption of NFC are contactless payments and ticketing applications, but the expected success of NFC has expanded and covered a wide variety of other applications [3], such as location-based services, gaming, access controls, and device pairing.

A typical NFC application is a distributed system which consists of three main components, specifically NFC tags, NFC readers or NFC-enabled mobile phones, and online servers, as shown in Figure 1.1. These three components communicate and coordinate their actions by passing messages. To be more specific, the NFC readers or the NFC-enabled phones communicate with the NFC tags over NFC, while the NFC readers or the NFC phones and the servers can communicate over the Internet. Therefore, a system of this type introduces a new type of security protocol which involves the three components and the communication channels between them in order to secure NFC applications.

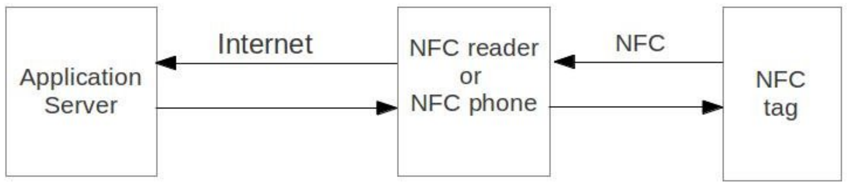


Figure 1.1: Architecture of an NFC-enabled system

As with all modern communication systems, security and system usability are important in NFC-enabled systems. However, since potential NFC-enabled systems are nearly endless and various types of NFC devices can be involved in these systems, there is no "one size fits all" security solution for all NFC systems. Instead, the set of possible security designs for a specific application heavily depends on the types of NFC tags used in the application. This is because different NFC tags provide different security features and capabilities. For example, some NFC tags are just memory tags and thus extremely constrained in terms of computational capabilities. On the contrary, more powerful tags, such as Java cards and NFC readers running in card emulation mode, can provide more intelligent functionality, such as the ability to be programmed and to perform calculations.

However, many NFC applications that have been proposed in the literature do not carefully consider the types and capabilities of NFC tags before applying them to the systems. This results in some designs which exhibit poor usability and potential security flaws. Therefore, in this thesis, we present some of the problems which we have found in several proposed NFC applications, and then design security protocols to secure them by considering the security requirements of the systems, user experience, and the types of NFC tags that should be employed in the systems. In our conclusions, we discuss the general principles of using different NFC tags for securing NFC applications.

1.1 Problem Statement and Methodology

As presented above, there is a wide, ever expanding range of uses for NFC that is being explored and made available in the world. The technology promises to benefit a variety of areas via numerous applications. However, different NFC applications have different security requirements and rely on different security features and other capabilities of the NFC devices involved in the applications. Therefore, the NFC tag capabilities and application requirements should be carefully considered when designing a system. In addition, usability is another important aspect contributing to the success of a system, hence this too must be considered during the design process.

The problem addressed in this thesis is how to exploit the different capabilities of NFC tags to secure NFC applications. To reach an answer, we follow these steps:

- We study some specific use cases in which NFC can be used to make these use cases more convenient to use and more secure.

- Study the security requirements of these specific use cases and design security protocols for them.
- Develop proof-of-concept implementations for two of our proposed protocols.
- Generalize the principles of using different NFC tags in the security of NFC applications.

1.2 Use cases

The following sub-sections introduce the use cases that are examined in this thesis.

1.2.1 NFC-enabled restaurant menu

Our proposed scenario for an NFC-supported restaurant menu is as follows: a customer (Bob) who has an NFC-enabled Internet-connected phone goes to a restaurant. Instead of waiting for the waiter, he taps his phone on an NFC tag glued to a table in the restaurant and his phone's browser is redirected to the restaurant's web page that lists all the meal options. He chooses some food, submits his choice; then the restaurant's kitchen will prepare the food and serve him when it is ready. Before leaving the restaurant, the customer pays for his meal by using his bank card or cash like in normal restaurants.

The security problem that we aim to solve in this scenario is the following: the food ordering is web-based and does not require users to log in to be able to use the service. Thus, anyone can order the food without revealing his identity. This means that if the URL of the web page is stored or remembered, then a dishonest person could stay at home, access the webpage and make as many food orders as he wants. Since the kitchen does not know that these submissions are fake, they prepare the food even though no one is there to pay for it. Therefore, we need to prevent this problem from occurring by applying security mechanisms. Additionally, the service is general and should be open to all people who enter the restaurant with an NFC-enabled phone. Therefore, our design do not require users to install a dedicated application in their phones to be able to use the service.

1.2.2 Security guard monitoring system

Most of our workplaces and offices have information or equipment that must be kept safe. Along with alarm systems, security guards frequently make rounds to check that locks are locked, doors are closed, and if direct actions need to be taken. Hence, the presence of guards at checkpoints at their scheduled time is important and needs to be monitored. The current widely used technique is to employ rugged readers and checkpoint tags which are available in different forms, such as RFID tags, and barcodes [4]. There are also some NFC-based security guard monitoring systems available on the market that use NFC-enabled phones and NFC tags attached to checkpoints [5–7]. At predefined intervals, each security guard uses a dedicated device to scan the barcode or RFID tag, or uses an NFC phone with a dedicated application to scan the NFC tag at a checkpoint, to prove his or her presence at this checkpoint.

However, we have found that both solutions exhibit security problems, as will be explained in detail in Section 3.1.1. Basically, these current solutions allow dishonest security guards to make copies of the barcodes or tags. This means that dishonest security guards could confirm their presence at a checkpoint while they actually do not come to work. Therefore, the main security goal in this use case is to guarantee that it is not possible for the security guards to create fake presence confirmations without doing their actual rounds. This scenario is different from the restaurant scenario in that the number of users is limited and the users belong to a specialized group. Therefore, the security guards could be provided with a dedicated phone having a dedicated application installed.

1.2.3 Vending machines

Recently, SMS-enabled vending machines have become popular. Instead of using coins or bank notes, a customer could use his or her phone to pay for products from the vending machines. Specifically, in order to buy an item from a vending machine, the customer composes an SMS message that contains the machine ID and the item's price, then sends the message to a short code. However, machines of this type have two security problems. Firstly, it allows a person who is not close to a vending machine to buy items from it. This is a problem if we think about a situation where Alice is talking to Bob who is a prankster. Alice goes out for several minutes and leaves her phone on the table. Bob may use Alice's phone to buy products from a possibly remote vending machine without Alice's permission. Secondly, a dishonest person could change the machine ID written on a machine named A to the ID of machine B. Then he waits in front of machine B to collect the beverage which someone else purchases from machine A.

Our observation is that NFC can be used to facilitate SMS message composition. In addition, we can design security protocols to deal with the two security problems presented above. Specifically, there are two security goals that we aim to achieve in this use case: (i) preventing a person who is not close to a specific vending machine from buying items from that machine, and (ii) preventing a malicious person from getting free items by changing the machine ID of a machine to the ID of another machine.

1.2.4 School attendance monitoring systems

Student absenteeism is a significant problem at many educational institutions and a major concern for educators. It results in high rates of course failure and, over the long term, greater likelihood of dropping out of school. In addition, at many schools where education is free, students can register for as many courses as they want, but never show up for any of these classes. This leads to waste of resources that have been invested in education. Therefore, a mandatory class attendance policy has been applied in many institutions to force students to attend all their scheduled classes. In addition, in some countries, such as India, a teacher attendance policy is also enforced. There are some solutions for student and employee attendance monitoring systems that have been proposed in the literature (Section 2.10). However, these solutions provide poor user experience, as will be explained in Section 3.4. Therefore, in this use case, our solution aims to provide a reliable and effective NFC-enabled system that facilitates school attendance supervision for both students and teachers.

1.3 Structure of the thesis

The rest of this thesis is divided into four chapters. Chapter 2 starts by describing NFC technology and NFC devices, along with an introduction to NFC type 4 tags and NFC Data Exchange Formats (NDEF). It then discusses the security mechanisms defined in NFC and security threats relevant to NFC, followed by specific types of NFC tags that are used in the four specific use cases described in Section 1.2. It concludes with an overview of NFC applications that have been previously proposed in the literature. Chapter 3 presents detailed secure NFC-enabled application designs for the four use case mentioned above. Specifically, in each use case, we review its current solutions and potential security problems in these solutions. Thereafter, we present our design goals, our proposed system design, and analysis of the solution. Chapter 4 presents the implementation of two use cases, specifically the NFC-enabled restaurant menu and the NFC-enabled security guard monitoring system. Chapter 5 generalizes the uses of different NFC tag types to secure NFC applications and explains how we verified our proposed protocols. Finally, Chapter 6 provides a summary of the thesis and suggest potential future work.

Chapter 2

Background

This chapter begins by describing NFC technology and NFC devices, followed by an introduction to NFC type 4 tags and NFC Data Exchange Formats (NDEF). Since we analyse security requirements and design security protocols for NFC applications, the security mechanisms defined in NFC and security threats relevant to NFC are presented. Thereafter, we discuss Mifare DESFire EV1 secure memory cards, programmable Java card technology, and NFC card emulation on an embedded computer, which are the three specific tags used in this thesis. The last section of the chapter describes NFC applications that have been proposed in the literature.

2.1 NFC

Near Field Communication (NFC) is a wireless proximity communication technology that enables data transfers between two devices which are close to each other, typically to a distance of less than 10 centimeters [8]. This simple means of establishing a connection is a major advantage of NFC over other wireless communication technologies, such as Bluetooth [9] and Wi-Fi [10]. However, compared to connection speeds of Bluetooth and Wi-Fi, NFC provides slower data transmission rate of up to 424 kilobits per seconds (kbps) [8]. In addition, the communication range of NFC is shorter than that of other communication technologies. However, this is not considered a drawback, but an inherent characteristic and a technical advantage of this technology. To be more specific, the close communication range enables intuitive transfers of data by tapping one device against a desired peer device and ignoring other devices that are outside this communication range. This not only helps to prevent signal interference between devices, but also secures users and applications, since users must be close enough to an NFC device to be able to nearly "touch" it, or in other words, in most cases, they intentionally wish to use the application.

NFC is based on the Radio Frequency Identification (RFID) technology. It operates at 13.56 MHz and relies on ISO14443 and ISO 18092 for low-level data exchange between two NFC devices [8]. Specifically, these two ISO standards specify the operating frequency, modulation, coding schemes, anti-collision routines, and communication protocols. NFC data exchange formats and NFC tag formats are defined by the NFC Forum[1].

NFC devices can be divided into two categories: active and passive devices. An active device, such as an NFC-enabled phone or an NFC card reader, is always connected to a power source or has batteries attached. Moreover, it generates an electromagnetic field when it wishes to communicate with its desired NFC peer. On the contrary, a passive device often does not have any power source, except the electromagnetic field generated by active devices that is in proximity of the passive device. Because of this, active devices must continuously poll for passive devices to detect if there is a passive device available in its range. Examples of passive devices are NFC tags, contactless smart cards, and NFC-enabled phones in card emulation mode. More details about active devices and passive devices are presented in the following sections.

2.1.1 NFC Active Devices

Active NFC devices can operate in three different modes [11]:

1. Reader/writer mode: In this mode, an active device is capable of reading and modifying data stored in passive devices. This mode is standardized in the ISO 14443 standard [12].

2. NFC peer-to-peer mode: The physical and data link layer of NFC peer-to-peer mode is standardized in ISO 18092 [13]. This mode allows a bidirectional data exchange between two active devices. For example, Bluetooth pairing parameters or virtual business cards could be exchanged between two NFC-enabled phones.

3. Card emulation mode: This mode uses ISO 14443 as standard for its physical and data link layer. An active NFC device operating in this mode appears to an external active device much the same as a passive device. For example, a phone working in emulation mode can present itself as a contactless credit or debit card. To make a payment, a user

[1]http://www.nfc-forum.org

simply selects the payment application, and holds the phone in front of a contactless reader. Therefore, we can design contactless payments and ticketing applications on mobile phones without changing the existing infrastructure.

2.1.2 NFC Tags

An NFC tag, for example a sticker or a wristband, is a passive device that consists of a small microchip, a little antenna, and a small memory to store data for transfer to active devices. Each tag is identified by its unique identifier (UID). In addition, different tags have different memory capacities, communication speed, and security features. For example, a Mifare Ultralight [14] tag is able to hold at most 48 bytes of data, whereas a Mifare DESFire EV1 tag [15] can store up to 8KB of data. A Mifare Ultralight tag does not support authentication of the card reader before reading or modifying of the data stored in the tag, while this feature is provided by some other tags, such as Mifare DESFire EV1 and Mifare Ultralight C [16] tags.

An NFC tag can store one or more application-defined data as payloads. Usually, these payloads are encapsulated first into a single NFC Data Exchange Format (NDEF) message, and then mapped into the data structure specified by the tag platform. The NDEF message and the tag platform encapsulations are used to identify the type of application data and to guarantee interoperability and co-existence between applications [17]. More detail about NDEF messages is presented in Section 2.2.

So far, the NFC Forum has defined four types of tags [18]. The following section discusses the NFC type 4 tags.

2.2 NFC Type 4 Tags

An NFC type 4 tag can store one or more NDEF applications which are identified by their application identifiers (AID). Each application contains a Capability Container (CC) file and one or more NDEF files. Each NDEF file can contain multiple NDEF messages [19]. Figure 2.1 illustrates an example of an NDEF tag application that contains two NDEF files.

In an NDEF application, a CC file with value 0xE103 indicates to the reader that the tag contains NDEF messages so that the reader can access and modify the data accordingly. The CC file is read-only and contains the following information:

- CC Length: This field is 2 bytes long and specifies the size of the CC file (this field included).
- Mapping Version: This field is 1 byte in length and indicates the mapping specification version with which this CC file is compliant.
- MLe: This 2-byte field specifies the maximum data size (in bytes) that can be read from the tag using a single read command
- MLc: This field is 2 bytes long, defining the maximum data size that can be sent to the tag using a single update or write command
- One or more NDEF File Control TLV (Type - Length - Value) blocks: A TLV block contains information to control and manage an NDEF file. Specifically, an NDEF file control TLV has T equal to 0x04, L equal to 0x06, and the value field is composed of 6 bytes that specify the size, read and write access conditions, and the identifier of the NDEF file which points to another file in the tag file system.

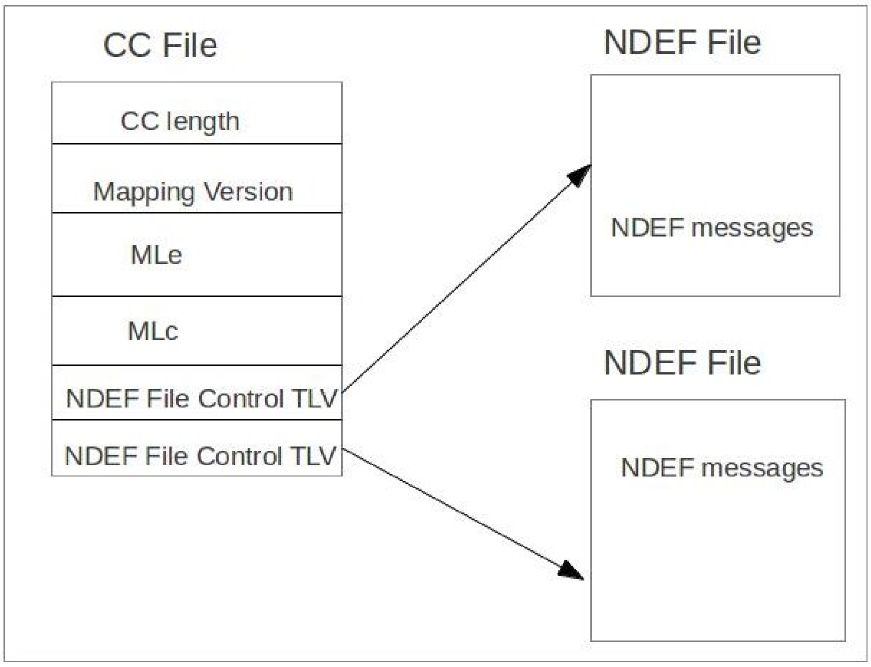


Figure 2.1: An NDEF application with two NDEF files

2.3 NDEF Messages

The NDEF specification [1] defines a message encapsulation format to exchange information between NFC devices, e.g. between a reader and a tag.

Figure 2.2 illustrates the general structure of an NDEF message. Specifically, an NDEF message contains one or more NDEF records (Section 2.3.1). These records can be chained together to support a larger payload. Each NDEF record uses three parameters: payload length, payload type, and an optional payload identifier to describe its payload. The first NDEF record in an NDEF message has the MB (Message Begin) flag set, while the last NDEF record is marked with an ME (Message End) flag. This means that an NDEF message with only one NDEF record has both MB and ME flags set.

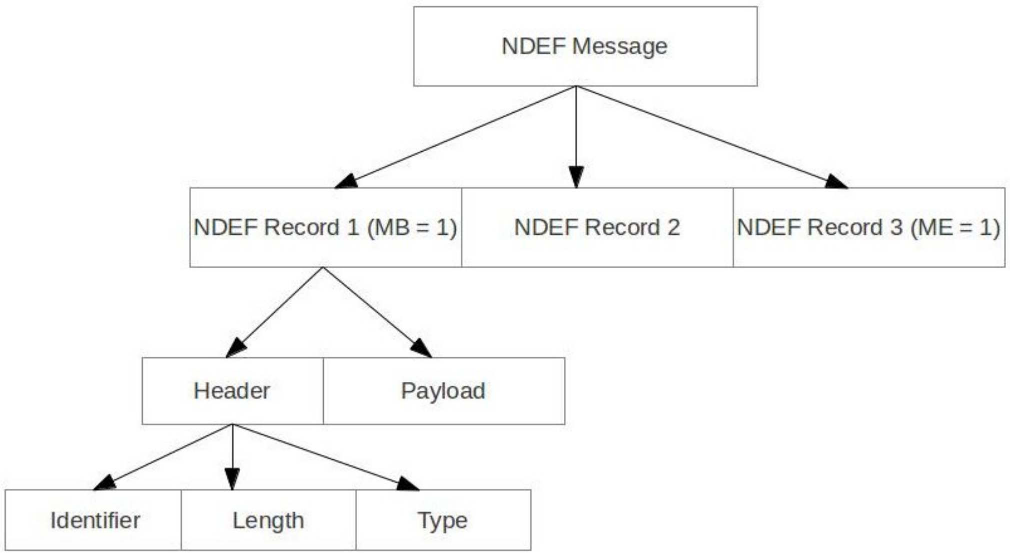


Figure 2.2: Structure of an NDEF message with three NDEF records

2.3.1 NDEF Records

An NDEF record is the unit for carrying a payload within an NDEF message. The record layout is shown in Figure 2.3, followed by descriptions of each field.

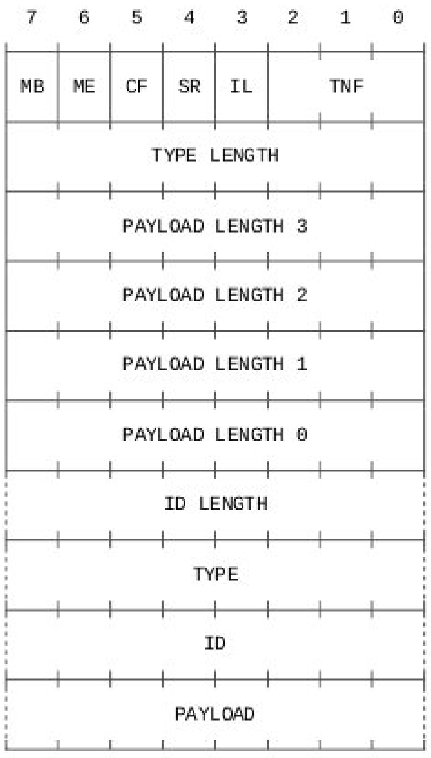


Figure 2.3: NDEF record layout [1]

MB: Message Begin flag, set to 1 when the record is the first record of an NDEF message, 0 otherwise.

ME: Message End flag, set to 1 when the record is the last record of an NDEF message, 0 otherwise.

CF: Chunk flag, indicating that this is either the first record chunk or a middle record chuck of a chunked payload.

SR: Short Record flag. If this flag is set, the PAYLOAD_LENGTH is a single octet instead of 4 octets as shown in the figure 2.3.

IL: The IL flag is a 1-bit field indicating that the ID_LENGTH field is present in the header or not.

TNF (Type Name Format): This 3-bit long field indicates the structure of the value of the TYPE field defined in Table 2.1.

TYPE LENGTH: an unsigned 8-bit integer specifying the length in octets of the TYPE field. This field is always 0 for certain values of the TNF field.

ID LENGTH: specifies the length in octets of the ID field. This field is present only if the IL flag is set to 1.

PAYLOAD LENGTH: specifies the length in octets of the PAYLOAD field. The size of this field could be 1 or 4 octets depending on the value of the SR flag.

TYPE: describes the type of the payload. The value of TYPE field must follow the structure, encoding and format implied by the value of the TNF field.

ID: an identifier in the form of a URI reference.

PAYLOAD: contains the payload.

Type Name Format	Value
Empty	0x00
NFC Forum well-known type	0x01
Media-type as defined in RFC 2046	0x02
Absolute URI as defined in RFC 3986	0x03
NFC Forum external type	0x04
Unknown	0x05
Unchanged	0x06
Reserved	0x07

Table 2.1: TNF field values

2.4 NDEF Read and Write Procedure in NFC Type 4 Tags

When a tag is in the proximity of a reader, the reader starts sending commands to the tag to detect if the tag contains any NDEF messages or not. It is worth mentioning that in this detection process, the NDEF file referenced by the NDEF TLV block at offset 0x0007 of the CC file is used. The procedure is described in Figure 2.4 [1].

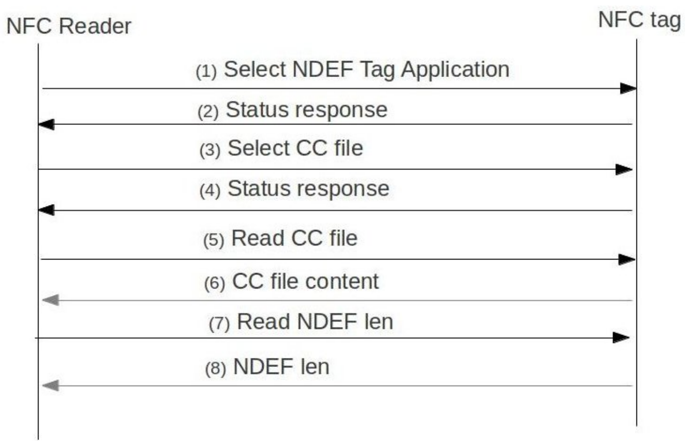


Figure 2.4: The NDEF detection procedure

If the NDEF length returned by the tag is in a valid range, the NFC reader knows that the tag contains at least one NDEF message. Depending on the tag communication settings and its read and write access rights, an authentication procedure between the reader and the tag may need to be carried out before these operations can take place. The following parts present read and update procedures when the NFC tag does not have any security settings.

NDEF Read Procedure NDEF read operations (Figure 2.5) happen after the reader successfully finds an NDEF message in the tag (Figure 2.4). If the requested NDEF file is the NDEF file that is referenced by the NDEF TLV block at offset 0x0007 in the CC file of the application, then the third and the fourth steps in the read procedure illustrated in Figure 2.5 can be skipped [1].

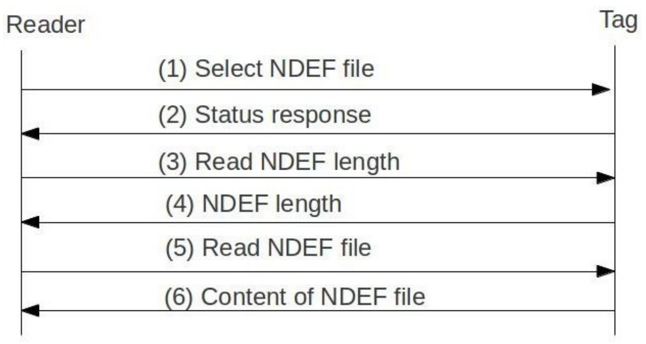


Figure 2.5: The NDEF read procedure

NDEF Write or Update Procedure NDEF write or update operations happen after the card NDEF detection procedure presented in Figure 2.4 is successfully completed and the NDEF length returned from the tag is greater than or equal to 0x0000. The write or update procedure is shown in Figure 2.6 [1]. In this procedure, if the requested NDEF file is the NDEF file that is referenced by the NDEF TLV block at offset 0x0007 in the CC file of the application, then the third and the fourth steps in the read procedure can be skipped. In addition, the messages in the fifth, sixth, and seventh steps can occur in a single update command if the desired written content fits inside the data field of an NDEF update command [1].

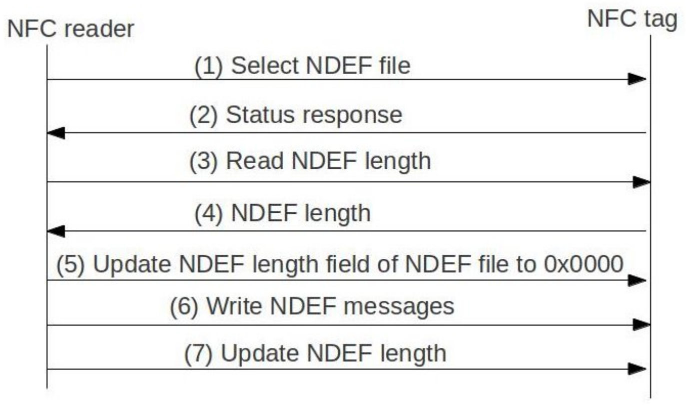


Figure 2.6: The NDEF write procedure

Example of the NDEF Read Procedure The following is a an example of the NDEF read procedure between a reader and a tag using the command set specified in ISO 7816-4 [20].

```
READER: 00 a4 04 00 07 d2 76 00 00 85 01 01 00
TAG : 90 00 (CORRECT EXECUTION)
READER: 00 a4 00 0c 02 e1 03 (SELECT CC FILE (ID: 0xE103))
TAG: 90  00  (CORRECT EXECUTION)
READER: 00 b0 00 00 0f  (READ CC FILE)
TAG: 00 0f 20 00 54 00 ff 04 06 e1 04 ff fe 00 00 90 00
READER:00 a4 00 0c 02 e1 04    (SELECT NDEF FILE)
TAG: 90 00   (CORRECT EXECUTION)
READER: 00 b0 00 00 02  (READ NDEF LEN)
TAG: 00 11 90 00  (NDEF LEN + CORRECT EXECUTION)
```

```
READER: 00 a4 00 0c 02 e1 04
(SELECT NDEF FILE (ISO FILE IS 0xE104))
TAG: 90 00  (CORRECT EXECUTION)
READER: 00 b0 00 00 11 (READ NDEF FILE)
TAG: 00 0f d1 01 0b 55 01 67 6f 6f 67 6c 65 2e 63 6f 6d 90 00
```

2.5 Security Mechanisms Defined in NFC

Low-level protocols in NFC, including ISO 14443 and ISO 18092, do not specify any specific encryption or security mechanisms to secure data transfered between two NFC devices [21]. However, NDEF specifications define a signature scheme for integrity and authenticity of NFC tag content, i.e. the data records within an NFC tag can be signed. The NFC tag signature scheme is presented in the next section.

2.5.1 The NFC Tag Signature Mechanism

A signature record is calculated over all records that start either from the first record of the NDEF message or from the first record following the preceding signature record, as shown in Figure 2.7. A signature record itself is not signed. The signature applies to the Type, ID (if present), and Payload fields of all records to be signed. However, the first byte of the NDEF header, including MB, ME, CF, SR, IL and TNF fields, is excluded [22]. In addition, it is worth noting that the signature scheme *ignores* the tag UID, thus allowing tag cloning. Also, it does not include reader authentication for access control.

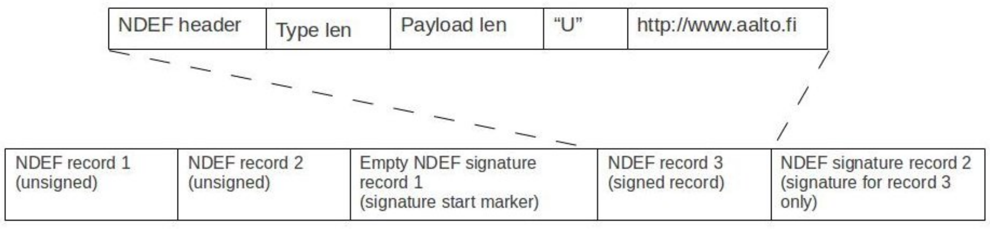

NDEF header
Type len
Payload len
“U”
http://www.aalto.fi
NDEF record 1 (unsigned)
NDEF record 2 (unsigned)
Empty NDEF signature record 1 (signature start marker)
NDEF record 3 (signed record)
NDEF signature record 2 (signature for record 3 only)

Figure 2.7: NDEF signature

Roland et al. presented a record composition attack which aims at composing records in such a way that the digital signature remains valid [23]. Saeed and Walter [22] presented procedures to hide records in an

NDEF message, but still keeping its signature valid. They also extend the decomposition attacks presented by Roland et al. [23]. The attack works as follows: the text of a smart poster states: “Do not board the train until you have a valid ticket”. This text is digitally signed and the signature is stored in a signature record. An attacker may split this message into two separate records. The first record stating “Do not board the train” is visible to the user, whereas the second record stating “until you have a valid ticket” does not appear to the user since the NDEF type of this part has been changed to an NFC unknown type. However, the digital signature remains valid and the user will consider the whole message as a valid message.

2.6 Security Threats Relevant to NFC

1. Denial of Service: In NFC, a DoS attack is possible because when an NFC reader and a tag are close enough, the reader will start reading the tag even if the tag is empty. This is because the tag is passive and harvests energy from the signal from the reader. Thus, the reader must continuously poll for tags to detect if there is a tag available in its range. This means that the NFC reader could be occupied or kept busy by putting an NFC tag within the reader’s proximity [21]. To avoid this, most mobile phones automatically turn off their NFC read and write functionality when the screen is off.

2. NFC relay attack: It has been suggested that NFC systems are particularly vulnerable to relay attacks. Francis et al. [24] presented a practical relay attack on NFC peer-to-peer mode using Near Field Communication Interface and Protocol (NFCIP) between two NFC-enabled mobile phones. The set up of this relay attack is shown in Figure 2.8. Specifically, two proxy NFC phones that are 100 meters away from each other establish a Bluetooth connection to forward messages from the initiator device to the target device. One of the proxy phones presents itself as the target phone to the initiator while the other one presents itself as the initiator to the target phone.

   An NFC relay attack is possible not only in peer-to-peer mode, but also in reader/writer mode. Francis et al. [25] presented a proof-of-concept relay experiment. To be more specific, two proxy enabled phones establish a Bluetooth connection between them to forward messages between a contactless smart card and a reader. One of the proxy phones presents itself as the contactless smart card to the original reader while the other presents itself as the reader to the contactless smart card.

Although both relay experiments used a Bluetooth connection between the two relay phones, any high-speed and reliable communication link between them would work.

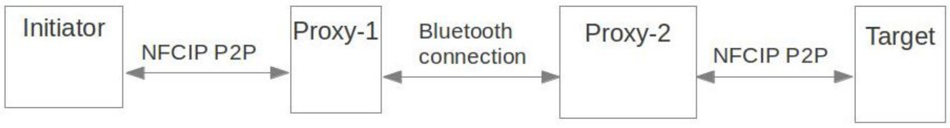


Figure 2.8: NFC relay attack in peer-to-peer mode

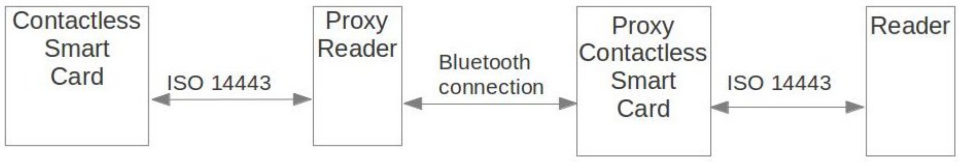


Figure 2.9: NFC relay attack in reader/writer mode

3. Spoof the tag content: An attacker could supply false information, such as a worm-URL or a false short code, to a user's device. For example, a tag in a smart poster which supports purchase of bus tickets via SMS messages could be replaced with another tag that contains an SMS message to a premium-rate service. When a user his or her phone to tap on the tag, the phone receives a pre-composed SMS message from the tag. After that, the user is asked if he or she wants to send the message. However, a user in a hurry might not check the SMS message carefully and hence might send the false SMS message.

4. Tag cloning: As presented in Section 2.2, the NDEF signature scheme ignores the tag UID. This means that an NDEF message written on a tag can be copied and written to another tag. However, some tags, such as DESFire EV1, and Ultralight C [16] tags, can be set to require user authentication before changing the data stored on them. This mechanism helps to lower the risk of tag cloning since only readers that share a secret key with the tag can access the data stored in the tag. However, if one of these readers is dishonest, it can reproduce the tag content without any obstacle.

5. Tag replacement or tag stacking: An attacker can replace a tag with his or her own tag that contains whatever malicious contents he or she

wishes. He or she could also stack a fake tag on top of the original tag to achieve the same goal as a tag replacement attack. However, in the latter case, if the tag supports collision detection, such as NFC type 4 tags, then NFC readers might detect the collision and take proper counteractions [26].

2.7 Mifare DESFire EV1 Tags

NXP semiconductors[2] developed the Mifare DESFire EV1 (MF3 IC D41) tag [15] which relies on the ISO 14443 Type A specification for contactless communication and can be formatted as a NFC type 4 tag. The Mifare DESFire EV1 tag is a tag based on secure memory cards. Specifically, the MF3 IC D41 chip has a central processing unit (CPU) which contains an asynchronous CPU core and a crypto co-processor [15]. However, it *does not* support customer-defined code, but only a pre-defined set of commands. In fact, people often say "program an NDEF application in a DESFire EV1 tag", but this actually means "write an NDEF message to the DESFire EV1 tag" by using the *write* command defined by the DESFire EV1 tag specification.

A Mifare DESFire EV1 tag can have 2KB, 4KB, or 8KB of memory depending on its specific version and allows up to 28 different NDEF applications to be stored on it. Each tag has a master key that is used for authentication with readers. In addition, each NDEF application in the tag can have up to 14 different DES/3DES keys which are used to perform three-pass mutual authentication between the tag and its communicating reader. After the authentication is completed, the reader and the card calculate a session key to protect the communication channel between them. To be more specific, three levels of communication security are supported: plain data transfer, plain data transfer with DES/3DES cryptographic checksum, and DES/3DES encrypted data transfer [15].

The three-pass mutual authentication procedure between a DESFire EV1 tag and a reader is shown in Figure 2.10 [27].

Specifically, the authentication steps are as follows:

1. The NFC reader starts the authentication procedure by sending an Authenticate command with a key number as parameter of the command. If the key number is 0x00, the master key of the tag is used for authentication.

2. If the requested key number is not correct, then an error code is sent back to the reader. Otherwise, the tag generates an 8-byte

[2]http://www.nxp.com/

random number RndB, encrypts this number using DES/3DES with the selected key K, and sends the result back to the reader.

3. The reader runs a DES/3DES deciphering operation using the key K on the response from the tag to retrieve RndB. This RndB is then rotated left by 8 bits to result in RndB'. In addition, the reader itself generates an 8-byte random number RndA. This RndA is concatenated with RndB' and DES/3DES deciphered in CBC mode using the key K.

4. The tag performs a DES/3DES encryption on the received tokens. The tag can now verify the sent RndB' by comparing it with the RndB' obtained by rotating the original RndB left by 8 bits internally. If the verification succeeds, the tag rotates the RndA value to the left by 8 bits to gain RndA'. This RndA' is then DES/3DES enciphered using the key K and is sent back to the reader.

5. The reader runs a DES/3DES decryption on the response from the tag. The result is compared with the original RndA value internally remembered by the reader.

6. The tag sets the authentication state for the currently selected application.

Provided that the authentication was successful, a 16 byte session key is calculated by employing RndA and RndB:
session key = $RndA_{1sthalf} + RndB_{1sthalf} + RndA_{2ndhalf} + RndB_{2ndhalf}$

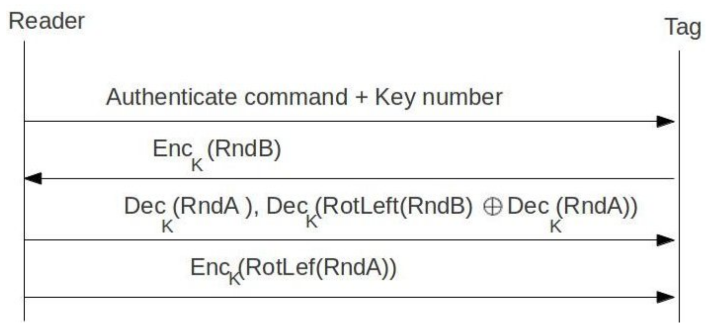

Reader
Tag
Authenticate command + Key number
Enc$_K$ (RndB)
Dec$_K$ (RndA), Dec$_K$(RotLeft(RndB) ⊕ Dec$_K$ (RndA))
Enc$_K$(RotLef(RndA))

Figure 2.10: DESFire mutual authentication procedure

2.8 Java Card Technology

This section discusses Java cards which have more capabilities and more intelligence than the tags described above.

2.8.1 Programmable Contactless Smart Cards

A programmable contactless smart card provides portability and built-in computational power. It has a single integrated circuit (IC) that contains a processor, memory, and I/O support. The card can be used for securing applications that use public-key or shared-key algorithms [28].

There are several similarities between a Mifare DESFire EV1 tag (as described in Section 2.7) and a programmable contactless smart card. Firstly, they are both passive devices which are powered by the electromagnetic field generated by a card reader and remain active only during the session with the reader. Secondly, both of them can be used for data storage and can host multiple applications on the same card. However, the types of applications stored in the two types of cards are different. Specifically, the DESFire EV1 tag is capable of executing a few pre-defined operations with limited functions. Therefore, it can store NDEF applications that do not require computations. On the contrary, a programmable contactless smart card can store and execute applications written in a high-level programing language, such as Java. This is because the microprocessor inside this type of card acts much the same as a CPU inside a personal computer and thus can be optimized for different user-defined applications that require dynamic computations

2.8.2 Java cards

A Java card is a programmable contactless smart card that is capable of running Java applets. This means that it has the flexibility and intelligence of a programmable contactless smart card. In addition, it supports a subset of the Java programming language with a runtime environment optimized for smart cards and other memory-constrained devices [28]. Thus, cards of this type also have the advantages of the Java programming language, such as security, robustness, and portability. The general architecture of a Java card is shown in Figure 2.11. Specifically, it consists of the following main components:

- Applets: They are Java applications which are compiled into bytecode instructions and installed in Java cards. They process incoming command requests and respond by sending data back to the reader.

- The Java Card Virtual Machine (JCVM): This defines a subset of the Java programming language and virtual machine specifications for smart card applications.
- The Java Card Runtime Environment (JCRE): This defines the Java Card runtime environment behavior, such as memory management, application management, security enforcement, and other runtime features.
- The Java Card API: This standardizes the set of core and extension Java card packages and classes for programming smart card applications.

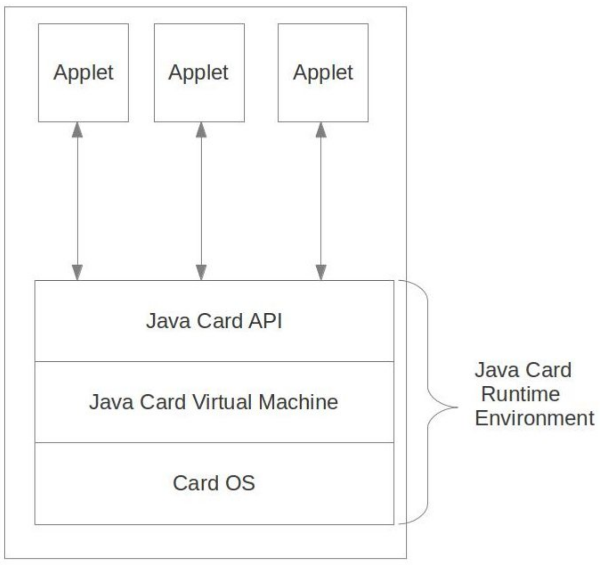


Figure 2.11: The Java card architecture

2.9 NFC Card Emulation

An NFC reader connected to an embedded computer (e.g. a micro-controller inside a vending machine) can emulate a tag. The emulated tag acts just like a real NFC tag, but it has more powerful capabilities and flexibility compared to a Java card or a DESFire EV1 tag. This is because the computer can have a software installed that controls the behavior of the emulated card. In addition, the computer has a clock and possibly Internet connectivity. An illustration of how an emulated card works is shown in Figure 2.12. Specifically, once the emulated card (reader A) receives messages from reader B, the emulated card forwards these messages to the software running in the connected computer. The software processes these messages and sends the result back to the emulated card so that the messages can be forwarded to reader B. Therefore, the computational capabilities of this card emulation are the capabilities of the connected computer and the software installed in the computer.

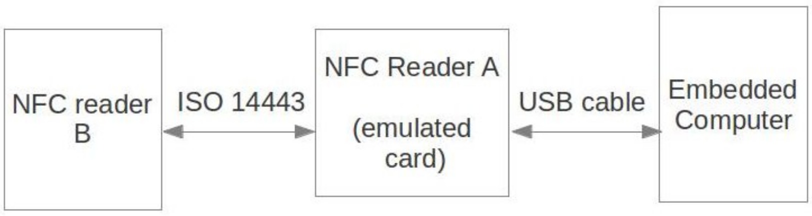


Figure 2.12: Card emulation mode

2.10 NFC Applications

The NFC Forum describes three key areas of NFC applications: service initiation, peer-to-peer, and payment and ticketing applications [29]. In the following three subsections, we describe some NFC applications presented in the literature and in industrial pilots. We categorize these applications based on the key areas that they belong to.

2.10.1 Service initiation

Various service initiation applications based on NFC tags and NFC-enabled phones have been proposed. These applications are built due to the fact that an NFC tag can store certain information, such as application-defined data or NDEF messages. In these applications, when a phone taps a tag, information stored in the tag is sent to the phone. Once the phone finishes processing this information, it presents the results to the user. For example, each exhibit in a museum can have an NFC tag that is placed close to the exhibit. By using an NFC-enabled phone to touch this tag, visitors can access more detailed information about the exhibit, such as photos, audio commentary, and video content. The information provided by the tag in this case is the exhibit identifier stored in the attached tag, for example a URL pointing to a web page that provides additional information about the object that was touched.

Smart posters The most common NFC application of this category is smart posters. A smart poster is a printed paper poster with an NFC tag attached to it. The tag can store some information, such as a URL for buying sports tickets or a timetable displayed at a bus stop [30]. However, as smart posters are deployed in public places that are vulnerable to security attacks, tags can be overwritten or even replaced by other tags. Therefore, Fischer [26] suggested that owners of the tags should consider write-protecting their tags unless the tags need to be re-written and then use an NDEF signature to provide integrity and authenticity for the tag content. However, the fact that anyone can place their own tags over the original tag is similar to someone putting his advertisement over the original poster (this operation is essentially for free in the case of normal posters). One way to protect smart posters would be to hide the tag in such a way that it would be visually detected that the tag was tampered with (e.g., removed). If another tag were to just be placed on top of it, collision detection would reveal that two tags are present [26].

Real-time reporting of security guards Incentive Lynx Security [5] uses a system based on NFC tags to provide real-time reporting of security guards and the locations they check when they carry out their patrols. Basically, an NFC tag is mounted on a checkpoint in a building to represent a location. By scanning a tag using an NFC-enabled phone with a dedicated application to retrieve the tag UID and then sending this UID to the application server over the Internet in real time, a security guard can prove that he is present at the checkpoint at the time the tag is tapped. More analysis of this use case will be presented in Section 3.1.1.

NFC-enabled restaurant A restaurant named “Pannu” in Oulu Finland offers meal ordering via NFC-enabled phones which have installed a dedicated application called *Restaurant Pannu* [30]. When a customer uses his NFC phone to tap on an NFC tag on a table in the restaurant, the *Restaurant Pannu* application is automatically run and displays a menu to the customer. When the customer finishes choosing his food, the application sends the customer’s order to the back-end system over the Internet. The back-end system will notify the restaurant’s payment system and kitchen about the order. A detailed analysis of this system will be discussed in Section 3.2.

NFC-enabled vending machine Mulliner [31] described how to apply NFC to SMS-enabled Selecta [32] vending machines. Specifically, a vending machine of this type can feature a tag containing an SMS message consisting of the machine ID and a short code to which the SMS message should be sent. A customer Alice who has a paybox[3] account uses her NFC-enabled phone to tap on the tag to retrieve the SMS message inside it. Once Alice sends the SMS message, the vending machine displays that it is ready to dispense an item. Alice selects her desired item and the amount of money is charged to her paybox account. This system will be analyzed in detail in Section 3.3.

NFC-enabled slide-show presentations Andersen and Karlsen [33] presented an NFC-based application that simplifies the user-computer interaction to set up a slide-show presentation. The scenario is as follows: Alice is going to have a presentation. Instead of manually connecting her laptop to a projector or downloading her presentation file from a file hosting service, she selects her file on her mobile phone, then taps her phone on an NFC tag placed at the presentation location. The information exchanged between the application and the NFC tag is not the presentation file, but actually is a URL that directs her phone's browser to a server running on a local presentation computer. This server is already connected to the projector. The application uploads the presentation file to the server specified by the URL. The server starts the presentation and a dedicated application on the phone can now be used to control moving slides forward, backward, or pause.

NFC-based pervasive games Garrido et al. [34] presented a use case of NFC-based pervasive games to encourage learning and to motivate students. In the game, players are given NFC-enabled phones that have a dedicated game application installed. In addition, there are hidden augmented objects placed at several places that students have to find. Each hidden object has an NFC tag attached to it. Once a student (Alice) finds an object, she uses her phone to tap on the tag on that object to get a bonus questionnaire. The tap event triggers the game application to connect to a game server over the Internet in order to download a questionnaire. Now, she has to answer the questions correctly in order to get points. Then the application provides players with instructions to reach the next destination in the game.

[3]http://www1.paybox.com/

NFC-based class attendance checking system Bueno-Delgado et al. [35] presented an NFC-based system to check student attendance in laboratory and theory lessons. This system includes three components: an NFC reader connected to a computer that is already available in every classroom, back-end systems such as servers to store user credentials, and NFC-enabled phones with a dedicated application. When a teacher (Alice) enters a class, she taps her phone on the reader to start the application running on her phone. She then logs in and indicates which group she wants to activate. This information is forwarded to the back-end server via the NFC reader and its connected computer. Once group activation is completed, students can enter the class. They tap their phones on the NFC reader to activate the application running on their own phones and then fill in their login name and password. This information is forwarded to the back-end server via the NFC reader and its connected computer. More analysis of this use case will be explained in Section 3.4.

NFC-based employee attendance checking system Patela et al. [36] presented an NFC-based mobile phone attendance checking system for employees. Basically, each employee is assigned a contactless smart card that stores his or her information, such as employee ID and name. When an employee (Alice) comes to work, she taps her card on an NFC reader at her workplace. Then her personal information is sent in real time to a mobile phone of the Human Resources (HR) department via the NFC reader. The HR employee looks up her employee ID in a database and then records Alice's attendance. These steps are repeated for all employees.

NFC-based meal delivery service In 2006 in Oulu Finland, an NFC-based meal delivery service was implemented and tested [37]. In this pilot, elderly clients ordered meals for home delivery services by tapping their NFC-enabled phones on the NFC tags placed on a daily menu. Later, the meals were delivered to the meal recipients by Oulu Logistic. In addition, each driver used a Nokia phone with a dedicated application to tap on an NFC tag at his workplace at the start of his delivery rounds to prove that he had started delivering food. Upon successful delivery at the recipient's home, the delivery man was required to tap his phone on the NFC tag glued to the home's door to prove that he had completed his task. He had to tap on the NFC tag at his workplace again when he finally completed his rounds.

2.10.2 Peer to peer

Among the three key areas of NFC applications, peer-to-peer applications seem to be the least popular [3]. In this mode, NFC is used to facilitate communication between two NFC-enabled devices. If the amount of data is small (less than 1KB, for example when sharing business cards), the data could be transmitted over NFC itself. Otherwise, NFC can be used to exchange parameters required to establish another wireless connection link, such as Bluetooth or Wi-Fi, to share information between the two devices [29]. For example, two mobile phone users wish to share photos with each other via a Bluetooth connection. They simply touch their two devices against each other to establish the Bluetooth paring and keying via NFC. After that, data can be shared over the Bluetooth connection which was just established [38].

NFC-enabled Bluetooth pairing Steffen et al. [39] proposed that NFC can simplify the Bluetooth pairing process between a mobile phone and a car's hands-free equipment. Specifically, a user taps her NFC-enabled phone on the car's NFC device. During this tap, the car transmits the necessary pairing information, such as its Bluetooth Address, PIN code, and its device name, via NFC to the mobile phone. After that, the Bluetooth interfaces of both the car and the mobile phone are activated, the pairing process is completed, and a secure link between two devices is established.

NFC-enabled health care service NFC could help to facilitate health-care services by providing user-friendly remote health monitoring, tracking, and control systems. Strommer et al. [40] observed that in off-line health monitoring, the heath parameters of a patient, such as heart-rate, weight, or blood pressure, can be measured off-line. After that, the data is transferred to the terminal, then analyzed and visualized for the user. However, this information is not utilized efficiently by many users, mostly because of the cumbersome data transfer from the measurement device to a personal computer or a mobile terminal. NFC could facilitate the transfer of data from the measurement device to a mobile terminal by having the patient tap their phone on the device.

2.10.3 Contactless payment and ticketing applications

Contactless payment and ticketing applications are currently the primary driver for the adoption of NFC on cell phones. In these system, the ticket or micro payment data is stored in a secure device, such as a contactless smart card or a mobile phone. When a user (Bob) wants to make a payment or to use his stored ticket, he presents his mobile phone or his card to a reader associated with his desired service. Usually, his card or his mobile phone and the reader use an application-specific protocol to process the payment. In the case of mobile phones, dedicated applications can be installed to load money or to buy tickets.

NFC in public transportation systems We have witnessed a great success of NFC in the area of contactless smart cards. For example, these cards have been widely used in some Asian countries and Scandinavian countries in public transportation systems. These systems are mostly based on proprietary solutions which use NFC tags to store credit (i.e. value) [41]. The Smart card Alliance has proposed an open payment ticketing system. In this system, each traveler has a travel account in a cloud, which is operated by a service provider (SP). The travel card does not store value but simply stores the user's identity and credentials. This identity and credentials are read by a reader at station gates and sent to a back-end server. The ticket identity and travel information are sent to a back-end server. Then, the back-end server calculates the fare based on the traveling distance and forwards the information to the SP for payment collection [41].

Google Wallet An example of mobile NFC payment is Google Wallet. In this system, the credit card or bank account information of a user is stored in Google's cloud. The secure element on the user's phone stores a virtual payment card identity. During payment, the user selects the payment card from a payment application on his phone. After that, the phone presents itself as a card to the point-of-sale (POS) terminal. Google then collects the payment from the user's credit card or bank account and makes the payment to the merchant [42].

NFC-enabled coupons Dominikus et al. [43] presented an NFC-enabled coupon application named mCoupon. This work is interesting because it proposed an application-specific security protocol for NFC tags and mobile phones. The authors proposed two security protocols to prevent an NFC-enabled coupon from being used multiple times, unauthorized generation, manipulation, and unauthorized copying by applying client authentication in the mCoupon system, as shown in Figure 2.13. Specifically, in both protocols, a passive NFC device plays the role of an issuer. In the simple protocol (Figure 2.13), a user touches the issuer with his mobile device to start the mCoupon application. After that, the application generates a challenge (R_M)

and sends it to the issuer. The issuer attaches some informative data (Data), e.g. about the type, issuing time, and validity range of the coupon; and encrypts the challenge and additional data using the secret key K_I . Then, it sends a valid mCoupon to the client's mobile device. The mCoupon consists of the issuer identifier (ID), the challenge, the additional informative data, and the encryption result, which is the response to the challenge. In the advanced protocol (Figure 2.13), a customer (Bob) uses his phone (M) to tap on the issuer to initiate the application. After that, the application sends a request to get a valid mCoupon containing the challenge R_M for the issuer (I). In this protocol, the issuer wants the customer's phone to authenticate it, so the issuer also sends a challenge (R_I) to the customer's phone. The phone signs this challenge by using its private key (PrK_M). It sends the signature and its identifier (ID_M) to the issuer. The issuer is not able to verify the signature, but uses the authentication data as input for an AES encryption. It attaches additional informative data (Data) and the authentication data to the challenge and encrypts this input using AES. As in the simple protocol, the valid mCoupon consists of the issuer ID, the client challenge, and the encryption result.

However, these two protocols are unrealistic for the current mass-produced passive NFC tags since these passive (i.e. batteryless) NFC tags cannot be programmed to implement application-specific cryptographic protocols and do not have a clock for freshness. Therefore, they cannot be used as an issuer. In addition, the advanced protocol has a potential security flaw in that the third message can be forwarded. However, this can be fixed if the issuer identifier (ID_I) is added to the signature in the third message.

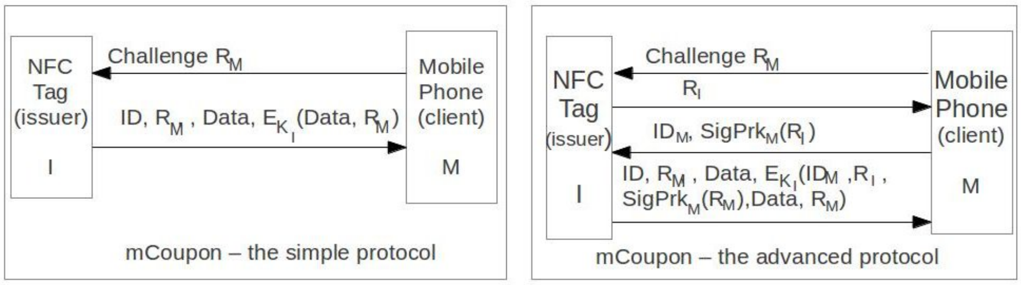

NFC Tag (issuer) I
Challenge R_M
ID, R_M , Data, E_{K_I}(Data, R_M)
Mobile Phone (client) M
mCoupon – the simple protocol
NFC Tag (issuer) I
Challenge R_M
R_I
ID_M, $SigPrk_M(R_I)$
ID, R_M , Data, E_{K_I}(ID_M ,R_I , $SigPrk_M(R_M)$,Data, R_M)
Mobile Phone (client) M
mCoupon – the advanced protocol

Figure 2.13: Procedures to issue an mCoupon

Chapter 3

Secure NFC-enabled Application Designs

In this chapter, we present our security designs for the four use cases which was briefly explained in Section 1.2, specifically the security guard monitoring system, the NFC-enabled restaurant menu, the NFC-enabled vending machine, and the student attendance monitoring system. These four applications were chosen because they have different security requirements and different target users. Hence, they provide different points of view regarding the security needed for NFC-enabled applications. Each application is presented in a separate section in which we review its current solutions, and potential security and usability problems in these solutions. Thereafter, we present our design goals, our system design, and analysis of the proposed solution.

3.1 NFC-enabled security guard monitoring system

In this section, we first describe some current solutions for the security guard monitoring use case that are available on the market. Thereafter, we present an analysis of the security requirements of a guard monitoring system, our security protocols to secure this system, and an analysis of our protocols.

3.1.1 Current solutions

Security guards have to make security rounds at non-regular intervals to inspect certain areas of a facility to make sure that there are no intruders and that no damage has occurred to the building or equipment. They also have to guarantee that gates and doors are locked during certain times, such as after business hours. Therefore, it is important to monitor the work attendance of security guards. To automate the monitoring process and to enable customers to check the time and the frequency of security guards' visits, there are two types of security guard monitoring systems on the market: NFC-enabled systems and non-NFC systems.

Non-NFC guard monitoring systems These systems employ rugged readers and checkpoint tags which are available in at least three forms: touch memory buttons, RFID tags, and barcodes [4]. Although these checkpoint tags are different in form, they have one property in common, i.e., each tag has its own unique identifier (UID). Therefore, each tag can represent a location that security guards have to visit during their patrols. To confirm a visit at a certain place, a security guard uses his rugged reader to scan the checkpoint tag bound to that place. During this scan, the reader records the tag UID and stores this UID along with the current time in its memory. At the completion of each round, the security guard uploads the data stored in the reader memory to a computer or a server.

There are several issues in these systems. Firstly, the reader does not prevent a dishonest security guard from changing the clock of his reader to his future patrol schedule, scanning a tag to have several records stored in the reader. When his reporting time comes, he can upload the record corresponding to the report time without being present at the checkpoints. Moreover, the dishonest security can tamper with the timestamp of the records stored in the reader's memory. Secondly, systems that use barcodes are easier to attack than systems that use touch memory buttons or RFID tags. This is because the barcodes can easily be copied. Thus, a security guard can copy all barcodes on his round, bring them home and produce the reports without going to work.

NFC-enabled guard monitoring systems There are NFC-enabled security guard monitoring systems on the market, such as Incentive Lynx Security (Section 2.10), inViu NFC-tracker [6], and NFC Patrol [7]. They are based upon installing NFC tags at checkpoints in the guarded areas. A security guard uses an NFC-enabled phone with a dedicated guard application to tap on each tag on his or her patrol round. During this tapping time (hereafter we call it a tap event), the tag UID is collected and sent to an application server in real time over the Internet. Some systems, such as Incentive Lynx Security, also employ the Global Positioning System (GPS) in the phone to keep track of the positions of the security guards. Although this solution adds more accuracy to the monitoring, it does not always work since GPS signals are not available inside buildings.

These systems are more advanced than the non-NFC guard monitoring systems presented above in that they provide real-time reports and do not require the security guard to carry one more piece of equipment. However, since the tag UID is fixed, it allows a dishonest security guard to adjust the clock on his phone, scan a tag several times to have several records stored locally in his phone, and upload each of them to the server when his reporting time comes. He could also manipulate the log file or generate fake reports if he knows the tag's UID. In addition, advanced equipment which is currently available to research laboratories (e.g. proxmark3 [44]) can be used to copy and spoof the tag UIDs. Therefore, the current NFC-enabled guard monitoring systems on the market do not guarantee that reports sent from the guard application can be trustworthy.

3.1.2 Attacker Model

In the NFC-enabled guard monitoring system, we consider the phone used by the security guards and the applications installed in the phone are untrusted while the tag and the back-end server are trusted. This is because, firstly, the phone may be lost or stolen. The 2011/12 Crime Survey for England and Wales reported that around 2% of mobile phone owners experienced a phone theft in 12 months [45]. Secondly, the phone may be used by a dishonest security guard, i.e. an inside attacker, who may want to generate presence confirmations without doing his or her actual rounds. Therefore, he or she may compromise the security of the phone and the applications installed in it for his or her purpose. For example, a malicious security guard can have root access to the phone, thus having complete control of the phone and its software stack [46]. This means that if we put secret information in the phone, such as a secret key, then it is possible that the security guard knows the key. Moreover, mobile malware has rapidly become a serious threat [46] and may tamper with the secret information and applications on the phone. In addition, it is not secure to store secret key in the phone since we do not have access to the secure element (SE) of the phone. Specifically, the dedicated SE embedded onto the phone during the manufacturing stage does not have the communication to the NFC controller in the phone and only proprietary protocols can be used so far [47]. What is more, some phones support a UICC-based SE [47], but they do not have a public API for accessing this SE [48].

3.1.3 Design Goals

Our proposed design aims to implement an effective and reliable security guard monitoring system. Specifically, it solves all the problems noted in the NFC-enabled security guard solutions explained in Section 3.1.1. In addition, our system facilitates transparency of guard operations and provides real-time as well as historical reporting capabilities. Moreover, it guarantees that the reports sent from the guard application are not possible to fake.

These goals can be achieved by a challenge-response protocol between NFC tags at the checkpoints and an application server. This leads to two possibilities: (i) using the three-pass mutual authentication provided by tags based on secure memory cards, such as DESFire EV1 tags (Section 2.7), or (ii) building a challenge-response protocol using programmable contactless smart cards, such as Java cards. Due to differences between the features of DESFire EV1 tags and Java cards, the requirements and design of the security guard application based on these two possibilities are different, as will be explained in Sections 3.1.5 and 3.1.6. However, the general structure of both system is the same as described in the next section.

3.1.4 The general system architecture

The general structure of both systems is the same and is depicted in Figure 3.1. Specifically, they include the following main components:

- Checkpoint tags: Checkpoint tags could be DESFire EV1 tags or Java cards. A checkpoint tag is attached to a checkpoint on a patrol round to represent a location.

- NFC-enabled phone with a dedicated application: NFC-enabled phones are assigned to security guards. A dedicated guard application needs to be installed in each phone before the system is used. To activate the guard application, a security guard uses his or her phone to tap on a checkpoint tag. After that, the application acts as a proxy to facilitate message exchanges between the checkpoint tag and a remote application server. We assume that there is a server-authenticated Transport Layer Security (TLS) [49] session between the guard application and the application server.

- Application server: The application server on the Internet is able to respond to requests sent from the guard application and processes these requests. It also keeps track of all the sessions that are on-going between it and the checkpoint tags.

- Database: A database is used to store shared keys between the application server and the checkpoint tags. In addition, the database stores information about all the sessions that have occurred between the server and the checkpoint tags.

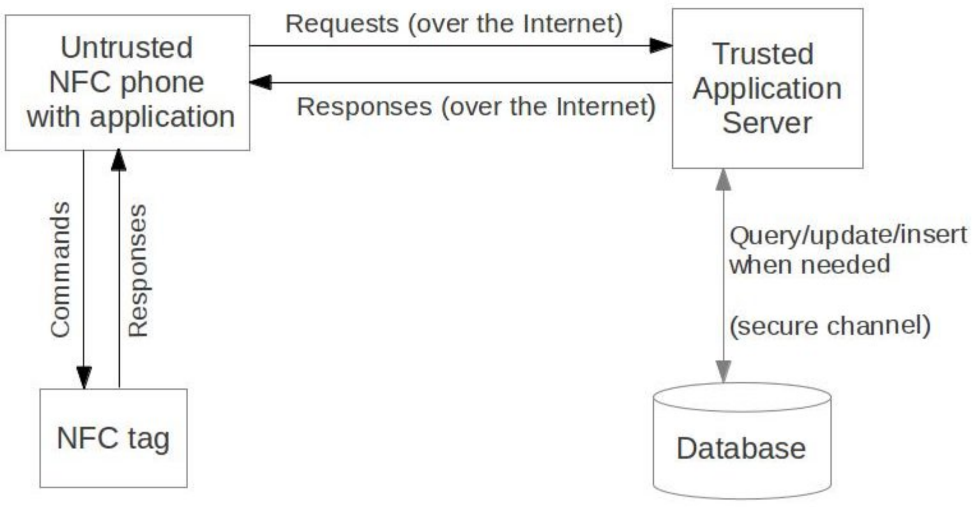

Untrusted NFC phone with application
Requests (over the Internet)
Responses (over the Internet)
Trusted Application Server
Commands
Responses
NFC tag
Query/update/insert when needed
(secure channel)
Database

Figure 3.1: Overall architecture of the security guard system

3.1.5 A secure guard monitoring system using Java cards

In this system, the specific type of the checkpoint tags is Java card. The Java card has an applet that keeps track of a sequence number and increases this sequence number by one when a phone taps on the card. In addition, the card has a shared key K (which is shared with the application server). The applet responds to commands (see below) sent from the guard application running in the guard's NFC-equipped phone. Moreover, it is able to log a number of the last events in case information about historical events needs to be retrieved from the card.

As regards the guard application on the phone, besides the general functions presented in Section 3.1.3, this application is also able to check for the availability of Internet connectivity before it attempts to send data to the application server. If an Internet connection is available, the data is sent instantly to the application server. Otherwise, the application stores this data locally until it senses that the connectivity status of the phone has changed and network is reachable.

3.1.5.1 Process

An illustration of the messages exchanged between the Java card at a checkpoint and the application server during a tap event is shown in Figure 3.2. Specifically, it comprises the following steps:

1. The guard application is automatically activated when the phone taps on the card. The phone receives the UID of the card once it finds the card.

2. The guard application checks if a network connection is available or not. If a connection to the server is available, then the application sends the card's UID to the server along with a request for a challenge. Otherwise, this step and the third step are skipped.

3. When the server receives a challenge request, it generates a random number R and sends this R back to the guard application. This challenge is also stored in the back-end database to keep track of the challenge-response session.

4. The guard application composes a challenge command which contains the phone identifier, IMSI and IMEI, and the current time on the phone. If the second and the third steps were done, then this command also carries the challenge R received in the third step. Now, the guard application sends this command to the Java card.

5. Once the card receives the challenge command, the applet on the card extracts the phone identifier, IMSI, IMEI, the time, and the challenge number R, if this value is present. It calculates a keyed hash (HMAC) MAC_K*([R], seq, card UID, phone ID, time, IMEI, IMSI, time)* using the shared key K. The applet writes this sequence number *seq* along with the challenge command and the MAC into its log file. Additionally, it returns the MAC value and the current value of sequence number to the guard application. Finally, it increments the sequence number stored in the card by one.

6. Once the guard application receives the response from the card, it checks for the availability of Internet connectivity. If an Internet connection is found, it forwards the response along with the card's UID to the application server. The server checks the MAC on the response

and updates the entry in the database that is associated with this challenge-response session. If an Internet connection is not available, the response from the card is locally stored in the local database of the guard application. A beep is locally generated to inform the guard that he or she can move the phone away from the card and continue his or her round. Once the phone finds an Internet connection, it immediately sends all the pending data to the server. The server checks the sequence number, the challenge R (if this value is present) and the MAC on these submissions. The server then updates its database accordingly.

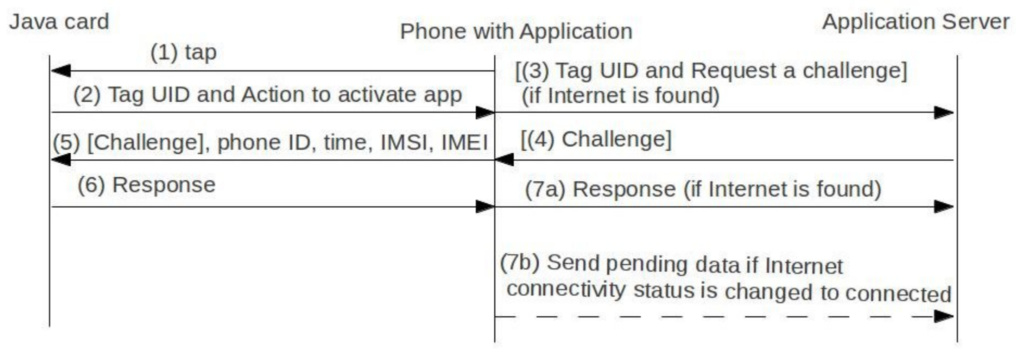

Java card
Phone with Application
Application Server
(1) tap
[(3) Tag UID and Request a challenge] (if Internet is found)
(2) Tag UID and Action to activate app
(5) [Challenge], phone ID, time, IMSI, IMEI
[(4) Challenge]
(6) Response
(7a) Response (if Internet is found)
(7b) Send pending data if Internet connectivity status is changed to connected

Figure 3.2: Security guard monitoring system using Java card

3.1.5.2 Protocol analysis

In this section, we present an analysis of the protocol described above to prove that it meets our stated goals. In addition, we explain the limitations remaining in our design.

Reliable security guard monitoring system A challenge-response session between the Java card and the server ensures that a tap event is fresh, since the challenge sent by the server is a random number. In addition, the phone identifier sent along with the challenge number helps to bind the identity of the phone to all these sessions. Replay attacks are prevented by using a HMAC [50] over all information that the card receives from the phone. By this, we make maximal information available for checking consistency of an authentication session [51].

Good usability The guard application automatically starts without any manual action or additional inputs from security guards. In addition, the sequence number that is maintained by each Java card and the application server helps to deal with intermittent Internet connections between the phone and the server. Let us assume, at time t_1, the phone has Internet connectivity, and a security guard uses this phone to scan a card. This means that, at the end of the session, the database at the server side has an entry for this challenge-response session which contains the time a challenge was sent to the card, the time a response from the card was returned, the challenge number, and the sequence number. At time t_2, another guard uses the phone but there is no Internet connectivity when it taps on the card. This second scan is not submitted to the server until the phone can establish an Internet connection. Once the phone is connected to the Internet, all pending submissions are sent to the server at time t_3. The server checks if the HMAC and the sequence number that are submitted by the guard application are correct or not and then updates its database. As part of this check, it can examine the time and sequence numbers of guard visits to a given tag. If the sequence numbers are not strictly increasing in time, it can generate an alert that there is a problem.

At time t_4, the phone has Internet connectivity, and another security guard uses this phone to scan the same card. The database at the server side has an entry for this scan. This means that, at time t_2, only the sequence number is used and the time sent in the response is from the phone, which is not trusted. Nevertheless, we have the upper and lower bounds on the time t_2 when the off-line session took place based on its previous and next sessions: $t_1 \leq t_2 \leq \min\{t_3, t_4\}$.

Historical reports The log in the Java card which stores a number of last sessions helps to provide historical information and to protect against data loss. The server can send a special command to the phone to retrieve the log data. This may help in situation where a phone that has some pending data to submit breaks before it gets Internet connectivity and the pending data is lost.

Further narrowing the time window The time window can be further narrowed down by using the following scheme: the phone can retrieve UID-independent challenges periodically from the server and use the last challenge that it receives from the server if there is no Internet connectivity at the time it taps on the tag. However, this requires a thorough investigation of the interval between these challenge requests if we are to prevent the battery drain problem on the phone.

Limitations The system described above has some limitations that should be understood. Let us imagine a guarded area that has no wireless Internet coverage. In this case, messages exchanged between the Java card and the application server occur as shown in Figure 3.3. If we assume that there is only one security guard who performs security checks and that this guard is dishonest, then this guard can change the clock on his phone and scan a Java card several times to have several records stored locally in the guard application. Later, the guard submits each of these records to the server when the reporting time comes. This is possible since the time supplied by the phone is not trusted and the guard can change it to any value he or she wants. However, if at least two guards do security checks alternatingly and at least one of them is honest, then the dishonest security guards cannot cheat. This is because each time the tag is tapped, the sequence number stored in it is automatically increased by one. To illustrate this, consider the case of two security guards Eve and Bob. These guards are assigned to do checking rounds every day at 8.00PM and 10.00PM respectively. Assume that the current sequence number is one. Eve, who is dishonest, is performing her check and wishes to skip her work tomorrow. She scans the Java card twice so that she has one record to report for now, and one for tomorrow. These two records have sequence numbers 1 and 2, respectively, and the sequence number remembered by the Java card is now 3. At 10.00PM, Bob, who is honest, does his check and submits to the server a record with sequence number 3. This means that the application server has records with sequence numbers 1 and 3, while sequence number 2 has not been submitted yet. From that, the administrator can learn that the sequence numbers and times of visits to this tag are not monotonically increasing. Thus the administrator knows that one of the guards has tampered with the checking process.

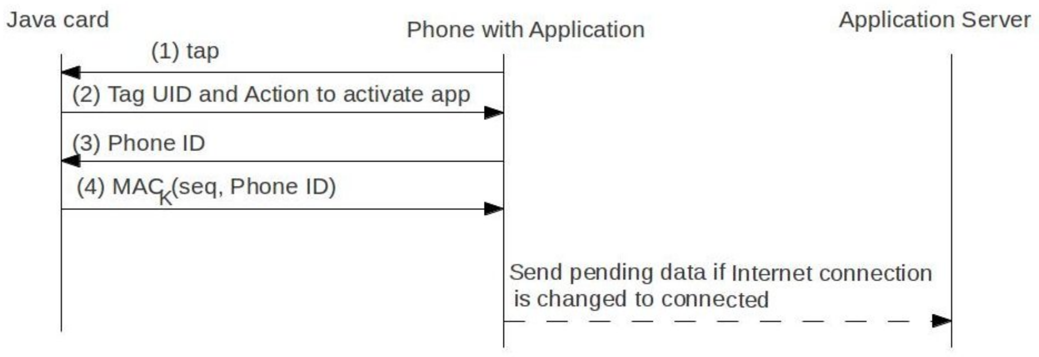

Java card
Phone with Application
Application Server
(1) tap
(2) Tag UID and Action to activate app
(3) Phone ID
(4) MAC_K(seq, Phone ID)
Send pending data if Internet connection is changed to connected

Figure 3.3: Security guard monitoring system using Java card (offline case)

Another limitation of the system is that it is vulnerable to the kind of relay attacks mentioned in Section 2.6. Specifically, the scenario is as depicted in Figure 3.4. There can be a person with a phone that does not need to have a security guard application installed. This phone has a reliable Internet connection to the security guard's phone which has a security guard application installed. All messages exchanged between the Java card and the application server are tunneled via both phones without being noticed by the server since the challenge-response session still takes place between the server and the card. However, this is not a significant problem with this use case since this still means that at least a person is present at the checkpoint at the checking time. It is possible that the person visiting this checkpoint does not do any of the observations or checks that the guard is supposed to do. However, this is just the same as the case that a non-enthusiastic security guard does not do any checks around the checkpoint after he or she scans the checkpoint tag to register his or her attendance.

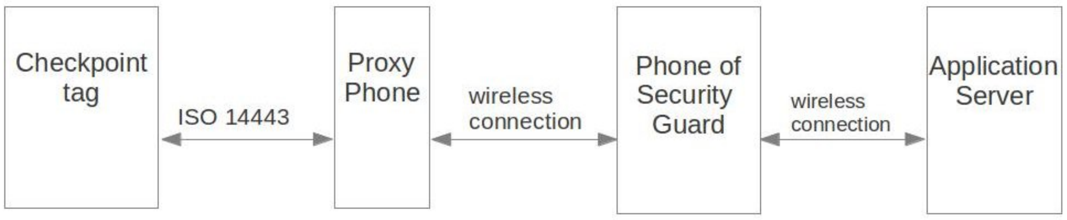


Figure 3.4: Relay attacks in the security guard use case

3.1.6 A secure system using DESFire EV1 tags

In this system, the NFC tags are DESFire EV1 tags, which cannot be programmed to implement arbitrary commands but have pre-defined commands for authentication, reading, and writing NDEF messages. Each DESFire EV1 tag has an NDEF application designed for the security guard system. The tag shares a key K with the application server. The system is designed based on the authentication protocol presented in Section 2.7.

3.1.6.1 Process

An illustration of the messages exchanged between a DESFire EV1 tag at a checkpoint and the application server during a tap event is shown in Figure 3.5. It is worth noting that this is the normal DESFire EV1 authentication protocol (Section 2.7) where the phone acts as the reader and the secret key is stored in the remote server. To be more specific, the process consists of the following steps:

1. The guard application on the phone is automatically activated when the phone taps on the tag. The application receives the tag's UID once it finds the tag. After that, the guard application sends a select-application command to the tag and it waits for the status response from the tag.

2. If the response from the tag is correct, then the application sends the authenticate command to the tag. The key number sent in this command is the key number associated with the NDEF application that is written in the DESFire EV1 tag.

3. The tag generates an 8-byte random number RndB and encrypts this number using the DES algorithm with the shared key K. The result is sent back to the guard application. When the guard application receives the encrypted value of RndB, it forwards this value along with the tag's UID to the server. Based on the tag's UID, the server looks up the corresponding secret key K and decrypts the challenge value from the tag to retrieve RndB. This RndB is then rotated left by 8 bits to obtain RndB'. After that, the server generates an 8-byte random number RndA and concatenates it with RndB'. This combination is then decrypted using the DES algorithm with the key K. The value $dec_K(RndA + RndB')$ is sent back to the guard application.

4. The guard application forwards $dec_K(RndA + RndB')$ to the tag. The tag can now verify the RndB' sent from the application by comparing it with the RndB' obtained by rotating the original RndB left by 8 bits internally. If this verification succeeds, the tag rotates the RndA value to the left by 8 bits to produce RndA'. This RndA' is then enciphered using DES with the key K and is sent back to the guard application on the phone.

5. The value $enc_K(RndA')$ is forwarded to the server. The server performs a DES decryption on that value. Finally, the result is compared with the server's internally rotated RndA'. The server can now store the result of the session into its database. The verification result (Success or Failure) is sent back to the guard application and a beep is locally generated to notify the security guard that he can remove the phone away from the tag.

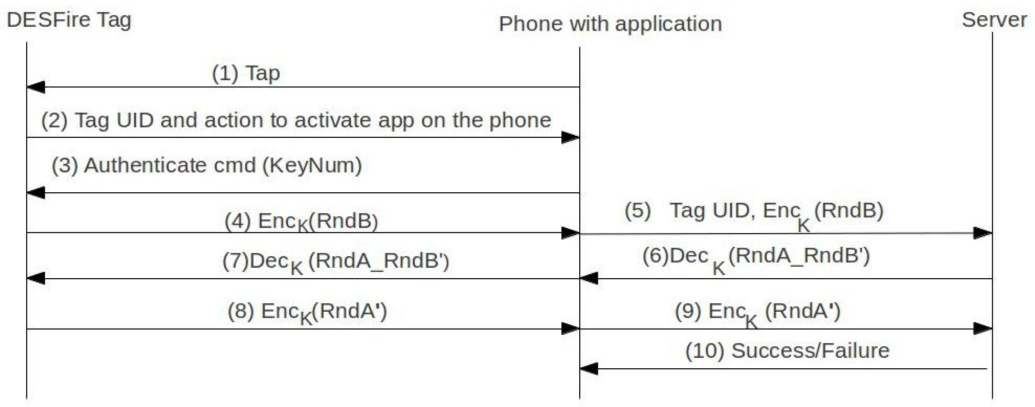


Figure 3.5: Security guard monitoring system using DESFire EV1 tags

3.1.6.2 Protocol analysis

In this section, we analyse our protocol to prove that it meets our design goals. In addition, we explain the limitations remaining in our design.

Reliable security guard monitoring system The challenge-response session between the tag and the server guarantees the freshness of a tap event. This is because RndA and RndB are random numbers, thus preventing the messages sent from the fourth step to the ninth step in Figure 3.5 from being replayed.

The application key is used instead of the master key of the tag As explained in Section 2.7, a DESFire tag can store multiple NDEF applications. This means that the checkpoint tag may store some other applications besides the NDEF guard application. Thus, in our design, the key of the NDEF application is used instead of the master key of the card. This is a good security practice because if the master key of the card is used and an attacker gains access to the database of the application server, he or she knows the master key. This allows the attacker to change the keys of all the applications in the tags and do whatever he or she wishes.

Good usability The guard application automatically starts without any manual action or additional inputs from security guards. In addition, based upon our testing (Section 4.2.3), we found that adding some delay to the message exchanges between the tag and the guard application on the phone did not affect the session. Therefore, we do not need to worry about the delay that may be added by network connections.

Limitations This design has some limitations that should be explained. Firstly, it requires the guard application to have an Internet connection during a tap event. Therefore, compared to the solution using Java cards presented in Section 3.1.5, this design is less flexible. Secondly, just as the case for the security guard application using Java cards, this design is also vulnerable to relay threats as presented in Section 3.1.5 (Figure 3.4). However, with the same reasons as presented in the system using Java cards, this is not a serious problem since this still means that at least a person is present at the checkpoint at the checking time. The possibility that the person visiting this checkpoint does not do any of checks at the checkpoint is just the same as the case that a non-enthusiastic security guard scans the checkpoint tag to register his or her attendance but does not do any observations.

3.2 NFC-enabled restaurant menu

In this section, we first review some current use cases of NFC in restaurants. Following this, we present an analysis of the security requirements of an NFC-enabled restaurant, our security protocol to secure this system, and an analysis of our proposed design.

3.2.1 Current solutions

In Section 2.10, the *Pannu* [30] restaurant was described to illustrate that NFC can be used in restaurants to facilitate the food ordering process, especially during the peak time, thus allowing restaurants to hire fewer waiters. What is more, assuming a person travels to a foreign country where he does not understand the language at all, this solution will allow him to intuitively point at and select food items. However, the *Pannu* restaurant requires customers to install a dedicated application on their phones in order to order a meal. This requirement might be a hindrance to the customers since installation of the application might be cumbersome and complicated. This results in poor user experience and system usability.

3.2.2 Design goals

Since restaurants are frequented by many different customers, the requirement that customers must install a dedicated application in their NFC-enabled phones to be able to order food is a disadvantage. Therefore, based on the fact that most NFC-enabled phones support web browsers, our design aims to make use of both the web browser and NFC features of the phone. Specifically, our proposed scenario is as follows: a customer (Alice) goes to a restaurant, she taps her NFC-enabled phone on a Java card attached on a table in the restaurant to retrieve a URL that directs her to the restaurant's web page. After studying the menu, she chooses her meal and then submits her order. The waiter will bring the food to her when the kitchen has finished preparing her order. As already implemented in many restaurants, the system helps the cooks to keep track of pending orders by having client computers in the kitchen that continuously display all the pending requests that have been submitted by customers. When an order is ready, the cooks delete the order from the list of pending orders.

The security problem that we want to solve in our system design for the scenario described above is the fact that the URL could be remembered or saved. As a consequence, a malicious person could stay at home and access the web page of the restaurant. He or she could then submit several food orders, but never come to the restaurant to pay and take away this food. Therefore, it is important that the system design has a mechanism to prevent fake food orders.

3.2.3 The system architecture

An illustration of the system is shown in Figure 3.6. Specifically, the system consists of the following components:

- Java cards: A Java card is glued onto each table in the restaurant. It contains a shared key K (which is shared with the server). In addition, it keeps track of a sequence number, which starts from zero and increases this sequence number by one when a phone taps on it.

- NFC phones: NFC phones are used by customers. The phones are not required to have any dedicated application installed but have to support a web browser. In addition, the phones need to have Internet connectivity.

- A web server: The web server hosts a web page that provides the menu and further information about the restaurant.

- A database: The database stores the secret keys that the Java cards on the tables share with the web server. The database also keeps track of all the orders that have been placed and processed.
- A websocket server: The websocket server connects to the database to retrieve new meal orders and pushes these new orders to the client computers in the kitchen.
- Client computers in the kitchen: There are client computers in the kitchen that continuously update the list of the food orders that are waiting to be served.

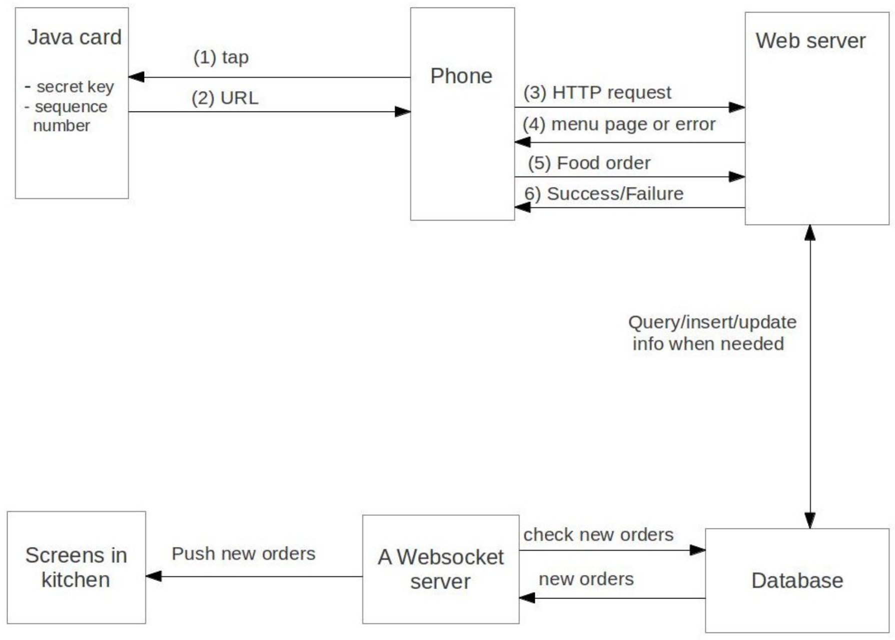


Figure 3.6: NFC Restaurant

3.2.4 Process

Details about the message exchanged between the components of the design presented in Figure 3.6 are:

1. Assume that the menu URL is *http://www.restaurant.com/mn.php*, and a Java card is glued to table 1. Both the Java card and the web server keep track of the sequence number *seq*, which initially starts at zero. They also share a secret key K.

2. When a customer (Alice) taps her phone on the Java card, the card calculates the keyed hash $MAC_K(table\ number, seq, card\ UID)$ using key K. This keyed hash and the table number ($t = 1$) are used to compose a URL of the form: *http://restaurant.com/mn.php?s=seq&hash=keyed-hash-value&t=1*. After that, the Java card encapsulates this URL into an NDEF message and sends it to the phone. It is important to note that NFC tags based on secure memory cards are not able to generate such dynamic URLs, while the programmable smart card can be programmed to do so. When the phone receives this NDEF message, its web browser initiates an HTTP request to the web server using the retrieved URL and waits for an HTTP response from the server.

3. When the server receives an HTTP request from a possible client, it checks if the sequence number and HMAC provided by the HTTP request are correct or not. If they are correct, a new entry corresponding to this session is inserted to the database. In addition, the server responds to the HTTP request with the menu page in an HTTP response. Otherwise, an error is returned in the HTTP response.

4. When the customer receives the menu page, she chooses her meals and submits her order.

5. Once the server receives a submission, it compares the sequence number and HMAC provided in the submission with the corresponding entry in the database to decide whether to accept the submission or not. The server must decide whether the submitted sequence number is valid and the keyed hash sequence is correct. Unfortunately, it is not the case that only the last sequence number is correct. For example, a group of people, let us say Alice and Bob, go to a table in the restaurant. Alice and Bob both tap their phones on the tag to access the menu. Since Alice tapped first, Alice's sequence number is smaller than Bob's. However, Bob finishes his submission before Alice and the sequence and HMAC in his submission are correct. In this case, it is important that if Alice's HMAC is correct, her submission should also be considered valid even though her sequence number is smaller than Bob's. In addition, some users may use their phones to tap on the card multiple times for no particular reason. Therefore, in the protocol, we define a sliding window. The size of the the window ($WINDOW$) of each table should be set to at least the number of seats at that table. When the submission of the last sequence number ($LAST\_SEQ$) is done, the submissions of all sequence numbers in the range $[LAST\_SEQ - WINDOW, LAST\_SEQ]$ within a predefined

amount of time (e.g. 20 minutes after the submission of the order with last sequence number) are considered valid.

6. The websocket server continuously checks the database to retrieve new meal orders. Whenever a new meal order is found, the websocket server pushes this order to the client computers in the kitchen in real time. When an order is completed, the cooks click on the order to delete it from the list.

3.2.5 Protocol analysis

In this section, we analyze our system design to prove that it meets our stated goals.

Good usability As our system is web based, customers are not required to have a dedicated application in their phones in order to be able to place meal orders. Therefore, the proposed design provide good system usability and user experience.

Protection from fake orders The use of HMAC values attached to the URL prevents dishonest people from staying home and placing orders at the restaurant. To be more specific, the card calculates the HMAC over the sequence number, the table number, and the card UID by using the shared key between the card and the application server. This HMAC is different each time the Java card is tapped due to the changes in the sequence number maintained by the card. This means that the HMAC value enables the card to return different URLs to the phone for different tap events. If the key used to calculate the HMAC is cryptographically strong, it is impossible for a malicious attacker to create a valid URL to generate fake submissions.

Error recovery Since we aim to provide customers with good service, we need an error recovery mechanism in the design. It is possible that a user (Bob) does not know that tapping on the tag is required for security purpose. This leads to situation where the user goes to the restaurant and taps on a card to order food, thus having the URL of the menu in the history of his phone browser. Next time when he goes to the restaurant, he thinks that the saved URL can be used, he opens it to order food. Since the sequence number is not correct, the web server should display a message suggesting him to speak to the waiters for confirmation of the order, or to tap on the card again, or ask if he wants to be redirected to a home-delivery ordering service.

Tag replacement or tag stacking It is possible that the card glued onto the table is replaced by a malicious person or another card is glued on top of the authentic one. In this case, customers can still call waiters for help. Therefore, in this use case, such attacks are not a serious problem.

3.3 NFC-enabled vending machines

In public places, such as schools or train stations, there are vending machines that dispense items, such as snacks and beverages, to customers after they have paid for the selected items. In addition to vending machines that support traditional payment methods, such as coins, printed notes, and bank cards, there are SMS-enabled vending machines. NFC can be used in these SMS-enabled vending machines to facilitate SMS message composition. Moreover, the addition of NFC can provide greater security to these vending machines. This section presents current solutions and how we can apply NFC to improve the usability and security of SMS-enabled vending machines.

3.3.1 Current solutions

The general architecture of an SMS-enabled vending machine is as shown in Figure 3.7 [52, 53]. To be more specific, to buy an item, a customer sends an SMS message with text consisting of the machine ID and payment amount to a short code of the beverage vendor. This message is routed through the mobile network to the mobile payment service. The mobile payment service parses the message as the price of the item and the machine ID. Then, it may send a confirmation request [54] or a receipt [52] to the customer. The cost of the purchase is added to the monthly phone bill or deducted from a pre-paid balance by the mobile phone operator. After that, the mobile payment provider communicates with the vending machine whose ID was in the SMS message to inform it about the credit value. The vending machine displays the customer's available credit (based upon the price that was sent), and the customer chooses his or her desired item. Thereafter, a receipt message is sent to the customer's phone.

As presented in Section 2.10, Mulliner [31] described how to apply NFC to SMS-enabled vending machines. Although this NFC solution facilitates SMS message composition, it does not make the vending machines more secure. Specifically, both the proposed NFC and non-NFC solutions exhibit two security problems. Firstly, a dishonest person could change the machine ID written on a vending machine A to the ID of a vending machine B. Then this person just waits in front of machine B to receive all the beverages when someone tries to buy a beverage from machine A. Secondly, since the short code could be saved in the phone or remembered by people, then a person who is not close to the vending machine could compose SMS messages to buy items from it. This results in problems, such as a dishonest person could borrow his friend's phone and use the phone to pay for a soft drink from a remote vending machine without his friend permission even if they are not at the machine.

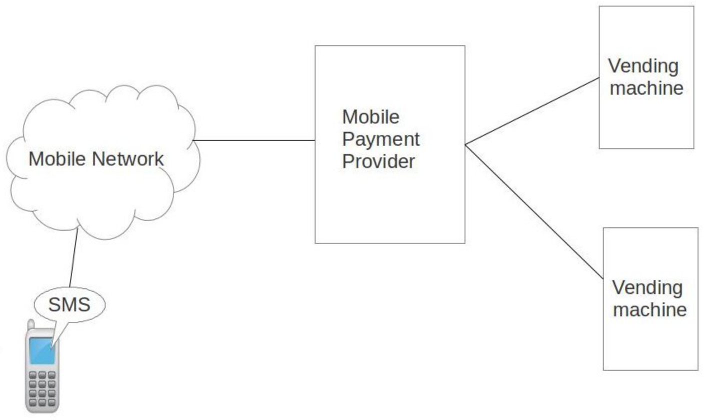


Figure 3.7: Mobile payment system for vending machines

3.3.2 Design goals

The aim of our proposed system is to overcome all the flaws indicated in Section 3.3.1. To be more specific, it prevents the following threats. Firstly, it prevents a person who is not close to a vending machine from buying items from that machine. Secondly, it prevents a malicious person from getting free items by changing the machine ID of a machine to the ID of another machine. In addition, our design does not require customers to install a dedicated application in order to buy items.

Our observation is that each vending machine already has a micro-controller which is connected to the Internet. Therefore, we can connect an NFC reader to this micro-controller to operate the NFC reader in card emulation mode or peer-to-peer mode. However, if we do not want to make change to the micro-controller (for example, making changes to the micro-controller might be difficult and costly due to its physical design) but we still wish to facilitate SMS composition and make the vending machine more secure than its current solutions, then we could use Java cards. The following subsections (Section 3.3.3 and 3.3.4) present designs for both cases: a security design for a vending machine that uses an NFC reader connected to the micro-controller inside the machine, and a secure design that uses Java cards.

3.3.3 A system using NFC reader

This section describes a security design for a vending machine that uses an NFC reader connected to the micro-controller inside the vending machine. The reader can run in card emulation mode or peer-to-peer mode.

3.3.3.1 The system architecture

The system consists of four main components as illustrated in Figure 3.8:

1. A vending machine: The vending machine is capable of payments via SMS messages.
2. An NFC reader connected to the micro-controller in the vending machine: This enables the reader to emulate a card or run in peer-to-peer mode.
3. A mobile payment provider: The mobile payment provider is connected to mobile phone network operators and to the vending machine via secure channels. This provider implements a billing service to deduct money from the customers' mobile balances.
4. NFC-enabled phones: These phones are used by customers.

3.3.3.2 Process

Details about messages exchanged between the different components of the system are depicted in Figure 3.8. Specifically, a customer (Bob) has to follow the following steps to buy to a soft drink from the vending machine:

1. He presses the button that corresponds to the his desired item. Since the vending machine has a list of items and their prices, the system can look up the price of the item that Bob chose. This press event is remembered by the micro controller in the vending machine.
2. He then taps his phone on the NFC reader attached to the micro controller. The micro controller will generate a random number. This random number along with the machine ID and the item price will be put in an NDEF record for an SMS message. This record is sent to

his mobile phone via the reader that runs in card emulation mode or peer-to-peer mode.

3. After he sends the SMS message and the message reaches the mobile payment provider, the mobile payment provider communicates with the vending machine whose ID number is specified in the SMS. After that, the mobile payment provider requests the micro-controller in that machine to return the latest random number and the time that the micro-controller generated this number. The returned value is compared with the random number provided in the SMS message that was sent by Bob. The mobile payment provider also checks if the message is received within a short period (e.g. less than 5 minutes) from the time when the random number is generated. If everything matches, then the mobile payment provider knows that the message is correct. Depending on the operator's policy, it may send back an SMS to the customer to ask him for confirmation (for example, by replying with a "YES" message). In addition, the mobile payment provider sends the credit value to the micro-controller. Now, the vending machine displays the credit value and Bob selects his desired item. If the price of the item is equal to or smaller than the credit value, then the vending machine dispenses the item.

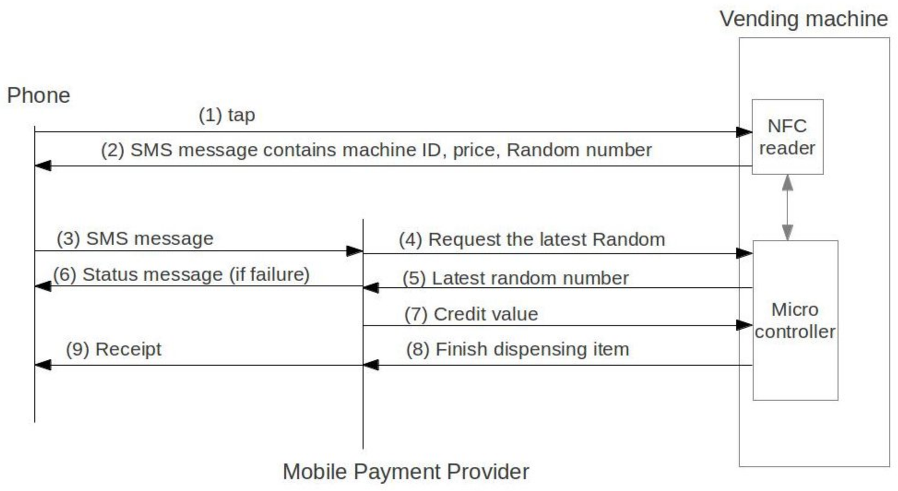

Vending machine
Phone
(1) tap
NFC reader
(2) SMS message contains machine ID, price, Random number
(3) SMS message
(4) Request the latest Random
(6) Status message (if failure)
(5) Latest random number
(7) Credit value
Micro controller
(9) Receipt
(8) Finish dispensing item
Mobile Payment Provider

Figure 3.8: NFC-enabled SMS Vending machine

3.3.3.3 Protocol analysis

In this section, we analyse our protocol to prove that it meets our stated goals. In addition, we explain the limitations that remain in our design.

Freshness for SMS messages A random number added to each SMS message provides a proof of freshness for the SMS message. This occurs because different SMS messages have different random numbers, and hence an attacker cannot guess the next random number and manually compose a valid SMS message to buy an item from the a possibly remote vending machine.

A malicious person cannot change the ID of a vending machine This is because all the information needed to buy an item is sent from the NFC reader. This reader is controlled by the micro-controller inside the vending machine. Thus, it is impossible for a malicious person to change the ID of a machine unless he has access to the micro-controller of the machine.

Another way to provide freshness for SMS messages It is worth mentioning that, as the micro-controller already supports a built-in clock, rather than using a random number in the message, we could use an HMAC of the tapping time. This provides the same security level as the above design. This is possible because HMAC values of different values of time are different, hence also adding a proof of freshness to SMS messages [50].

Limitations A limitation of our solution is that it does not prevent relay attacks: a person Charlie is far from the machine. He asks his friend Bob who has an NFC-enabled phone to go close to the machine. Bob taps his phone on the machine to get the SMS message and forwards that message to Charlie, or sends the random number to Charlie and Charlie will compose the SMS message himself. By doing this, Charlie can buy the beverage without being close to the machine. Thus, to be precise, our protocol guarantees that the buyer or the buyer's agent is near the vending machine.

3.3.4 A system using Java cards

This section presents a solution for SMS-enabled vending machines that uses Java cards. This design does not require any changes in the physical design of the micro-controller. Instead, we only need to upgrade the software in the micro-controller.

3.3.4.1 The system architecture

In this design, the system consists of four main components (as shown in Figure 3.9):

1. A vending machine: The vending machine is capable of payments via SMS messages.
2. Java cards: Java cards are glued to the vending machine. Each card maintains a sequence number and a secret key K, which is shared with the micro-controller inside the vending machine. It is worth noting that a Java card does not have interfaces to communicate with other devices, except through NFC readers, and thus a card cannot know which item the user has selected. Therefore, in this case, a separate card represents each type of item that is sold by the vending machine.
3. A mobile payment provider: The mobile payment provider is connected to mobile phone network operators and to the vending machine via secure channels. This provider implements a billing service to deduct money from the customers' mobile balances.
4. NFC-enabled phones: These phones are used by customers.

3.3.4.2 Process

Details about messages exchanged between the different components of the system are depicted in Figure 3.9. To be more specific, a customer (Alice) has to follow the steps described below in order to buy to an item from the vending machine:

1. Alice taps her phone on the Java card corresponding to her desired item.
2. When the card is tapped, it calculates a HMAC over the current sequence number, the machine ID, the Java card's UID, and the credit value using the key K: MAC_K *(price, seq, machine ID, card UID).* After that, the card composes an SMS message consisting of the machine ID, the item's price, the sequence number *seq*, the card UID, and the MAC value, wraps this SMS message in an NDEF message and sends it to the phone.
3. Alice sends the message using her phone. When the message reaches the mobile payment provider, the payment provider queries the micro-controller of the vending machine whose ID is sent in the SMS message

to check if the sequence number and the MAC provided by the SMS message are valid. It is worth noting that, although the sequence numbers maintained by the Java card and by the micro-controller initially start at the same value, they are not synchronized since there are cases where a customer taps his phone on the card to receive the SMS message but never sends this message. Therefore, a sequence number in an SMS message is considered valid by the micro-controller if it is greater than or equal to the current sequence number known by the micro-controller. If so, the micro-controller computes an HMAC over the machine ID, the sequence number, the card UID, and the credit value provided by the message and then compares with the HMAC in SMS message. If the message passes all these checks, Alice is allowed to continue to proceed to buy the item.

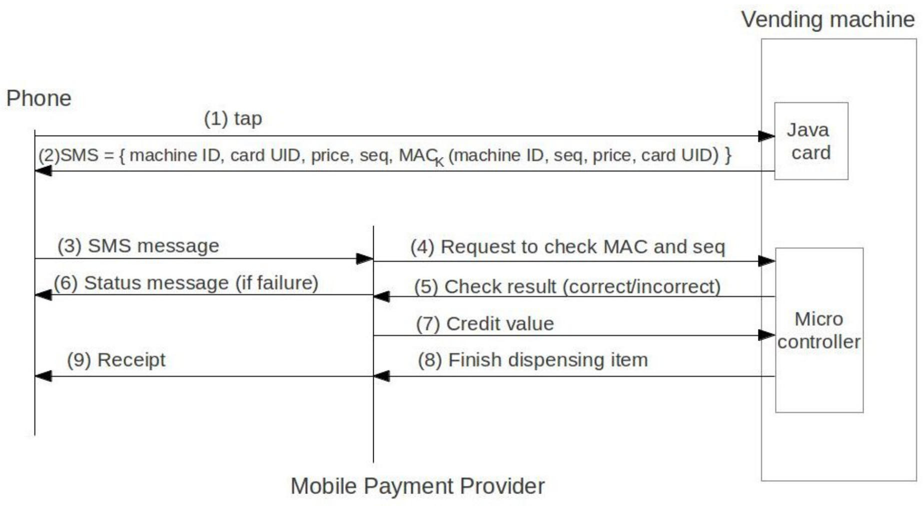

Vending machine
Phone
(1) tap
Java card
(2)SMS = { machine ID, card UID, price, seq, MAC_K (machine ID, seq, price, card UID) }
(3) SMS message
(4) Request to check MAC and seq
(6) Status message (if failure)
(5) Check result (correct/incorrect)
(7) Credit value
Micro controller
(9) Receipt
(8) Finish dispensing item
Mobile Payment Provider

Figure 3.9: NFC-enabled SMS Vending machine using Java cards

3.3.4.3 Protocol analysis

This section presents an analysis of our proposed design.

Freshness for SMS messages The HMAC value of the sequence number added to each SMS message provides a proof of freshness for the SMS message. This occurs because different SMS messages have different HMAC values [50], and hence an attacker who does not have the secret key cannot calculate an HMAC and compose a valid SMS message to buy an item from a possibly remote vending machine.

Limitations A limitation in our solution is that it does not prevent relay attacks as already described in the system using NFC readers (Section 3.3.3). Therefore, to be precise, our protocol ensures that the buyer or the buyer's agent is near the vending machine. In addition, it is important to keep in mind that, compared to the card emulation solution, Java cards are more vulnerable to card replacement attacks. This means that a dishonest person could switch Java cards between two machines. Then he just waits in front of one machine to receive all the beverages when someone tries to buy beverages from the other machine. Therefore, these Java cards should be glued or placed in such a way that it is difficult to replace them. For example, we could place these tags behind a plexiglass window so that it is difficult for a malicious party to tamper with the cards.

3.4 School attendance monitoring system

In this section, we first review some current solutions for the school attendance monitoring use case. Following this, we present an analysis of the security requirements, our security protocol to secure this system, and an analysis of our proposed design.

3.4.1 Current solutions

The traditional way to check student attendance in a class is a roll call: the teacher has a list of students registered for his course and he calls, one by one, the names of the students in the classroom. Alternatively, the teacher circulates a piece of paper within the classroom and students write their attendance directly on the paper. In the latter case, some students can write the names of other students who are not presented in the class.

As explained in Section 2.10, Bueno-Delgado et al. [35] presented an NFC-based system to check the attendance of students in laboratory and theory lessons. However, this solution requires students to tap their phones on an NFC reader to activate the application and then continue to keep their phones within the reader's range while entering their login information to the application. This requirement results in poor usability since it is difficult to keep the phone in the reader's range while entering information into the phone. It can happen that the connection between the student's phone and the reader is torn down in the middle of the session (for example, due to the movement of the phone while typing on it). As a consequence, a second tap is required. In addition, manually entering login names and passwords to the application is error prone and time consuming.

The attendance checking system for employees presented by Patela et al. [36] (see Section 2.10) is not efficient because it requires a lot of manual work. There is no point in sending the employee information to the HR representative's mobile phone. Instead, during each tap event, the information can be sent directly to a back-end server via the NFC reader. Afterwards, the server uses this information to look up the employee in order to do further processing. HR representatives can query the back-end server for information whenever they want.

3.4.2 Design goals

The main goal of our proposed system is to achieve an effective and reliable attendance checking system. Firstly, this system checks the attendance of both teachers and students. It ensures that it is not possible for teachers and students to report their attendance when they are not present in class. Secondly, it is fast and does not require manual inputs from the students. Thirdly, the system provides real-time reporting capabilities.

3.4.3 The system architecture

The system architecture is shown in Figure 3.10. To be more specific, the system consists of five main components:

1. Location tags: A DESFire EV1 tag is glued to the door of the classroom.

2. Each teacher and student has a DESFire EV1 tag. This tag stores a URL which includes the ID of the tag owner, such as his or her employee ID or his or her student ID.

3. An NFC-enabled phone: This phone is used by the teacher. The phone has an application installed, which is similar to the application presented in Section 3.1.6. In addition, we assume that the phone always has Internet connectivity during the time it taps on an NFC tag.

4. A database: This database stores the student ID and credentials, list of courses, list of students enrolled in each course, and the name and ID of the teacher who is responsible for the course. Moreover, it stores

the shared keys between the location tags and the server. In addition, it stores information about all the sessions that have occurred between the server and the location tags.

5. A server: The server has a connection to the database and processes HTTP requests generated when a student or a teacher taps his tag on the teacher's phone.

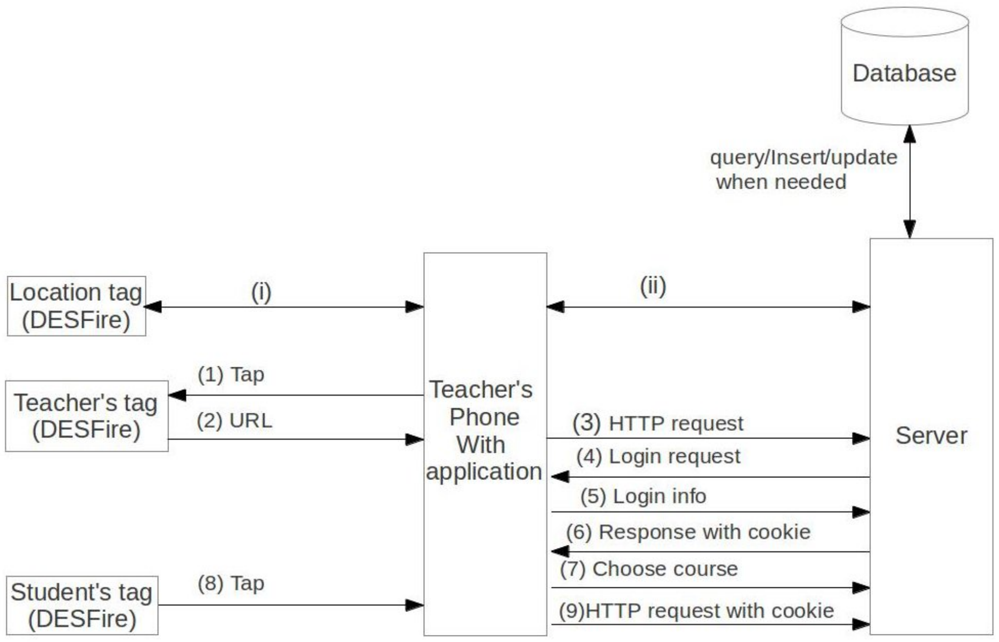


Figure 3.10: A school attendance monitoring system

3.4.4 Process

Details about the messages exchanged between the different components of the design presented in Figure 3.10 are the followings:

1. The teacher (Bob) proves that he is present at his scheduled time. This is done as follows: each teacher is equipped with an NFC-enabled phone with a dedicated application. In front of the door of each classroom, there is a location tag, which the teacher taps on with her NFC phone to prove that he has been present at the classroom. The design, processing, and messages exchanged between the location tag and the web server are the same as the system presented in Section 3.1.6.

2. The teacher taps his phone on the tag to get the URL. The browser of his phone sends an HTTP request to the server. The server requests the teacher to log in before allowing him to use the system. When this log-in is completed successfully, the server sends back an HTTP response, which contains a session cookie. This cookie has the expiration time set to the scheduled duration of the class.

3. When a student (Alice) enters the classroom, she taps her tag on the teacher's phone. Based on the URL stored in this tag, an HTTP request with the session cookie are sent to the web server to register the attendance of this student in the system. In this way, students do not need to do anything other than tap their tags on the teacher's phone. Also only the teacher needs to have a compatible phone.

3.4.5 Protocol analysis

In this section, we analyse our protocol to prove that it provides good usability and reliability. In addition, we explain the limitations that remain in our design.

Good usability and reliability Firstly, the system guarantees that a teacher must be present at the classroom in order to register his attendance with the same reasons as presented in Section 3.1.6. Secondly, it does not require students to have an NFC phone to be able to register their attendance, but rather each student has to have a student tag. In addition, it does not require additional actions from students after they tap their tag on the teacher's phone; thus the attendance registration process is fast and convenient. Thirdly, since every HTTP request sent to the server to register a student attendance has the session cookie attached, and this cookie expires after a preset time, a student cannot create a valid attendance registration unless he comes to the class on time.

Limitations There are some weaknesses remaining in this protocol that should be explained. Firstly, the design poses a transitive relation between different components as shown in Figure 3.11. Specifically, the teacher and his phone proves his attendance by tapping on the location tag on the door of the class room. Thereafter, the students tap their ID tags on the teacher's phone to prove their attendance. This means that if one component in the chain of trust is broken, then the security of the system fails. Secondly, a person and his tag are not fixed to each other. For example, a student can give his ID tag to his friend to help him to register for his attendance.

This problem can be limited if the ID tag binds more important personal information of its owner, such as access control to the university computer system, so that the owner may not want to give it to other people.

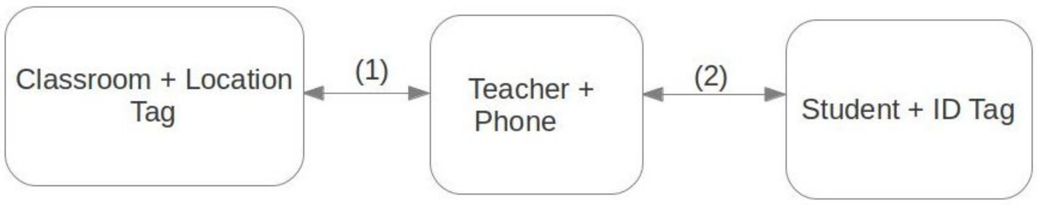


Figure 3.11: Transitive attendance verification

Chapter 4

Implementation

This chapter describes the implementations of two of the secure NFC-enabled systems presented in Chapter 3: the NFC-enabled restaurant menu and the NFC-enabled security guard monitoring system. These two systems were selected from the four use cases because they have different requirements and use different types of tags: a system uses DESFire EV1 tags, a system uses Java cards which requires a dedicated application on the phone, and a system that uses Java cards but uses web browsers in the phone instead of a dedicated application.

4.1 Devices and Programming Environments

All applications were implemented and tested in a Google Nexus S phone which runs Android version 4.1.2. As for all Android phones, this phone automatically looks for NFC tags when its NFC function is enabled and its screen is unlocked. When the phone finds an NFC tag close to it, it starts an NDEF detection procedure, as presented in Section 2.4. Once the phone finds an NDEF message in a tag, it encapsulates the message into an intent which is then used to locate applications that are interested in the type of data stored in the message [55]. If more than one application can handle the scanned data, an Activity Chooser is displayed on the phone so that users can choose the appropriate application.

In all proof-of-concept prototypes, we did not work on real Java cards. Instead, we emulated the functionality of Java cards by using a reader connected to a computer. The NFC reader used in this thesis is the ACR122U NFC reader [56]. We first connected the reader to a 64-bit x86 Linux machine which has the libnfc [57] library version 1.4.2 installed. This libnfc library is an open source library written in the C programming language to support research on the hardware and protocol levels of NFC. The purpose of the libnfc library is to provide developers with a way to work with NFC hardware at a higher level of abstraction, but with complete coverage of low-level PN53x [58] chipset commands. It supports both ISO 14443 and ISO 18092 standards; and can be used to transform an NFC device into an active or a passive device. The program to emulate an NFC reader as a Java card was written in the standard C99 programming language. The code for the emulated card for the security guard monitoring use case is provided in Appendix A.1.5. The code for the Java card emulation in the restaurant use case is provided in Appendix A.2.1. When these two card emulation programs were successfully implemented, we attached the reader to a Raspberry Pi [59] (RP) model B which uses the Debian operating system and compiled the code on this platform by using its default GNU compiler collection (GCC).

The DESFire tags used in this thesis was the Mifare DESFire EV1 with 4KB of memory. This card was formatted to be used as an NFC type 4 tag. The code used to format the tag and write an NDEF message to the tag is described in Appendix A.1.4.

4.2 NFC-enabled guard monitoring systems

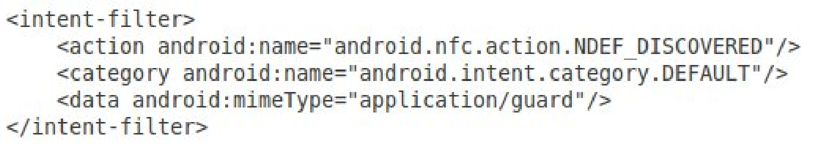
Since all of our systems aim to automate the work or security guards as much as possible, we attempt to avoid the Activity Chooser in the phone from appearing. To do so, in each system, instead of using well-known MIME types (TNF = 0x01), we define our own MIME type (TNF = 0x02) (Section 2.3.1). Each application registers an Intent filter to handle data of the application-specific MIME type defined by us.

4.2.1 Guard monitoring system using Java cards

Application server The application server for this security guard use case was implemented in a 64-bit x86 Linux machine by using the Jersey[1] library for building RESTful web services. The server is able to generate random numbers by using the *SecureRandom* class supported by Java. In addition, it handles HTTP requests sent from the phone application. It is connected to a MySQL database that stores shared keys (in this case shared between the server and Java cards) and information about all the sessions between the server and the Java cards. Details about the code are provided in Appendix A.1.1.

[1]https://jersey.java.net/

The security guard application on the phone Details about the code of this guard application are shown in Appendix A.1.2. The guard application is automatically started when the phone finds an NDEF message of MIME type *application/guard* stored in a Java card close to the phone. This is done by an intent filter defined as shown in Figure 4.1.



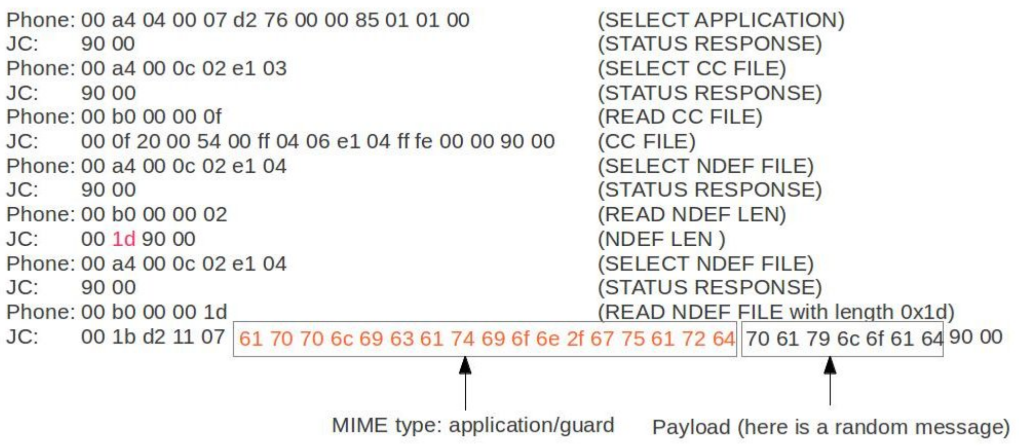
```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="application/guard"/>
</intent-filter>
```

Figure 4.1: Intent filter

The guard application communicates with the application server by using HTTP GET requests. As regards the communication between the Java card and the guard application, an illustration of commands sent between the phone and the Java card until the guard application is activated is shown in Figure 4.2 (extracted from our running code). After the guard application in the phone is activated, the communication between the Java card, the guard application, and the application server is as described in the protocol design (Section 3.1.5). Basically, the guard application uses the HTTP GET method to request the application server for a challenge number, if it finds that the application server is reachable. Thereafter, it composes a challenge command and sends this command to the Java card. In this implementation, we defined the challenge command of form: 0x11 [number of values] [length of each value] [values]. In this prototype implementation, the Java card and the server use the SHA1 algorithm [60] to calculate the HMAC.

To store pending records when the application does not have Internet connectivity, we implemented an SQLite database [61] inside the application. In addition, the application has a connectivity-change receiver that is able to sense changes in the Internet connectivity status of the phone. This receiver extends the *BroadcastReceiver* class provided by Java, which always listens to broadcast events sent by Android. Therefore, the application is able to know that the network connectivity status has changed and that the network is now reachable. Thus, the application will start to send all data stored in its SQLite database.

The emulated Java card The code for the emulated card in this use case is provided in Appendix A.1.5. When the phone taps on the emulated card, the card responds to the NDEF detection procedure of Android phones (as shown in Figure 4.2). Specifically, the NDEF message that the card sends to the phone during this procedure has the MIME type *application/guard*; thus the phone knows that it has to activate our guard application. After that, it responds to the challenge command sent from the guard application. In addition, before the Java card responds to a challenge command sent from the guard application, it records all of the information about this session into its log file. Then, the card increases the sequence number stored in it by one. In this implementation, the log file in the Java card stores the last ten sessions between the card and the application server.



```
Phone: 00 a4 04 00 07 d2 76 00 00 85 01 01 00          (SELECT APPLICATION)
JC:    90 00                                             (STATUS RESPONSE)
Phone: 00 a4 00 0c 02 e1 03                              (SELECT CC FILE)
JC:    90 00                                             (STATUS RESPONSE)
Phone: 00 b0 00 00 0f                                    (READ CC FILE)
JC:    00 0f 20 00 54 00 ff 04 06 e1 04 ff fe 00 00 90 00 (CC FILE)
Phone: 00 a4 00 0c 02 e1 04                              (SELECT NDEF FILE)
JC:    90 00                                             (STATUS RESPONSE)
Phone: 00 b0 00 00 02                                    (READ NDEF LEN)
JC:    00 1d 90 00                                       (NDEF LEN )
Phone: 00 a4 00 0c 02 e1 04                              (SELECT NDEF FILE)
JC:    90 00                                             (STATUS RESPONSE)
Phone: 00 b0 00 00 1d                                    (READ NDEF FILE with length 0x1d)
JC:    00 1b d2 11 07 61 70 70 6c 69 63 61 74 69 6f 6e 2f 67 75 61 72 64 70 61 79 6c 6f 61 64 90 00
```

MIME type: application/guard
Payload (here is a random message)

Figure 4.2: Activate application

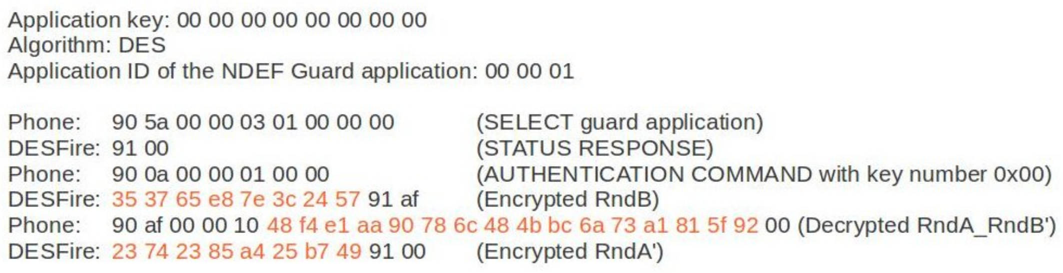
Administrative functions There are two additional functions for administrators provided by this guard application. Firstly, it has a button to view all pending data stored locally in the application. Secondly, the administrator can tap a phone with the guard application installed on a Java card to get this application run, then the administrator can press a button labelled *View Log* to view the last ten records stored in the Java card.

4.2.2 Guard monitoring system using DESFire tags

DESFire tag Our DESFire tag stores an NDEF application which contains an NDEF message with type *application/guardDESFire*. The key that the NDEF application uses to do authentication with the server is key number 0x00 on the DESFire tag.

The guard application The guard application in the phone registers an intent filter to handle NFC tags with the MIME type *application/guard-*

DESFire. Compared to the guard application presented in Section 4.2.1, this application is much simpler; it does not have a connectivity-change receiver nor a database inside the application. The only function of this application is to relay messages between the tag and the server. The guard application uses the HTTP GET method to send requests to the application server. In addition, it uses commands defined by the DESFire specification to communicate with the DESFire tag. Figure 4.3 is an example of a successful session between the DESFire tag and the application server. This example is extracted from our running code (the messages exchanged between the phone and the application server are not shown in this figure).



```
Application key: 00 00 00 00 00 00 00 00
Algorithm: DES
Application ID of the NDEF Guard application: 00 00 01

Phone:    90 5a 00 00 03 01 00 00 00          (SELECT guard application)
DESFire:  91 00                               (STATUS RESPONSE)
Phone:    90 0a 00 00 01 00 00                (AUTHENTICATION COMMAND with key number 0x00)
DESFire:  35 37 65 e8 7e 3c 24 57 91 af       (Encrypted RndB)
Phone:    90 af 00 00 10 48 f4 e1 aa 90 78 6c 48 4b bc 6a 73 a1 81 5f 92 00 (Decrypted RndA_RndB')
DESFire:  23 74 23 85 a4 25 b7 49 91 00       (Encrypted RndA')
```

Figure 4.3: DESFire security guard message flow

The application server The application server was implemented in a 64-bit x86 Linux machine by using the Jersey library for building RESTful web services. The server is connected to a MySQL database that stores shared keys between the server and DESFire EV1 tags and information about all the sessions between the server and the DESFire EV1 tags. Details about the code are provided in Appendix A.1.1. The server is able to generate random numbers by using the *SecureRandom* class supported by Java. In addition, it handles HTTP GET requests sent from the phone application. Details about the code are shown in Appendix A.1.3.

4.2.3 Effect of network latency

In order to measure the amount of network latency that the DESFire EV1 mutual authentication session can tolerate, we configured the Android phone and the application server to use the same network of our department. In addition, we delayed the messages that were sent from the server to the tag in the authentication session. Specifically, when the guard application received the value $dec_K(RndA{+}RndB')$ from the application server (the sixth message in Figure 3.5), instead of instantly forwarding this value to the tag, we set the application to sleep for a period equaling to the delay that we wanted to add, as shown in Figure 4.4. The piece of code to make the application sleep for predefined *DELAY* milliseconds: *SystemClock.sleep(DELAY).*

We run the experiment with different DELAY values, specifically 30 seconds, 60 seconds, and 120 seconds. For each value, we run the experiment for 5 times. We observed that for all the experiments, the delay did not break the session.

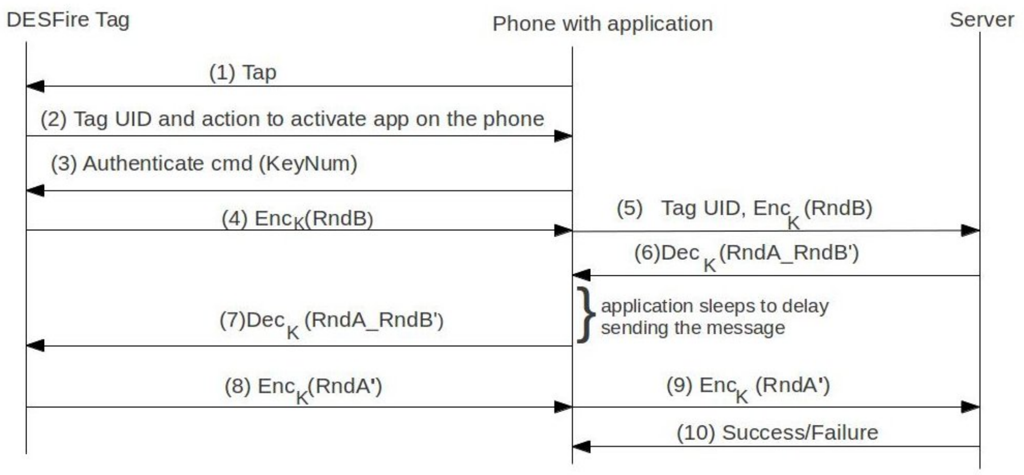


Figure 4.4: Experiment to measure the network latency

4.3 NFC-enabled restaurant menu

The NFC-enabled restaurant use case does not require a dedicated application installed in the phone. Instead, it uses the browser in the phone.

Java card Our emulated Java card keeps track of a sequence number and has a secret key that is shared with the web server. When the phone taps on the card, the card calculates an HMAC using the SHA1 algorithm over the sequence number, the table number and the card's UID. The HMAC, sequence number, and table number are appended to the URL of the restaurant's web page.

Web server Our web server is an Apache server version 2.2.20. The server connect to a MySQL database, named *Restaurant*, that stores the secret keys of the Java cards associated with the restaurant tables. This database also stores all the orders that have been placed and processed. When an HTTP request comes to the server and the sequence number and the HMAC value provided by the HTTP request pass all the checks on the server, then the server returns a menu in the HTTP response. When the customer has decided on his meal, he submits his choice. The submission also includes the sequence number, HMAC, and table number - as in the earlier URL. Thus, the server can check that this is a valid submission. The idea of storing all the orders that have been placed and processed is not new and has been supported by many restaurants.

Websocket server The websocket server was implemented by using the nodejs[2] platform version 0.10.10. The reason for the use of websocket [62] is that it supports bidirectional communication between a client and a server without the need to use the polling mechanism of traditional HTTP. The websocket server has access to the *Restaurant* database to get new orders and continuously pushes these new order to the websocket-supporting browsers in the client computers. These browsers have permanent connections to a websocket server in order to keep them updated with new orders. Once an order is completed, the cook selects the entry in the list and clicks on the *Delete* button. This delete event is processed by the websocket server to delete the corresponding entry in the *Restaurant* database.

[2]http://nodejs.org/

Chapter 5

Discussion

In this chapter, we generalize the uses of general NFC tags for different security requirements and considerations. We then briefly summarize the methodological principles used in this thesis. Thereafter, we present how we used *proverif*, a protocol verifier, to verify some of our protocols. Following this, we discuss some reflections on economic, social, and ethical issues associated with this thesis project.

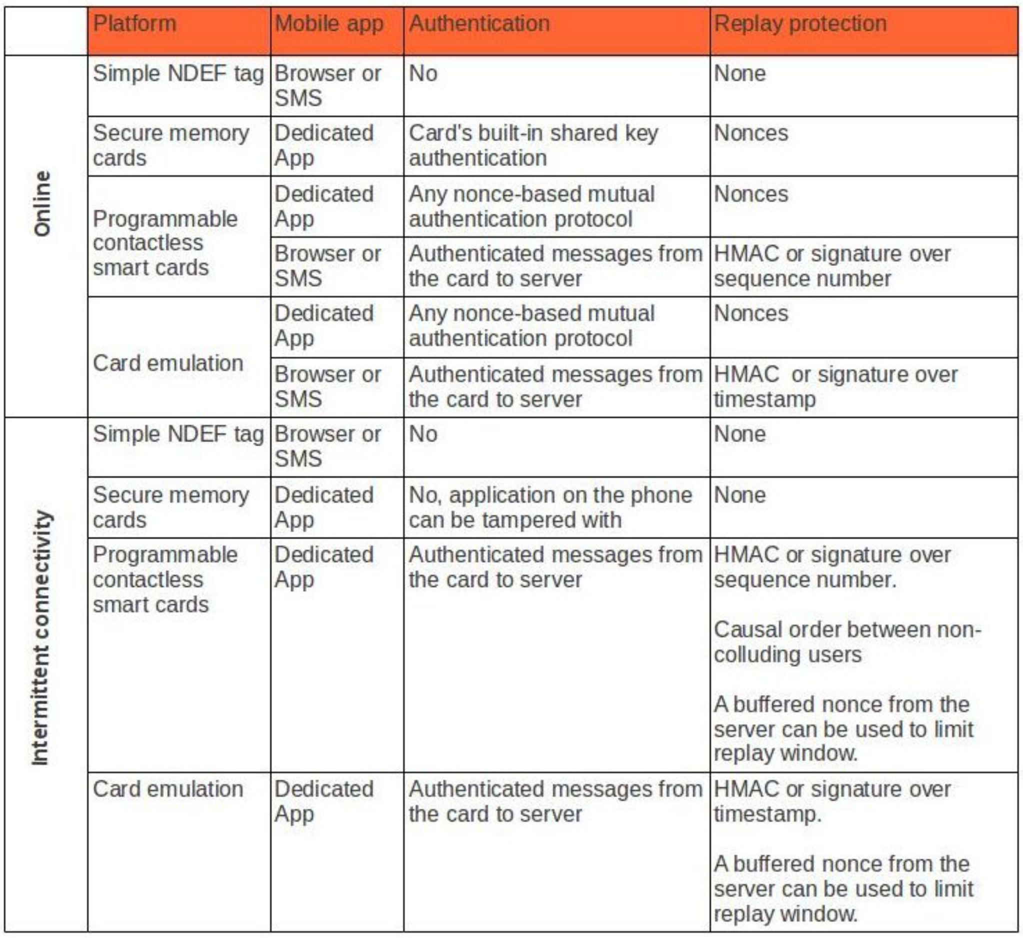
5.1 Security Generalization

In previous chapters, we presented several specific NFC-enabled use cases using different NFC tags, specifically DESFire EV1 tags, Java cards, and NFC card emulation using an NFC reader connected to a computer. All presented solutions use off-the-shelf devices which are readily available on the market. This means that our solutions can be easily deployed in real world.

All the presented use cases illustrate that NFC-enabled systems are distributed systems involving three main components: an NFC tag, an NFC-enabled phone with or without a dedicated application, and a back-end server. The NFC-enabled phone and the dedicated application help to relay messages exchanged between the NFC tag and the back-end server. In addition, the security of NFC-enabled systems depends on the Internet connectivity of the phone at the time the phone taps on the NFC tag. The general principles of using NFC tags for securing NFC-enabled systems with different Internet connectivity status is summarized are Figure 5.1.

As explained throughout the thesis, NFC tags comprise tags based on secure memory cards (e.g. Mifare DESFire EV1 tag), programmable contactless smart cards (e.g. Java cards), and emulated cards. Other than these three types of tags, NFC tags can be simple NDEF tags that are capable of storing NDEF messages but do not support any built-in authentication mechanisms. For example, Mifare Ultralight tags [14] are simple NDEF tags. These types of tags can be used in a variety of service-initiation applications, as described in Section 2.10. It is worth mentioning that, in these systems, we do not need a dedicated application installed on the phone. Instead, we can use the web browser or the SMS application supported by the phone. In addition, systems based on simple NDEF tags do not require the phone to have Internet connectivity. They also do not provide any authentication or replay protection.

In the following sections, we present a detailed discussion about the principles of using the NFC tags based on secure memory cards, programmable contactless smart cards, and card emulation.



	Platform	Mobile app	Authentication	Replay protection
Online	Simple NDEF tag	Browser or SMS	No	None
	Secure memory cards	Dedicated App	Card's built-in shared key authentication	Nonces
	Programmable contactless smart cards	Dedicated App	Any nonce-based mutual authentication protocol	Nonces
		Browser or SMS	Authenticated messages from the card to server	HMAC or signature over sequence number
	Card emulation	Dedicated App	Any nonce-based mutual authentication protocol	Nonces
		Browser or SMS	Authenticated messages from the card to server	HMAC or signature over timestamp
Intermittent connectivity	Simple NDEF tag	Browser or SMS	No	None
	Secure memory cards	Dedicated App	No, application on the phone can be tampered with	None
	Programmable contactless smart cards	Dedicated App	Authenticated messages from the card to server	HMAC or signature over sequence number. Causal order between non-colluding users A buffered nonce from the server can be used to limit replay window.
	Card emulation	Dedicated App	Authenticated messages from the card to server	HMAC or signature over timestamp. A buffered nonce from the server can be used to limit replay window.

Figure 5.1: General principles of using NFC tags

5.1.1 Tags based on secure memory cards

The tag based on secure memory cards used in all the use cases presented in Chapter 3 is the DESFire EV1 tag. Some other examples of tags based on secure memory cards are Mifare Ultralight C [16] and Mifare Classic [63] tags. A tag of this kind is capable of storing application-defined data and allows application developers to control the read and write access to this data by using the card's built-in nonce-based shared-key three-pass mutual authentication between the tag and the NFC reader. This built-in authentication mechanism for read and write access controls is more advanced than the irreversible lock-byte mechanism in some type 2 tags with static memory structures [64], such as MiFare Ultralight [14] tags. The reason is that a secure memory tag allows the tag owner who has the key to change the tag content whenever he wants. In contrast, once a MiFare Ultralight tag is set to the read-only state, its content cannot be changed any more.

Apart from the read and write access controls using the built-in authentication mechanism, an authentication session between a secure memory tag and a back-end server via an NFC phone can prove to the back-end server that the user or its agent is close to the tag at the time the session takes place. Since a tag can represent a location due to its unique tag UID, an authentication session between the tag and the back-end server via the phone can prove that the phone is at that specific location. This feature has been employed in the security guard monitoring system presented in Section 3.1.6, and in the school attendance monitoring system presented in Section 3.4. This can also be used in the food delivery application presented in Section 2.10 by using the same design as in the guard monitoring system. It is worth noting that, in order to use this built-in authentication mechanism, a dedicated application installed on the phone is necessary to facilitate message exchanges between the tag and the application server. In addition, it is compulsory to have an Internet connection between the phone and the server at the time the tag is tapped. Our experiments (Section 4.2.3) showed that some delay in the network connection does not break the authentication sessions.

If the location has no wireless Internet coverage and the phone does not have Internet connectivity at the time it taps on the tag, then the tag's built-in authentication between the tag and the server cannot be executed. In this case, if we have a dedicated application on the phone, the application can scan and store the tag's UID and sends this data to the application server when the phone gets connected to the Internet. However, the application on the phone can be tampered with. Thus, data stored in the application can be changed by attackers. In addition, if only the UID is used to verify the freshness of a session, then an attacker can manually generate this data. To illustrate this, Henzl [65] examined several access control systems and found that the tested systems use only the UID of the card for verifying the user identity and that there is no authentication between the card and the reader or encrypted communication. Therefore, the author was able to get the UID without authenticating to the card and then replay it to get access to the locked door.

As mentioned before, there are various tags based on secure memory cards, such as the Mifare Classic tag [63], and the Mifare Ultralight C tag [16]. However, it is important to examine the security strength of the authentication protocol provided by each tag. For example, any bad properties in the random number generator could compromise the security of the whole authentication protocol. Merhi et al. [66] confirmed that the random number generator of Mifare Ultralight C card is a major improvement over that of Mifare Classic. Nohl et al. [67] performed reverse engineering on the Mifare Classic tag and found that the nonce generated by this tag is not random, which enables the attacker to replay the authentication. Moreover, Koning et al. [68] exploited the weakness on the random number generator of Mifare Classic card to recover the keystream generated by the card. In addition, they were able to read all memory blocks of the first sector of the card and modify memory blocks on the card. What is more, Garcia et al. [69] were able to extract the secret key from just two eavesdropped authentications between a card and a genuine reader. This is a significant problem since the Mifare Classic tag is the most widely used contactless smart card in the market (covering more than 70% of contactless cards in the market, as claimed by Garcia et al. [70]). Oswald et al. [71] presented a side-channel attack on Mifare DESFire (MF3ICD40 chip) tags. Specifically, with about 1 000 000 measurements they are able to fully recover the 3DES key stored on a Mifare DESFire card. However, their side-channel attack is currently not applicable to the DESFire EV1 tag [27], which is the specific DESFire tag used in this thesis.

5.1.2 Programmable contactless smart cards

As explained in Section 2.8.1, Java card is an example of programmable contactless smart cards. It is more intelligent than a secure memory tag in that it is programmable; thus it can keep track of a sequence number and perform computations. However, there is no built-in authentication protocol defined by this type of card and the security mechanism must be implemented by the application in the card. As with the secure memory tags, security of NFC-enabled systems using programmable contactless smart cards depends on the Internet connectivity of the phone and the applications on the phone, as explained below.

Reliable connectivity If we have reliable wireless Internet coverage at the location of the tags, the programmable contactless smart card enables us to design nonce-based mutual authentication protocols that fit our security requirements and make them as flexible as we want. In addition, as with the secure memory tags, we need a dedicated application installed on the phone in order to facilitate message exchanges between the card and the application server. To illustrate that the programmable card is more flexible than the secure memory tag, let us consider the two security guard systems described in Section 3.1.5 and 3.1.6. Specifically, in the security guard system using DESFire EV1 tags (Section 3.1.6), the card with its built-in three-pass mutual authentication protocol helps to authenticate the reader and the tag to each other. Although we understand that, in this use case, authenticating the reader side is not necessary since it does not add more security to the system, we cannot customize the protocol. In contrast, when using Java cards in the same use case, we can design a challenge-response protocol that meets our requirements with fewer round messages and more functionality, as explained in Section 3.1.5.

This card can also be used in NFC-enabled systems that use web browser or built-in SMS application on the phone instead of a dedicated application. Specifically, due to the ability to keep track of a sequence number in the programmable smart card, an authenticated message which contains an HMAC or a signature can be sent from the card to the server to provide replay protection. For example, in the NFC-enabled restaurant use case presented in Section 3.2, a sequence number and its HMAC are added to an URL to prove the freshness of an HTTP request. Another example, in the vending machine system using Java card, the sequence number and its HMAC are added to the SMS message to prevent replay attacks.

Intermittent connectivity Lack of reliable Internet connectivity at the location of the tags means that we need a dedicated application installed in the phone to buffer messages from the card to the server. If the phone does not have Internet connectivity during the time it taps on the tag, the sequence number maintained by the card can provide a certain level of replay protection by verifying the causal order between taps by non-colluding users. Specifically, the authenticated message can contain an HMAC or a signature over the sequence number maintained by the card. This was applied and analysed in Section 3.1.5. However, as explained in Section 3.1.5, the order of events cannot guarantee the trustworthiness of a report if all users collude or if there is only one user using the system. In order to limit the replay window, we can use a buffered, relatively recent nonce sent from the server when the phone does not have Internet connectivity at the time it taps on the tag.

In addition, to understand the general principles of using programmable smart cards, it is important to understand and make use of security mechanisms that are enforced by the card platform in the implementation of the system. For example, the Java card relies on the JCVM abstraction layer which provides several security services to the application layer. Specifically, its firewall mechanism provides secure and isolated execution of applets [72]. In addition, illegal memory accesses are prevented by stack and buffer overflow checking. Moreover, in the last version of the Java Card specifications, the bytecode verifier is embedded on card, making the bytecode verification process mandatory. This mechanism guarantees that an application that is not conform to the Java Card specifications cannot be loaded onto the Java card. However, there are some attacks against this platform, such as fault attacks [73] and combined attacks [74]. In addition, the attacker may collect information on the card to elaborate hardware and software attacks, or attack the virtual machine and the firewall between applications on the card [75].

5.1.3 Card emulation

An NFC reader connected to a computer can emulate an NFC tag. This emulated tag appears to a reader much the same as a real NFC tag, such as a DESFire EV1 tag or a Java card, but it is more powerful than those tags. Specifically, once an emulated card receives messages from a reader, these messages are interpreted and processed by the computer to which the emulated card is connected; thus, the computational capabilities of this card emulation are the same as the capabilities of the connected computer. In addition, the emulated card can provide a clock and possibly its own Internet connection. However, one disadvantage of emulated cards is that the system can be bulky since it needs to be connected to a computer, even if the computer is as small as a Raspberry Pi. Also, not all computers have built-in NFC support. For example, in the vending machine presented in Section 3.3, it may be difficult to change the micro-controller inside the machine so that we can attach an NFC reader to it for card emulation. The security of NFC-enabled systems using NFC readers running in card emulation mode depends on Internet connectivity of the phone, as explained below:

Reliable connectivity Assuming that the location of the tags has a reliable wireless Internet coverage, as for the programmable contactless smart card, the emulated card allows us to design nonce-based mutual authentication protocols that fit our security requirements and make them as flexible as we want. In addition, as with the two types of tags mentioned above, we need a dedicated application installed on the phone in order to facilitate message exchanges between the card and the application server.

This card can also be used in NFC-enabled systems that use web browser or built-in SMS application on the phone instead of a dedicated application. Specifically, due to the ability to support timestamp, an authenticated message, which contains an HMAC or a signature over the timestamp, can be sent from the card to the server to provide replay protection. It is important to note that an HMAC over the timestamp provides more security than an HMAC over the sequence number. This is because the timestamp is more fine-grained than the sequence number, or in other words, the replay window of the timestamp is smaller than that of the sequence number.

Intermittent connectivity If we cannot have a reliable wireless Internet coverage at the location of the tags, provided that the phone has a dedicated application installed, the HMAC calculated over the timestamp can provide replay protection. To illustrate this, in the vending machine application (Section 3.3), timestamp in a message that is protected by HMAC guarantees the freshness of the message. In order to limit the replay window, we can use a buffered, relatively recent nonce sent from the server in case the phone does not have Internet connectivity at the time it taps on the tag.

5.2 Protocol verification

In Chapter 3, we proposed several security protocols with different security properties. In each security protocol design, we presented our theoretical protocol analysis of the design. In order to ensure that our designs meet the security goals, we tried to use *proverif* [76], a security protocol verifier, to verify our protocols. Unfortunately, this tool does not provide recursion or loop. Thus, we can just have an approximate model of the sequence number. In order to simplify the verification, we verified a simpler version of the security protocol for the guard monitoring system using Java cards (Section 3.1.5), as shown below. Specifically, the sequence number is removed from the protocol to simplify the verification.

```
1.  S -> C: R
```

```
2.  M -> C: R, phoneInfor
```
3. `C -> S: UID, R, phoneInfor,` MAC_K`(UID, R, phoneInfor)`

The verification program assumes that the card and the server communicate with each other in a hostile environment. These two communicating parties share a secret key which is calculated from the card's UID and the server's master key by using a one-way key generator function.

The security goal that we aim to verify in this protocol is the following correspondence property [77]: if a server accepts a message which comprises a random number R, some information about the phone, and the card's UID at time t_1, then the card must have sent the same information at an earlier time $t_0 < t_1$. The protocol model is shown below. In the model, the phone is also modeled for clarity but it has no security and can be an attacker. Verification of the model with proverif showed that no attack was found in our model.

```
free c.
free phoneInfor.
private free masterKey.
fun mac/2.
fun keygen/2.
query evinj:acceptResponse(x1,x2,x3)
  ==> evinj:sendResponse(x1,x2,x3).

let card =
in (c, message);
let (R, phoneInfor) = message in
event sendResponse(uid, R, phoneInfor);
out(c, mac((uid, R, phoneInfor), k)).

let phone =
in (c, R);
out (c, (R, phoneInfor)).

let server =
new R;
out (c, R);
in (c, (cardUID, =R, phoneInfor, mac1));
let key = keygen(cardUID, masterKey) in
if mac1 = mac((cardUID, R, phoneInfor), key) then
event acceptResponse(cardUID, R, phoneInfor).
```

```
process
(
!(
new uid;
let k = keygen(uid, masterKey) in
card )
|  !(phone)
|  !(server)
)
```

5.3 Summary of methodology

In this thesis, to gain a general picture about how NFC tags could be used, we started by reviewing current NFC-enabled systems and found possible problems in them. After that, we chose four specific use cases in which problems were found, examined their security requirements and designed new security protocols for them. By examining specific use cases, we gathered detailed requirements and derived details on the interactions between the system components and users as well as other external entities interacting with it. In addition, to deeply understand each use case, we studied its state of the art and security threats associated with it. From that, we gained a clear idea of the current security objectives and security requirements of the system. Based on these security goals and knowledge about properties of NFC tags, we chose a suitable type of tag and designed security protocols for the system. We also developed proof-of-concept implementations for some of the proposed protocols in this thesis. In addition, we used the *proverif* security verification tool to verify one of the proposed protocols. In Section 5.1, we generalized the principles of using NFC tags based on knowledge gained from the designs and implementations of the specific use cases as well as on our understanding of the features of NFC tags. In addition, we discussed the differences between the properties of different tags as well as the threats associated with them.

5.4 Reflections

We have witnessed over the past years how NFC technology has rapidly evolved with a number of proposed solutions and pilot trials. This technology has been widely used in some of our everyday applications, such as access controls, and public transportation.

As pointed out by Ozdenizci et al. [78], the most popular NFC related research subjects are on developing new NFC enabled applications. In this thesis, we started by examining four specific use cases that have already been proposed in the literature and pilot trials. Our contribution is to add security and usability to these use cases. In addition, we generalize the principles of using different NFC tags based on these specific use cases. With regards to ethical and social aspects, our solutions aim to prevent or discourage fraud in the examined systems and to provide good usability to users. Moreover, our solutions bring clear business benefits and cost savings for organisation by automating the work and making the organizational processes more reliable.

There are some improvements that can be added to this thesis. Firstly, in all our implementations, instead of working on real Java cards, we emulated functions of them by using an NFC reader connected to a computer. Besides implementations on real Java cards, we can do performance measurements on different types of tags over Internet connections with variable latency to evaluate the performance of different designs.

Chapter 6

Conclusion

NFC technology is evolving rapidly with various use cases that have been proposed in the literature and a large number of pilots. Applications of NFC have shifted from its original drivers, i.e. contactless payment and ticketing applications, to a surprisingly diverse range of other applications, such as authentication, education, and healthcare applications. Juniper Research [79] predicts that in 2013 there will be 700 million owners of NFC-enabled phones, while in 2014 one mobile user out of six in the world will own a mobile device equipped with this technology. This large number of potential users means that NFC promises a great potential future although it is still in its early development phase.

Although a variety of NFC-enabled systems have been proposed, they have many problems, such as poor usability, potential security flaws, and even unrealistic protocols as we explained in Chapter 3. One critical challenge in NFC is to ensure security in NFC-enabled systems. However, we have discussed in Chapter 1 that ensuring security in NFC applications is non-trivial because different NFC applications have different security requirements and rely on the security features as well as computational capabilities of the NFC devices involved in them.

The thesis illustrated that NFC applications are distributed systems which include three main components: NFC tags, NFC readers or NFC-enabled phones, and online servers. These systems introduce a new type of security protocol which involves the three components and the communication channel between them in order to secure NFC applications.

In this thesis, we focused on exploiting different capabilities of NFC tags for securing NFC applications. We move from specific use cases to general principles. Our analysis has shown that among the three types of NFC tags discussed in our work, including tags based on secure memory cards, programmable contactless smart cards, and emulated cards, the secure memory tags have the least capability and flexibility but they can nevertheless be used to build secure systems for many applications. Its built-in three-pass mutual authentication not only benefits applications that require settings on read and write permissions, but can also be used to prove the freshness of the event that the tag is tapped, provided that the dedicated application installed in the phone has an Internet connection at the time the tap event occurs. The programmable contactless smart cards are more intelligent and flexible than tags based on secure memory cards in that they can keep track of a sequence number and perform computations. This sequence number is helpful when the sequence of events can provide a certain level of trust in the freshness of the tap event. However, compared to the emulated cards, both types of tags are more vulnerable to tag replacement attacks. Thus, they should be placed in such a way that it is difficult to be replaced by another tag. The most powerful type of tags used in this thesis is the emulated card. This card provides a clock for complete protection against replay attacks even in offline settings as well as strong computational capacity due to the computer connected to the reader.

Our implementations in this thesis are working prototypes providing the essential core features which are usable as a solid base for future developments and experiments. This was a challenging and interesting project since it involved research and development in many different areas, including working on Raspberry Pi, writing Android applications, C programming, Java applications, PHP programming, websocket programming, and implementing a RESTful web service interfacing with a database. In addition, we gained experience in programming NFC tags, card emulation, and security protocol verification tools.

To the best of our knowledge, this is the first work that presents how to apply the different properties of NFC tags for securing NFC applications. Possible future work after this thesis should consider the following:

- Discover and study other use cases where NFC can be applied.
- Implement the presented use cases using real Java cards.
- Improve the applications for real-world deployments. For example, we can add more functions to the security guard application, such as to enable a security guard to take pictures and report equipment or property damage or unusual occurrences at checkpoints.

Bibliography

[1] NFC Forum, "NFC Data Exchange Format (NDEF)," 2006, http://www.nfc-forum.org/specs/spec_license/document_form/custom_layout?1363171338162. Accessed 12.03.2013.

[2] A. Cavoukian, *Mobile Near Field Communications (NFC):" tap'n Go": Keep it Secure and Private.* Information and Privacy Commissioner of Ontario, Canada, 2011.

[3] K. Ok, V. Coskun, M. Aydin, and B. Ozdenizci, "Current benefits and future directions of NFC services," in *Education and Management Technology (ICEMT), 2010 International Conference on*, 2010. doi: 10.1109/ICEMT.2010.5657642 pp. 334–338.

[4] "BCS Checkpoint/Incident/Media Tags," http://www.bcsint.com/cmtags.php. Accessed 15.04.2013.

[5] "Incentive Lynx Security selects SmartTask to enhance proof of attendance and workflow management," http://www.incentive-lynx.com/pages/news-items/incentive-lynx-security-selects-smarttask-to-enhance-proof-of-attendance-and-workflow-management.php. Accessed 04.06.2013.

[6] "Real-time Guard Tour Monitoring for Security Companies with the ENAiKOON NFC Tracking System," http://www.eventsecurityportal.com/eventsecurity_news.asp?articleid=268721&arttitle=Real-time%20Guard%20Tour%20Monitoring%20for%20Security%20Companies%20with%20the%20ENAiKOON%20NFC%20Tracking%20System. Accessed 21.04.2013.

[7] "NFC Patrol," http://www.hotech.gr/index.php/what-we-do-en/solutions/nfc-solutions/patrol. Accessed 21.04.2013.

[8] R. Want, "Near field communication," *Pervasive Computing, IEEE*, vol. 10, no. 3, pp. 4–7, 2011. doi: 10.1109/MPRV.2011.55

[9] P. McDermott-Wells, "What is bluetooth?" *Potentials, IEEE*, vol. 23, no. 5, pp. 33–35, 2005. doi: 10.1109/MP.2005.1368913

[10] B. Crow, I. Widjaja, J. G. Kim, and P. Sakai, "Ieee 802.11 wireless local area networks," *Communications Magazine, IEEE*, vol. 35, no. 9, pp. 116–126, 1997. doi: 10.1109/35.620533

[11] NFC Forum, "NFC Frequently Asked Questions," http://www.nfc-forum.org/resources/faqs/. Accessed 12.03.2013.

[12] "ISO14443," http://www.openpcd.org/ISO14443. Accessed 04.06.2013.

[13] "Information technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1)," http://standards.iso.org/ittf/PubliclyAvailableStandards/c056692_ISO_IEC_18092_2013.zip. Accessed 04.06.2013.

[14] NXP Semiconductors, "MIFARE Ultralight contactless single-ticket IC," http://www.nxp.com/documents/data_sheet/MF0ICU1.pdf. Accessed 10.06.2013.

[15] "MIFARE DESFire EV1 contactless multi-application IC," 2010, http://www.nxp.com/documents/short_data_sheet/MF3ICDX21_41_81_SDS.pdf. Accessed 10.06.2013.

[16] "MIFARE Ultralight C," http://www.nxp.com/products/identification_and_security/smart_card_ics/mifare_smart_card_ics/mifare_ultralight/series/MIFARE_ULTRALIGHT_C.html, Accessed 10.06.2013.

[17] "Introduction to NFC Forum Type Tags," http://www.nfc-forum.org/resources/white_papers/NXP_BV_Type_Tags_White_Paper-Apr_09.pdf. Accessed 12.03.2013.

[18] "NFC specifications," http://www.nfc-forum.org/specs/spec_license. Accessed 15.04.2013.

[19] NFC Forum, "Type 4 Tag Operation Specification," 2011, http://www.nfc-forum.org/specs/spec_license/document_form/custom_layout?1363171723024. Access 13.03. 2013.

[20] "ISO/IEC 7816-4:2013," http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=54550. Accessed 07.06.2013.

[21] G. Madlmayr, J. Langer, C. Kantner, and J. Scharinger, "NFC Devices: Security and Privacy," in *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, 2008. doi: 10.1109/ARES.2008.105 pp. 642–647.

[22] M. Saeed and C. Walter, "A Record Composition/Decomposition attack on the NDEF Signature Record Type Definition," in *Internet Technology and Secured Transactions (ICITST), 2011 International Conference for*. IEEE, 2011, pp. 283–287.

[23] M. Roland, J. Langer, and J. Scharinger, "Security Vulnerabilities of the NDEF Signature Record Type," in *Near Field Communication (NFC), 2011 3rd International Workshop on*, 2011. doi: 10.1109/NFC.2011.9 pp. 65–70.

[24] L. Francis, G. Hancke, K. Mayes, and K. Markantonakis, "Practical NFC Peer-to-Peer Relay Attack Using Mobile Phones," in *Radio Frequency Identification: Security and Privacy Issues*, ser. Lecture Notes in Computer Science, S. Ors Yalcin, Ed. Springer Berlin Heidelberg, 2010, vol. 6370, pp. 35–49. ISBN 978-3-642-16821-5. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-16822-2_4

[25] L. Francis and G. Hancke, "Practical relay attack on contactless transactions by using NFC mobile phones," *IACR ePrint Archive*, vol. 618, 2011.

[26] J. Fischer, "NFC in cell phones: The new paradigm for an interactive world [Near-Field Communications]," *Communications Magazine, IEEE*, vol. 47, no. 6, pp. 22–28, 2009. doi: 10.1109/MCOM.2009.5116794

[27] T. Kasper, I. Maurich, D. Oswald, and C. Paar, "Chameleon: A Versatile Emulator for Contactless Smartcards," in *Information Security and Cryptology - ICISC 2010*, ser. Lecture Notes in Computer Science, K.-H. Rhee and D. Nyang, Eds. Springer Berlin Heidelberg, 2011, vol. 6829, pp. 189–206. ISBN 978-3-642-24208-3. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24209-0_13

[28] Z. Chen, *Java card technology for smart cards: architecture and programmer's guide*. Prentice Hall, 2000.

[29] Innovision Research, "Near field communication in the real world: turning the NFC promise into profitable, everyday applications," 2006, http://www.nfc-forum.org/resources/white_papers/Innovision_whitePaper1.pdf. Accessed 10.06.2013.

[30] NFC Forum, “Smart Posters, How to use NFC tags and readers to create interactive experiences that benefit both consumers and businesses,” April 2011.

[31] C. Mulliner, “Vulnerability Analysis and Attacks on NFC-Enabled Mobile Phones,” in *Availability, Reliability and Security, 2009. ARES 09. International Conference on*, 2009. doi: 10.1109/ARES.2009.46 pp. 695–700.

[32] “Vending machines,” http://www.selecta.com/vending-machines/. Accessed 04.06.2013.

[33] A. Andersen and R. Karlsen, “Experimenting with Instant Services Using NFC Technology,” in *SMART 2012, The First International Conference on Smart Systems, Devices and Technologies*, 2012, pp. 73–78.

[34] P. Garrido, G. Miraz, I. Ruiz, and M. Gomez-Nieto, “Use of NFC-based Pervasive Games for Encouraging Learning and Student Motivation,” in *Near Field Communication (NFC), 2011 3rd International Workshop on*, 2011. doi: 10.1109/NFC.2011.13 pp. 32–37.

[35] M. Bueno-Delgado, P. Pavon-Marino, A. De-Gea-Garcia, and A. Dolon-Garcia, “The Smart University Experience: An NFC-Based Ubiquitous Environment,” in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on.* IEEE, 2012. doi: 10.1109/IMIS.2012.110 pp. 799–804.

[36] S. B. Patela and N. K. Jainb, “Near Field Communication (NFC) based Mobile Phone Attendance System for Employees,” *International Journal of Engineering*, vol. 2, no. 3, 2013.

[37] T. Tuikka and M. Isomursu, “Touch the future with a smart touch,” *VTT Tiedotteita-Research Notes*, vol. 2492, 2009.

[38] NXP Semiconductors, “NFC Connection Handover 1.2,” http://www.nfc-forum.org/specs/spec_license/document_form/custom_layout?1366282155771. Access 15.04.2013.

[39] R. Steffen, J. PreiBinger, T. Schollermann, A. Muller, and I. Schnabel, “Near Field Communication (NFC) in an Automotive Environment,” in *Near Field Communication (NFC), 2010 Second International Workshop on*, 2010. doi: 10.1109/NFC.2010.11 pp. 15–20.

[40] E. Strommer, J. Kaartinen, J. Parkka, A. Ylisaukko-oja, and I. Korhonen, "Application of Near Field Communication for Health Monitoring in Daily Life," in *Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE*, 2006. doi: 10.1109/IEMBS.2006.260021. ISSN 1557-170X pp. 3246–3249.

[41] "Transit and Contactless Open Payments: An Emerging Approach for Fare Collection," http://www.smartcardalliance.org/resources/pdf/Open_Payments_WP_110811.pdf. Accessed 01.05.2013.

[42] "Google Wallet. How it works," http://www.google.com/wallet/how-it-works/. Accessed 01.05.2013.

[43] S. Dominikus and M. Aigner, "mCoupons: An Application for Near Field Communication (NFC)," in *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, vol. 2, 2007. doi: 10.1109/AINAW.2007.230 pp. 421–428.

[44] "Proxmark3 User Guide," http://proxmark3.com/dl/PM3-UserGuide-r486.pdf. Accessed 27.06.2013.

[45] "Summary - Mobile phone theft," http://www.ons.gov.uk/ons/rel/crime-stats/crime-statistics/focus-on-property-crime--2011-12/rpt-section-2---focus-on-mobile-phone-theft.html. Accessed 07.06.2013.

[46] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, ser. SPSM '11. New York, NY, USA: ACM, 2011. doi: 10.1145/2046614.2046618. ISBN 978-1-4503-1000-0 pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/2046614.2046618

[47] M. Reveilhac and M. Pasquet, "Promising Secure Element Alternatives for NFC Technology," in *Near Field Communication, 2009. NFC '09. First International Workshop on*, 2009. doi: 10.1109/NFC.2009.14 pp. 75–80.

[48] M. Roland, J. Langer, and J. Scharinger, "Practical Attack Scenarios on Secure Element-Enabled Mobile Devices," in *Near Field Communication (NFC), 2012 4th International Workshop on*, 2012. doi: 10.1109/NFC.2012.10 pp. 19–24.

[49] S. Turner and T. Polk, “The TLS protocol version 2.0,” March 2011, http://tools.ietf.org/html/rfc6176. Accessed 04.06.2013.

[50] H. Krawczyk, R. Canetti, and M. Bellare, “HMAC: Keyed-hashing for message authentication,” 1997.

[51] T. Aura, “Strategies against replay attacks,” in *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, 1997. doi: 10.1109/CSFW.1997.596787. ISSN 1063-6900 pp. 59–68.

[52] “M-Payment systems,” http://www.mobill.se/en/apps-services/m-payment/. Accessed 21.04.2013.

[53] “What are SMS payments,” http://www.mobiletransaction.org/what-are-sms-payments/. Accessed 21.04.2013.

[54] “Coca-Cola Mobile Vending Machine Purchase Trial,” http://support.t-mobile.com/docs/DOC-5195. Accessed 21.04.2013.

[55] “The Tag Dispatch System in Android,” http://developer.android.com/guide/topics/connectivity/nfc/nfc.html#tag-dispatch. Accessed 15.04.2013.

[56] “ACR122U USB NFC Reader,” http://www.acs.com.hk/index.php?pid=product&id=ACR122U. Accessed 10.06.2013.

[57] “Libnfc Main Page,” http://nfc-tools.org/index.php?title=Main_Page. Accessed 10.06.2013.

[58] “PN53x,” http://nfc-tools.org/index.php?title=PN53x. Accessed 10.06.2013.

[59] “Raspberry Pi,” http://www.raspberrypi.org/downloads. Accessed 10.06.2013.

[60] D. Eastlake and P. Jones, “US secure hash algorithm 1 (SHA1),” 2001, http://www.ietf.org/rfc/rfc3174.txt.

[61] “SQLiteDatabase,” http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html. Accssed 15.04.2013.

[62] I. Fette and A. Melnikov, “The websocket protocol,” http://tools.ietf.org/html/rfc6455.

[63] “MIFARE Classic - a pioneer and front runner in contactless smart card ICs,” http://www.nxp.com/products/identification_and_security/smart_card_ics/mifare_smart_card_ics/mifare_classic/. Accessed 10.06.2013.

[64] NFC Forum, “Type 2 tag operation specification,” 2011, http://www.nfc-forum.org/specs/spec_license/document_form/custom_layout?1370858678468. Accessed 10.06.2013.

[65] M. Henzl, “Security of Contactless Smart Cards,” in *Proceedings of the 17th Conference STUDENT EEICT 2011.* Brno University of Technology, 2011, pp. 585–589.

[66] M. Merhi, J. Hernandez-Castro, and P. Peris-Lopez, “Studying the pseudo random number generator of a low-cost RFID tag,” in *RFID-Technologies and Applications (RFID-TA), 2011 IEEE International Conference on*, 2011. doi: 10.1109/RFID-TA.2011.6068666 pp. 381–385.

[67] K. Nohl, D. Evans, S. Starbug, and H. Plötz, “Reverse-engineering a cryptographic RFID tag,” in *Proceedings of the 17th conference on Security symposium*, 2008, pp. 185–193.

[68] G. Koning Gans, J.-H. Hoepman, and F. Garcia, “A practical attack on the mifare classic,” in *Smart Card Research and Advanced Applications*, ser. Lecture Notes in Computer Science, G. Grimaud and F.-X. Standaert, Eds. Springer Berlin Heidelberg, 2008, vol. 5189, pp. 267–282. ISBN 978-3-540-85892-8. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85893-5_20

[69] F. Garcia, G. Koning Gans, R. Muijrers, P. Rossum, R. Verdult, R. Schreur, and B. Jacobs, “Dismantling MIFARE Classic,” in *Computer Security - ESORICS 2008*, ser. Lecture Notes in Computer Science, S. Jajodia and J. Lopez, Eds. Springer Berlin Heidelberg, 2008, vol. 5283, pp. 97–114. ISBN 978-3-540-88312-8. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88313-5_7

[70] F. Garcia, P. van Rossum, R. Verdult, and R. Schreur, “Wirelessly Pickpocketing a Mifare Classic Card,” in *Security and Privacy, 2009 30th IEEE Symposium on*, 2009. doi: 10.1109/SP.2009.6. ISSN 1081-6011 pp. 3–15.

[71] D. Oswald and C. Paar, “Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World,” in *Cryptographic*

Hardware and Embedded Systems - CHES 2011, ser. Lecture Notes in Computer Science, B. Preneel and T. Takagi, Eds. Springer Berlin Heidelberg, 2011, vol. 6917, pp. 207–222. ISBN 978-3-642-23950-2. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-23951-9_14

[72] D. Perovich and S. Antipolis, "Secure object sharing development kit for Java Card," in *Proceedings, VerifiCard Annual Meeting*, 2002.

[73] G. Barbu, G. Duc, and P. Hoogvorst, "Java Card Operand Stack: Fault Attacks, Combined Attacks and Countermeasures," in *Smart Card Research and Advanced Applications*, ser. Lecture Notes in Computer Science, E. Prouff, Ed. Springer Berlin Heidelberg, 2011, vol. 7079, pp. 297–313. ISBN 978-3-642-27256-1. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-27257-8_19

[74] G. Barbu, H. Thiebeauld, and V. Guerin, "Attacks on Java Card 3.0 Combining Fault and Logical Attacks," in *Smart Card Research and Advanced Application*, ser. Lecture Notes in Computer Science, D. Gollmann, J.-L. Lanet, and J. Iguchi-Cartigny, Eds. Springer Berlin Heidelberg, 2010, vol. 6035, pp. 148–163. ISBN 978-3-642-12509-6. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12510-2_11

[75] S. Chaumette and D. Sauveron, "New security problems raised by open multiapplication smart cards," *LaBRI, Université Bordeaux*, vol. 1, pp. 1332–04, 2004.

[76] B. Blanchet and B. Smyth, "Proverif 1.85: Automatic cryptographic protocol verifier, user manual and tutorial," 2011, http://www.hit.bme.hu/~buttyan/courses/BMEVIHIM132/manual.pdf.

[77] B. Blanchet, "Automatic verification of correspondences for security protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 363–434, 2009.

[78] B. Ozdenizci, M. N. Aydin, V. Coskun, and O. Kerem, "Design science in NFC research," in *Internet Technology and Secured Transactions (ICITST), 2010 International Conference for.* IEEE, 2010, pp. 1–6.

[79] "NFC Mobile Payments and Marketing Opportunities," http://juniperresearch.com/reports/NFC_mobile_payments_and_marketing_opportunities. Accessed 01.04.2013.

Appendix A

Source code

A.1 Security guard monitoring system

A.1.1 Application Server

The implementation of the application server is at https://guard-application-server.googlecode.com/svn/trunk/. It consists of the main class *SecGuard.java* and its helper classes.

A.1.2 Guard application for the system using Java cards

The implementation of the guard application is at https://guard-application-jc.googlecode.com/svn/trunk/. It consists of the main class *Guarding_JC.java* and its helper classes. Specifically, four classes: *DBEntry.java*, *TosendDatabaseHelper.java*, *TosendDataSource.java*, and *TosendTable.java*, deal with the local SQLite database in the application to store pending reports when the phone does not have Internet connectivity. The *ConnectivityChangeReceiver.java* class handles the changes in the Internet connectivity status of the phone. The *Utility.java* class facilitates data processing.

A.1.3 Guard application for the system using DESFire EV1 tags

The implementation of the guard application is at https://guard-application-desfire.googlecode.com/svn/trunk/. It consists of the main class *Guarding.java* and its helper class named *Utility.java*

A.1.4 DESFire EV1 tag

To write an NDEF message with a our own MIME type (for example, *application/guardDESFire* in this use case), we made use of the guard application to write an array of bytes, that we manually calculated, to the DESFire tag. The code for this task is the *writeNdef* function in the *SecGuard.java* class in the guard application provided in Appendix A.1.3.

An implementation to write an URL or an SMS message to a DESFire EV1 tag is at https://write-ndef-to-desfire.googlecode.com/svn/trunk/. The implementation provides functions to format a DESFire EV1 tag as an NFC type 4 tag, a function to write an URL or an SMS message to the tag, and a function to read the content of the tag. The implementation consists of the main class, named *CardReader.java*, and two helper classes: *NdefMessage.java* and *NdefRecord.java*.

A.1.5 Java card

The code to emulate an NFC reader as a Java card in this use case is at https://javacard-emulation-security-guard.googlecode.com/svn/trunk/. To run the code, an NFC reader connected to a Linux computer which has the libnfc library [57] version 1.4.2 installed are needed. The implementation consists of the main file *process.c* and its helper classes.

A.2 NFC-enabled restaurant

A.2.1 Java card

The code to emulate an NFC reader as a Java card in this use case is at https://javacard-emulation-restaurant.googlecode.com/svn/trunk/. To run the code, an NFC reader connected to a Linux computer which has the libnfc library [57] version 1.4.2 installed are needed. The implementation consists of the main file *process.c* and its helper classes.

A.2.2 Web server

The implementation of the web page for the restaurant use case is at https://webserver-restaurant.googlecode.com/svn/trunk/. The code was written in PHP. To run the code, an Apache[1] server is needed.

[1]http://httpd.apache.org/

A.2.3 Websocket server

The implementation of the websocket server for the restaurant use case is at https://websocket-restaurant.googlecode.com/svn/trunk/. To run the code, a nodejs[2] library need to be installed.

[2]http://nodejs.org/

TRITA-ICT-EX-2013:125