UNIVERSITY OF BIRMINGHAM

Analysis of a real world RFID system

Author: Andrei-Marius Longhin
Student ID: 1583317
Supervisor: David Oswald
School of Computer Science, University of Birmingham

This dissertation is submitted on 9 April, 2018 for the degree of BSc in Computer Science

Abstract

Felicity Card (FeliCa) is a contactless smart card system developed by Sony. Although this technology is increasingly used, a thorough analysis of the system has not been carried out yet. This Final Year Project analyses FeliCa, focusing on three main aspects: inner workings of a card, communication between a card and a reader and security. In the first steps of the project, information about the cards was gathered using multiple sources, from user manuals provided by the manufacturer, to reports generated by already-existing card reader software. Then, once the unencrypted communication protocol had been pinned down, software was developed to drive the interaction between card and reader, using native FeliCa commands. Finally, a number of driver libraries from Sony were reverse-engineered in order to reconstruct the cards security functions. A brief vulnerability analysis of these functions was performed using the communication software from the previous step. The Project provides both a detailed analysis of the system and computer software to put the cards to use. It also yielded some interesting conclusions about the security of this technology, most notably that one specific family of cards is highly likely to be vulnerable to attacks.

Acknowledgements

Firstly, I would like to thank my supervisor, David Oswald, for providing me with such an interesting topic for the Final Year Project. David has been of tremendous help to me throughout the year, from offering prompt and detailed feedback on each project stage to always coming up with new solutions or possibilities whenever I was unsure about the next steps.

I would also like to thank my fellow supervisees for always supporting me and giving me fresh views on whatever I was working on in various project steps. The discussions we had always kept me focused and motivated to keep doing my best on this project.

My friends have also been very supportive throughout the year and this has really made a difference. Therefore, I would like to thank them all for the help and laughs offered when I needed them the most.

Last but not least, I extend my deep love and gratitude to my family, especially my mother and grandmother, without which I wouldn't even dream of achieving anything I've ever managed to accomplish, this project included.

Location of GIT repository

All software for this project can be found at `https://git-teaching.cs.bham.ac.uk/mod-ug-proj-2017/axl518`.

Contents

1 Introduction **11**

1.1 Aims of project . . . 11

1.2 Background needed . . . 11

1.3 Structure of solution . . . 12

1.4 Next sections . . . 12

1.5 Notations used . . . 13

2 Background **15**

3 Technical information **17**

3.1 RFID . . . 17

3.2 NFC . . . 17

3.3 FeliCa . . . 18

3.4 Types of cards . . . 18

3.5 Types of readers . . . 19

3.6 File system . . . 19

3.7 Codes description . . . 20

3.8 Card case study: DES/3DES blank FeliCa card . . . 21

3.9 Summary of section . . . 22

4 Communication protocol **23**

4.1 Structure of a command . . . 23

4.2 Commands specification . . . 24

4.3 Command-response software . . . 24

4.4 Run of unencrypted protocol . . . 25

4.5 Brute-forcing service and area codes . . . 27

4.6 Software evaluation . . . 28

4.7 Summary of section . . . 28

5 Security **31**

5.1 High level overview of mutual authentication . . . 31

5.2 Card case study: DES/3DES Octopus card 32
5.3 Reverse-engineering driver DLLs 33
5.4 Overview of DES 35
5.5 Overview of 3DES 35
5.6 Overview of AES 36
5.7 Reverse-engineering mutual authentication 36
5.8 Access key generation 37
5.9 Mutual authentication flow 37
5.10 Encrypted communication after mutual authentication 38
5.11 On the security of FeliCa 39
5.12 Summary of section 40

6 Discussion **41**

7 Conclusion **43**

8 References **45**

A Structure of submitted zip file **47**

B Commands specification **49**
B.1 Polling 49
B.2 Request service 49
B.3 Read without encryption 50
B.4 Request response 51
B.5 Write without encryption 51

C Service code brute forcer output **53**
C.1 Blank FeliCa card 53
C.2 Octopus card 53

D Mutual authentication commands specification **55**
D.1 Authentication 1 55
D.2 Authentication 2 56

E How to run the code **57**

Chapter 1

Introduction

Sony Felicity Card (FeliCa) is a contactless smart card system developed by Sony. It is used for purposes such as transportation, electronic payments or access control.

FeliCa is advertised as being a technology that couples the ”contactless” convenience with high security and reliabilty.

1.1 Aims of project

The aim of the work described in this Report was to provide a thorough analysis of the FeliCa system. This analysis would cover both the hardware and software components of cards and readers, modelling the interaction between them and assessing the security of this interaction.

1.2 Background needed

The most recommended background reading in order to help understanding the importance of the problem solved by this report is the paper written by Bono et al. (2000). That paper presents the steps undertaken to break another RFID-based device, the Digital Signature Transponder manufactured by Texas Instruments. It is particularly relevant to the task at hand because it underlines the importance of strong security in an RFID technology. It also shows the practical implications of an attack on the system and details the steps needed to successfully perform such an attack.

Furthermore, Sony's overview of the FeliCa product (*What is FeliCa?*, no date) gives more details as to what the purpose of this technology is and what is expected in terms of security and reliability of the system.

1.3 Structure of solution

The analysis of FeliCa yielded a solution made up of three main components. The first one is the technical description of the product and its communication protocol, which serves as an introduction to the system and helps the reader understand the inner workings of a card and how it interacts with a reader. This knowledge is then built upon in the next two components, which are the interaction software and the security protocol sketch and analysis. The software allows a user to control a reader by sending FeliCa native commands to a card and reading responses. This software is then used to model a normal interaction between card and reader. After that, a bruteforcer is built using the command sender in order to establish what kind of security functionality each card supports. The third part of the solution gathers together a number of diagrams detailing the security functions of the cards and some analysis outcomes on the strength of these functions.

1.4 Next sections

The remainder of the Report will describe the analysis methodologies employed alongside the results obtained and the efficiency of the solutions found.

The "Background" section will help the reader get familiar with the FeliCa product, whilst also putting the work in context of other similar endeavours at security analysis of an RFID system, such as the one performed by Bono et al. (2000).

Each of the following three sections denotes one of the main aspects that were involved in the analysis. Thus, the "Technical information" section will sum up the technologies and standards that underly the FeliCa system. Furthermore, this section will provide detailed technical information of each of the cards and readers, alongside a discussion on each of these specifications' relevance to the security of the cards. For example, a very interesting point of discussion is raised by how the structure the cards' file system affects the way cryptographic keys are stored and managed. This section will arm the reader with enough knowledge to understand what goes on in the next sections, titled "Communication protocol" and "Security" respectively.

Felicity cards communicate with readers through a simple, native command based protocol. This protocol along with each of its commands will be explained in the "Communication protocol" section. Since software was created in order to facilitate this communication, this section will also describe the software development practices undertaken and the final product. The section will conclude by discussing the importance of pinning down and implementing the communication between reader and card in a security-oriented context.

The "Security" section will describe the last part of the analysis, putting together information obtained from Security Target documents, JIS and ISO standards and other various sources. It will also describe the reverse engineering done on a number of Windows libraries extracted from Reader driver software provided by Sony. All the information previously obtained is then used in order to sketch a rough approximation of the security protocol that protects communications between card and reader. Each of these three sections follows the same high-level structure, being composed of three main parts: analysis methods, analysis results and the evaluation of the solution.

The final section of the Report is the "Conclusion" one, which summarises all the information found and discusses the security of the FeliCa product in context of all the findings of the analysis. Finally, the last two parts of the Report are the list of References and the Appendix.

1.5 Notations used

Throughout the report, the notation XXh denotes the byte whose hexadecimal notation is XX (so XX as a byte).

Any other notations present in formulas and everywhere else are explained as they are introduced.

Chapter 2

Background

No one seems to have looked at FeliCa academically so far, therefore this is the gap this project will try to fill. However, there has been extensive research in the area of RFID security in general, and other protocols have been thoroughly examined.

A historic overview of RFID is presented by Rieback et al. (2006), ever since its creation in 1940 up to modern day usages, with security pitfalls and measures being highlighted. The first application based upon RFID, the IFF (Identification Friend or Foe) systems was the first one whose security would be breached and this led to Allied planes being shot down. This offers a lot of perspective into the impact a security pitfall in any important RFID protocol could have on society, even more so nowadays when RFID functions as a medium for numerous tasks including managing supply chains, tracking livestock, preventing counterfeiting, controlling building access, supporting automated checkout, developing smart home appliances, locating children, and even foiling grave robbers.

FeliCa has itself seen a lot of important usages so far, especially in Japan, where it is even used in electronic payments (*FeliCa*, no date), therefore it is high time someone looked at exactly how secure this implementation is. Rieback et al. (2006) also provide some techniques academics have come up with in modern days, such as signal jamming or challenge-response identification, but as its been proved many times, an otherwise adequate security measure might still be ineffective if implemented incorrectly. As part of this project, techniques that have been used to ensure FeliCa is secure will be discovered and the correctness of their implementation will be assessed.

Since there hasnt been much research that could be found directly related to the FeliCa implementation, the most relevant literature for this project is the one where attacks against similar protocols are detailed. This is because once its been established how the protocol works, a start in exploiting it would be trying out attacks that others have succeeded with against other implementations. This is not to say every single attack will be blindly thrown at the implementation. Inspiration will be drawn from two other similar endeavours and the methodologies will be adapted to suit the task at hand.

Bono et al. (2000) describe their success against the DST (Digital Signature Transponder), which is an RFID device manufactured by Texas Instruments. The approach to breaking the protocol has was a simple, yet effective one, entailing three main steps:

1. Reverse Engineering
2. Key breaking
3. Simulation

These steps are very similar to the ones that are going to be employed in this project, especially if a valid attack is found to work against FeliCa. The reverse engineering step is the hardest one and perhaps it has been the diligence with which its been carried out by Bono et al. (2000) that has led to such a success in breaking DST. They obtained a rough schematic of the block cipher used in the challenge-response protocol and started performing experiments to observe responses given certain inputs and encryption keys. They are themselves aware that this was the scientific heart of their endeavour and this is why a massive amount of work will be dedicated to this step, in the case of this project. The key length being very small (only 40 bits), they managed to recover a DST key in under an hour from two responses to arbitrary challenges.

This goes to show that if one has enough knowledge of the protocol, it could be easier than expected to find vulnerabilities. After that, they managed to simulate an RF (Radio Frequency) output and to spoof a reader, using the key and serial number of a DST. This led to a simulation whereby a reader was spoofed using a software radio, highlighting the practical significance of this breakthrough. (Bono et al., 2000)

Whilst FeliCa isnt in any way related to DST (apart from the fact that theyre both RFID implementations), this work has significant relevance to the project, in that it shows that with sound methodology and good work ethic one can discover straightforward vulnerabilities (e.g length key being too small). It also offers some practical insight into the inner workings of a real-world RFID protocol (e.g the block ciphers used, the challenge-response protocols etc.) and emphasises the importance of the information gathering step.

This project will build upon similar endeavours of penetrating RFID protocols. The novel and tricky aspect of it is that this particular protocol hasnt been yet examined academically. This final year project will address exactly that gap of knowledge.

Chapter 3

Technical information

The first step in analysing the Sony FeliCa product is understanding how it works internally. This section will present its underlying technologies and will offer an overview of its technical specifications and how they are relevant to the report as a whole.

3.1 RFID

Radio-frequency identification (RFID) uses electromagnetic fields to identify tags that can contain electronic information. Based on where they get their power from, two types of tags distinguish themselves. Thus, the first category is that of passive tags, which get their energy from a nearby RFID reader. The second type of tag is the active one, which has a local source and can operate hundreds of meters away from the reader (*Radio-frequency identification*, no date).

Their main advantage when compared to similar technologies, such as bar codes, is that the tags don't need to be within the line of sight of the reader. This allows tags be embedded on objects e.g access cards or fobs (*Radio-frequency identification*, no date).

RFID has seen an increase in usage as a technology, powering complex systems worldwide, from access control systems to transportation cards or even electronic payments solutions.

3.2 NFC

Near-field communication (NFC) is a set of communication protocols that allows two electronic devices to establish a communication, by placing them not more than 4 centimeters from each other (*Near-field communication*, no date).

It is particularly important to make a distinction between RFID and NFC in order to understand where the FeliCa technology stands in this hierarchy of communication systems. On the one hand, the reader can recall that RFID can uniquely identify items (or tags) using electromagnetic fields. On the other hand, NFC is a specialised subset of the RFID family, more specifically HF-RFID (High Frequency RFID), which operates at a frequency of 13.56 MHz (Thrasher, 2013).

3.3 FeliCa

FeliCa is built on top of the RFID and NFC technologies presented in the previous sub-sections. According to Sony's overview of the product, its main features are:

- **Contactless high speed data transmission**: Each transaction is completed in approximately 0.1 seconds. Being contactless, users don't need to take the card out of wallets of purses in order to use them.
- **Single card for multi-usage**: A single card can offer multiple services simultaneously, so the same card can be used both as an e-payment card and an access card for office entrance.
- **High security**: The system is certified at ISO/IEC 15408 EAL4/EAL4+ security level, an international criteria to measure the security level of a system. The security strength of the product will be tested in the "Security" section of the report.
- **Shape flexibility**: From coins to wrist watches or key holders, FeliCa can be used in a variety of form factors.

(*What is FeliCa?*, no date)

3.4 Types of cards

Based on capacity and usages, the FeliCa cards and tags can be classified into 4 categories:

- **FeliCa Standard**: this is the main FeliCa product, used for applications such as transportation, electronic money and employee identification.

  This category is further divided into 3 more sub-categories, the difference between them being the encryption algorithms supported:

  1. DES/3DES cards
  2. AES cards
  3. cards that support both DES/3DES and AES

- **FeliCa Lite**: this is a minimized product that has an optimized file system and streamlined security functions. This kind of chip can also be placed on tags and stickers.

- **FeliCa Plug**: product with a wireless interface that can be embedded in electronic devices
- **FeliCa Link**: combination between FeliCa Lite and FeliCa plug. This series of products has both wired and wireless interfaces.

(*What is FeliCa?*, no date)

This project concentrates on the FeliCa Standard DES/3DES type of cards.

3.5 Types of readers

The different types of readers differ mainly in what electronic device they need to be attached to and also how the connection is made between reader and its controlling device. Thus, there are wireless and wired readers, computer-driven readers and also readers embedded in mobile phones.

For the purposes of this project, two readers have been used:

- a PC-controlled, wired reader, model RC-S380. This reader was provided with the starter kit from the manufacturer.
- an NFC reader embedded into an Android phone

The first reader proved to be important for the ”Security” section of the project, since some of its driver software has been reverse-engineered. The phone reader was used in the ”Communication protocol” section for sending commands and receiving responses from the cards under analysis.

3.6 File system

As according to the user manual (*FeliCa Card User's Manual Excerpted Edition*, no date), System, Area, Service and Block Data are the building blocks of FeliCa's file system.

Blocks are the lowest level component of the file system. A block is a collection of 16 bytes of data.

Service handless access methods and rights to Block Data. It also stores an authentication key to handle the access rights. There are Service access methods whereby data can be read/written without using this authentication key. There are also services for which mutual authentication using this key needs to be performed before any data read/write can be done. A list of services and whether they require authentication or not can be found in the user manual.

Area is the concept for hierarchically managing Block Data. An Area can register services and change keys for those services. Whenever data needs to be manipulated, an area identifier must be supplied to indicate which area the Block Data resides in.

One single physical card can store more logical cards. This is what powers the multi-usage feature of FeliCa and is done by dividing a card into Systems.

The System and Area created first, at manufacture, are called System 0 and Area 0 respectively. These constitute the ”root” of a FeliCa card’s file system.

The hierarchy that makes up the file system of a FeliCa card is:
System > Area > Service > Block

3.7 Codes description

The Code Descriptions document describes the code types used in the Sony FeliCa technology as follows:

1. **Manufacture ID (IDm)** (8 bytes): serial number of a card, used to identify the target card of communication. This code uniquely identifies a card, or, if there are multiple systems on a card, it uniquely identifies a system on a card, since there is one IDm per system in that case.

2. **Manufacture Parameter (PMm)** (8 bytes): used by a reader/writer to identify the type of chip and performance of a card. The first 2 bytes of the PMm represent the IC Code (2 bytes), which indicates the chip type (e.g a value of FFh FFh indicates a chip that adheres to the JIS X 6319-4:2016 specification). The remaining 8 bytes make up the Maximum Response Time Parameter, a self-explanatory performance measure.

3. **System Code** (2 bytes): value used to identify the System on a card. The Code Descriptions document lists the following possible system codes:

   (a) **12h FCh** - a system that uses the NDEF format. This is a data format used by almost every FeliCa Standard product.

   (b) **40h 00h** - system with NFCF functionality. This is a communication protocol which will be used later to send commands to FeliCa cards.

   (c) **88h B4h** - FeliCa Lite system

   (d) **AAh 00h - AAh FEh** - for system conforming to the JIS X 6319-4:2016 standard

   (e) **FEh 00h** - for system of cards managed by FeliCa Networks inc. directly

   (f) **FEh E1h** - FeliCa Plug system

Any other custom system code could be set by the manufacturer when a card is issued.

A fourth one called Data Format Code has been omitted from this list, since it is only used by FeliCa Plug and FeliCa Lite series, thus being outside the scope of this project.

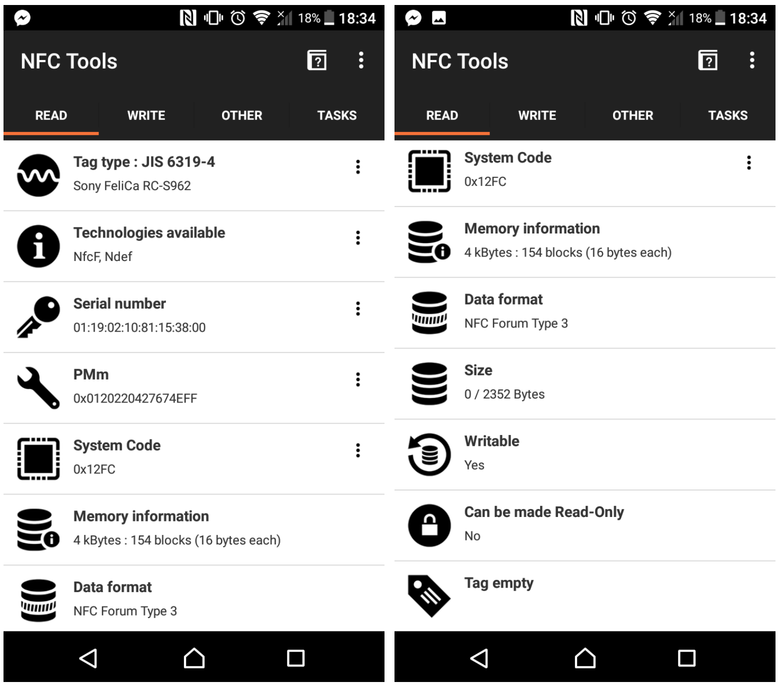
(*FeliCa Technology Code Descriptions*, no date)

3.8 Card case study: DES/3DES blank FeliCa card

So far, it has been shown that the main distinction between different FeliCa Standard cards is given by the type of encryption supported. Thus, there are DES/3DES cards, AES cards and cards that support all of these algorithms.

The first card that was looked at as part of the project was a blank FeliCa RC-S962 card. This card was provided with the starter kit from the manufacturer.

To see the card's specifications, an Android application called NFC Tools was used. Below is the output of this application when this particular card was read using the NFC reader embedded in an Android phone:



Tag type : JIS 6319-4
Sony FeliCa RC-S962

Technologies available
NfcF, Ndef

Serial number
01:19:02:10:81:15:38:00

PMm
0x0120220427674EFF

System Code
0x12FC

Memory information
4 kBytes : 154 blocks (16 bytes each)

Data format
NFC Forum Type 3

System Code
0x12FC

Memory information
4 kBytes : 154 blocks (16 bytes each)

Data format
NFC Forum Type 3

Size
0 / 2352 Bytes

Writable
Yes

Can be made Read-Only
No

Tag empty

Figure 3.1: Android app output for blank FeliCa card

What needs to be remembered from 3.1 is:

- The card uses the JIS 6319-4 standard of communication. This standard will be discussed in greater detail in the "Communication protocol" and "Security" sections, where it is shown how FeliCa implements the standard and what this means in terms of security of the products.

- One of the technologies used is Nfcf. Nfcf will be used to send FeliCa native commands in the section dedicated to communication.

- All other information present in the output shows examples the technical specifications previously discussed in this section, such as the 12h FCh system code and its significance (see Code descriptions).

  As will be shown by a test performed later, in the "Communication protocol" section, this card does not support authentication so it will have little use in the security analysis endeavours presented in the security dedicated section.

3.9 Summary of section

The "Technical Information" section has discussed the technical details of FeliCa, whilst highlighting some of the bits of information that are particularly relevant for the next steps, such as the way memory is mapped.

In the next section, it will be shown how FeliCa developed its own set of byte commands that it uses for communication. An in depth look will be taken at the protocol used and its relevance to the last section, "Security", will be discussed.

Chapter 4

Communication protocol

This section will cover the specifics of the way a card and a reader communicate. It will only touch the unencrypted version of the protocol i.e the one which does not require mutual authentication to be performed beforehand.

All the native commands used to communicate will be detailed using a standardised structure. After that, a piece of software will be proposed that enables this communication via native commands. A full run of the unencrypted protocol will be shown and the reader can refer back to the commands specification to understand what happens at each stage of the interaction. Design decisions and evaluation of the software will also be discussed in the "Communication protocol" section.

4.1 Structure of a command

FeliCa commands are nothing more than collections of bytes. The high level structure of every command is:

Command length (in bytes) — Command packet data,
where the includes itself in the calculation
i.e $Length = 1 + Length(commandpacketdata)$

The Command packet data bit may vary depending on the specifics of the command, but it will always contain an Opcode that uniquely identifies the command being sent.

Each command will be described by mentioning its name and detailing the bytes of its Command packet data. Along with that, the response to the command will be described in a similar fashion, since its structure more or less mirrors the one of its respective command.

4.2 Commands specification

The detailed specification of each FeliCa command can be found in Appendix B - Commands specification. This Appendix is compiled using information present in the user manual (*FeliCa Card User's Manual Excerpted Edition*, no date). Note that for the sake of brevity, only those commands relevant to the purpose of this section (i.e those needed for a full run of the protocol) are included. Also, the length of command/response bit will be omitted as it is required by all commands by default and has been explained in the previous section.

4.3 Command-response software

After documenting the commands, the next logical step was to build a piece of software able to actually send them. Here, multiple design decisions had to be made in order to come up with the best solution:

- What would the reader be? It could either be an NFC-enabled Android phone or the reader provided by Sony in the starter kit.
- What architecture to use? On the one hand, if a phone were to be used as a reader, a modified client-server architecture would be best, whereby the client would be an Android application that just drivers the reader, and the server would be a computer software that sends the commands to the client and outputs the response to the console. On the other hand, using the native FeliCa reader would allow the application to be made up of just a single component, that of the PC software, since the reader offers Windows development support.
- How much automation is required? Should the user be allowed to send single commands, thus relying on her ability to emulate the protocol by sending the right commands at the right time? Or should the whole protocol be automated and the user would just provide the required parameters such as the IDm of the card she wants to interact with?

In order to answer these questions, the initial purpose of the software needs to be recalled. The desired software solution would be the simplest one to develop (since this is just part of the main solution and not the whole aim of the project, a trade-off needs to be made, reducing the complexity of the software as much as possible in order to decrease the time needed to build it and, more importantly, the probability of a malfunction).

The software would mainly be used by myself and potentially by other researchers/developers interested in modelling the protocol and looking at the security of FeliCa, so the highest degree of freedom in terms of how to implement the protocol would be needed.

With these points in mind, a decision was made to opt for the modified client-server architecture, with an Android application controlling the phone's built-in NFC reader in order to send commands coming from the PC server, one at a time. The client and server communicate through raw sockets, so an internet connection is required on both ends - this is perhaps the main drawback of this approach.

Of course, another advantage of this approach is that the one-by-one command sender can easily be built upon in order to develop more complex functionality, as will be shown in a later section, where the use of a brute forcer built on top of the basic command sender is detailed.

4.4 Run of unencrypted protocol

Using the command sender software described in the previous section, one can easily model a full run of the unencrypted protocol. The next step logical of the analysis was exactly this protocol run, which is further detailed in this sub-section. It may prove useful to refer to Appendix B - Command specifications when studying the protocol run-through, as it explains in detail what each byte in commands and responses stands for.

1. **Sending the Polling command**

   Command bytes:

```
06h 00h 12h FCh 01h 03h
```

Response bytes:

```
14h 01h 01h 19h 02h 10h 81h 15h 37h 00h 01h 20h 22h 04h 27h 67h 4Eh FFh 12h
FCh
```

Notes: The card being acquired has IDm 01h 19h 02h 10h 81h 15h 37h 00h and system code 12h FCh.

2. **Sending the Request service command**

   Command bytes:

```
0Dh 02h 01h 19h 02h 10h 81h 15h 37h 00h 01h 09h 00h
```

Response bytes:

```
0Dh 03h 01h 19h 02h 10h 81h 15h 37h 00h 01h 09h 00h
```

Notes: The Random Service without encryption is requested. This is a good one to use when just reading or writing data.

The service exists, since key version sent back is 09h 00h.
It is worth noting that the version is the same as the service code itself, which is usually the case on blank FeliCa cards. This is harmless though, since it is just the key version and not the key itself.

3. Sending a Read without encryption command

Command bytes:

```
10h 06h 01h 19h 02h 10h 81h 15h 37h 00h 01h 09h 00h 01h 80h 00h
```

Response bytes:

```
1Dh 07h 01h 19h 02h 10h 81h 15h 37h 00h 00h 00h 01h 10h 0Ch 0Ah 00h 93h 00h
00h 00h 00h 00h 01h 00h 00h 0Ch 00h C6h
```

Notes: The same service requested in the previous command is used to read the data. The actual data on the block is 10h 0Ch 0Ah 00h 93h 00h 00h 00h 00h 00h 01h 00h 00h 0Ch 00h C6h (last 16 bytes of response).

4. Sending a Request response command

Command bytes:

```
0Ah 04h 01h 19h 02h 10h 81h 15h 37h 00h
```

Response bytes:

```
0Bh 05h 01h 19h 02h 10h 81h 15h 37h 00h 00h
```

Notes: Status flag in response is 00h so a write can proceed.

5. Sending a Write without encryption command

Command bytes:

```
20h 08h 01h 19h 02h 10h 81h 15h 37h 00h 01h 09h 00h 01h 80h 00h 0Ah 0Ah 0Ah
0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah
```

Response bytes:

```
0Ch 09h 01h 19h 02h 10h 81h 15h 37h 00h 00h 00h
```

Notes: The data written is 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah (last 16 bytes of command). Both status flags are 0, so this data has been successfully written.

6. Sending another Read without encryption command (optional step)

Command bytes:

```
10h 06h 01h 19h 02h 10h 81h 15h 37h 00h 01h 09h 00h 01h 80h 00h
```

Response bytes:

```
1Dh 07h 01h 19h 02h 10h 81h 15h 37h 00h 00h 00h 01h 0Ah 0Ah 0Ah 0Ah 0Ah
0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah
```

Notes: This final Read command is sent to ensure that the data that was written in the previous step has actually been written to the block. As can be noticed from the response, the right data has indeed been written successfully.

4.5 Brute-forcing service and area codes

A simple way to check whether a given card supports authentication (and, thereby, encrypted communications) is to look for the presence of a service that requires authentication. As can be recalled from the "System, Service, Area" sub-section, there are services that support authentication (e.g 00h 00h) and services that do not (e.g 17h 01h).

Therefore, in order to test for authentication support, all the services and areas present on the card must be listed. After that, this list can be cross-checked with the one of services that require authentication in order to make a decision.

For this purpose, a brute forcer was built on top of the command server. At a high level, it works in the following way:

- Firstly, it generates all the possible service/area codes (i.e all possibilities of a 2 byte tuple e.g 00h 00h etc.).
- An initial attempt was one which sent one Request service command per generated code, so it was sending 2^{64} commands. The card would cease communication after a reasonably big number of commands, no matter what timeout was used in between commands, so this was not a feasible solution.
- To overcome this problem, the service codes were grouped in tuples of 32 codes each. Recall that the Request service command can request up to 32 services at once and that helped the brute forcer to be smarter and faster. The card did not complain about the $2^{64}/32$ commands being sent and the program ran smoothly after this improvement.
- A response to the command is interpreted as follows: if the key version coming back for a service is something other than 0, then store the service code and key version pair in a text file, otherwise ignore it.

The brute forcer was then put to work against both cards from the previously presented study cases. The program's output is shown in Appendix C - Service code brute forcer output. It was found that the blank FeliCa card does not support authentication, whereas the Octopus one did have authentication-driven services present. This observation will be a valuable one for the "Security" section, since a big part of this section revolves around mutual authentication.

4.6 Software evaluation

The evaluation strategy for the command sender software is based on the following aspects of the code:

- efficiency
- the degree to which the software achieves its initial purpose

An example of a trick that increased the efficiency of the software was presented in the brute forcer sub-section. The only speed requirement of this solution was that it does the job in a reasonable amount of time. This could of course depend upon the user's own judgement, but it also implies that detailed timings of the program's runs (e.g complexity analysis or millisecond time recordings) are not necessary. Rough approximations of the runtime duration of different paths in the program are a better measure to judge the efficiency of this solution. Below are some example measurements recorded on normal runs of the software:

- Sending a single command and receiving the response to it is usually instantaneous.
- Brute forcing a single card for service codes took approximately 3 hours, with some interruptions being caused by the card not allowing any more commands, so the software had to be restarted.

By far, the most important measure involved in the command sender's evaluation is the degree of reliability of the system. A full run of the protocol was implemented successfully using the software. What is more, the brute forcer served its purpose of listing the service codes present on the two test cards without any fault. Ergo, the main goals considered when building the software were accomplished.

4.7 Summary of section

This section described the communication protocol used by Sony FeliCa devices in detail. It also presented a software solution that enables communication between a card and a reader, alongside an extension that allowed the brute forcing of service codes on a card.

The next section, entitled ”Security”, will make use of information from this and the ”Technical information” sections to draw conclusions on the security of FeliCa products.

Chapter 5

Security

So far, the specifications and way of communication of FeliCa cards has been studied. This section takes advantage of all the information gathered so far in order to assess the security strength of the system.

5.1 High level overview of mutual authentication

The most important security functionality provided by Sony FeliCa cards is mutual authentication. Authentication is necessary in the following contexts:

- Communication between card and reader through an encrypted channel
- The reader accessing an authentication-required type of service on the card

The first scenario ensures that the traffic that passes around between card and reader can only ever be read by the parties involved, even if an attacker manages to intercept communications and get the packets. The second one makes sure no unauthorised access to sensitive data occurs, since authentication is required before using services that can manipulate this kind of data. Therefore the goal of mutual authentication is that, once it's been performed successfully, both the card and the reader are sure of each other's identity.

The Sony FeliCa mutual authentication protocol follows JIS X 6319-4, which is a Japanese standard. This standard describes the protocol as being formed of four parts:

1. The reader sends the **Authentication 1** command to the card. In the payload for this command, it has to send **Challenge data 1**. This is an 8 byte parameter that is generated using the **access key** that is already known by the reader and a cryptographic algorithm of the manufacturer's choice. Let this algorithm be called **Algorithm A**.

2. On receiving this command, the card generates **Challenge data 1'**, deriving it from **Challenge data 1** and using cryptographic **Algorithm B**.

   It also generates **Challenge data 2**, using the list of **service and area keys** of the services/areas that the reader wants to access, alongside cryptographic **Algorithm C**.

   Then it sends these 2 newly-generated parameters back to the reader as a payload of the response to the **Authentication 1** command.

3. After getting the response, the reader first needs to verify the challenge response got from the card. It does this by applying **Algorithm D** (which is the exact inverse of **Algorithm B**) to **Challenge data 1'**. If what is obtained is **Challenge data 1**, the reader is sure of the card's identity.

   If the verification fails, the mutual authentication process is halted immediately.

   Otherwise the reader generates its own challenge response, **Challenge data 2'**, by applying cryptographic **Algorithm E** to **Challenge data 2**. It sends this newly-generated parameter as payload of the **Authentication 2** command.

4. On receiving **Authentication 2**, the card applies **Algorithm F** (the exact inverse of **Algorithm E**) to **Challenge data 2'**. If what is obtained is **Challenge data 2**, then the card is sure of the reader's identity and the process of authentication has been completed.

   A **session key** is then obtained from **Challenge data 1** and **Challenge data 2**. Using this **session key**, the card encrypts its serial number and sends it as a response to **Authentication 2**, indicating success.

(*JIS X 6319-4, High Speed proximity cards*, 2005)

The full specifications of the **Authentication 1** and **Authentication 2** commands can be found in Appendix D - Mutual authentication commands specification. This appendix was compiled using information from the JIS document.

5.2 Card case study: DES/3DES Octopus card

The second card that was looked at as part of the project was an Octopus card. Cards of this type are used in public transportation in Hong Kong and Japan. The FeliCa chip model of this card is RC-S915.

Some of the card's specifications that are particularly relevant are:

- the usage of the Nfcf technology

- a system code of 80 08, which cannot be found on the list provided in the Code descriptions sub-section. This is most probably because this system code is a custom one chosen specifically for Octopus cards.
- this is a DES/3DES card

These were obtained from the same Android application used in Case study 1.

Recall that brute forcing the service codes on this card in the previous section led to the conclusion that the card supports mutual authentication (see Appendix C). This is particularly important in that it will allow a further analysis on FeliCa's authentication procedures.

5.3 Reverse-engineering driver DLLs

Armed with the structure of the authentication procedure that was presented in the previous sub-section, the next step was to find out more about the specifics of the FeliCa implementation of this procedure, such as the actual cryptographic algorithms used for each part of authentication.

For this reason, the next thing to do was to look at the software provided in the starter kit. All the PC drivers were installed on a Windows environment. In the installation folders, a number of Dinamically Linked Libraries (DLLs) were found. It was safe to assume those DLLs were used by the reader/writer software, so they needed to be examined in more detail.

The IDA pro software was used to examine one such library called rw.dll (possibly meaning Read/Write). The first thing that stood out was the type of programming language and compiler used to build the DLL: Visual C++. This made it close to impossible to decompile the DLL, so the only thing that could be done was disassemble it using IDA.

Making sense of the disassembled code proved to be a very difficult task, since there were a lot of function calls between different functionality pieces present in the library. This made it very hard to keep track with the data from each register at a given point in the code. Therefore, an approach based on understanding what every single line of assembler code does was not going to be too successful.

Luckily, the developer of the DLL left a lot of logs present in the code, presumably for debugging purposes and the logs proved crucial in reverse-engineering the authentication procedure.

The next thing to do was look at the DLL's exports. Most of the imports were functions that performed a single action in the interaction between card and reader, as can be seen in the figure below:

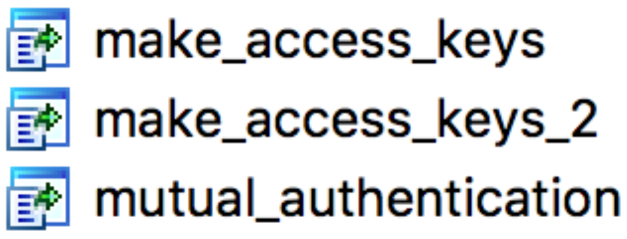
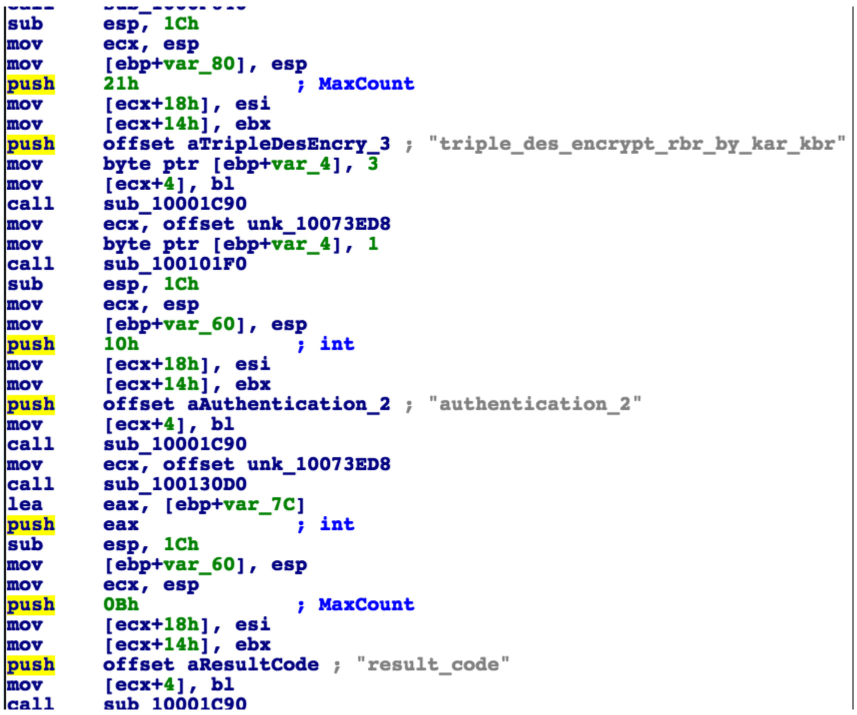

make_access_keys
make_access_keys_2
mutual_authentication

Figure 5.1: Some of the DLL's exports

The functions from 5.1 perform some of the authentication flow, notably generating access keys and starting the authentication process.

After isolating the relevant exports, it was time to look at the assembler code in each of them. Below is an example of such a function:



```
sub     esp, 1Ch
mov     ecx, esp
mov     [ebp+var_80], esp
push    21h              ; MaxCount
mov     [ecx+18h], esi
mov     [ecx+14h], ebx
push    offset aTripleDesEncry_3 ; "triple_des_encrypt_rbr_by_kar_kbr"
mov     byte ptr [ebp+var_4], 3
mov     [ecx+4], bl
call    sub_10001C90
mov     ecx, offset unk_10073ED8
mov     byte ptr [ebp+var_4], 1
call    sub_100101F0
sub     esp, 1Ch
mov     ecx, esp
mov     [ebp+var_60], esp
push    10h              ; int
mov     [ecx+18h], esi
mov     [ecx+14h], ebx
push    offset aAuthentication_2 ; "authentication_2"
mov     [ecx+4], bl
call    sub_10001C90
mov     ecx, offset unk_10073ED8
call    sub_100130D0
lea     eax, [ebp+var_7C]
push    eax              ; int
sub     esp, 1Ch
mov     [ebp+var_60], esp
mov     ecx, esp
push    0Bh              ; MaxCount
mov     [ecx+18h], esi
mov     [ecx+14h], ebx
push    offset aResultCode ; "result_code"
mov     [ecx+4], bl
call    sub_10001C90
```

Figure 5.2: Assembler code for authentication2 function

5.2 shows the code of the function that performs the second step of authentication. The three strings present in the code are log statements. These logs helped to reconstruct what was happening in each of the important authentication functions. The findings that arose from interpreting these strings (from this and other exported functions) are the following:

- The access key is generated using the root (Area 0 and System) keys alongside the list of keys belonging to the areas and services that the reader wants to access after authentication.

- This access key is then split up into two smaller access keys called KAR and KBR respectively.
- The second step of mutual authentication (i.e the one where Challenge data 2' is generated) makes use of a Triple DES encryption using KAR and KBR as keys.
- After authentication is done, two new parameters arise, a session id and a session key. No clue was present as to how these are generated.
- Then, in the procedure that encrypts traffic after authentication, simple DES is used. There was no clue as to what key is used.

These findings, along with information from a Security Target document and what was already known from the JIS document, helped reconstruct the authentication system.

But first, the implications of using those particular cryptographic algorithm need to be understood so the next three sections offer some specifics about the strength of the algorithms used by FeliCa.

5.4 Overview of DES

The Data Encryption Standard (DES) is a symmetric-key cryptographic algorithm used for encrypting and decrypting electronic data. The length of the cryptographic key used is 56 bits.

DES was developed in the 1970s and has since then been proved to be insecure, mainly due to the key size being too small. In January 1999, a collaboration between the Electronic Frontier Foundation and distrubted.net led to a DES key being broken in 22 hours and 15 minutes. After that, the National Institute of Standards and Technology (NIST) decided to withdraw DES as a standard.

(*Data Encryption Standard*, no date)

All in all, using the Data Encryption Standard for any security-critical application is considered bad practice, given its vulnerability to brute force attacks.

5.5 Overview of 3DES

Triple DES (3DES) is a symmetric-key cryptographic algorithm that applies the DES algorithm three times to each data block.

With DES being deemed vulnerable due to its small key size, an improvement had to be made. This is how 3DES was invented. It involves performing two DES encryptions and one DES decryption of each block of data, as follows:

$Ciphertext = E_{K_3}(D_{K_2}(E_{K_1}(Plaintext)))$
where K_1, K_2, K_3 are the 3 cryptographic keys used, E means encryption, D denotes decryption

It follows that decryption of a block of cipher text is done as below:
$Plaintext = D_{K_1}(E_{K_1}(D_{K_3}(Ciphertext)))$
where the same notational conventions apply

The most important thing about Triple DES is that, at the moment, it is a secure cryptographic algorithm and, as such, can be used for real-world applications.
(*Triple DES*, no date)

5.6 Overview of AES

The Advanced Encryption Standard (AES) is the third symmetric-key algorithm used in the security modules of Sony FeliCa.

AES is a subset of the Rijndael family of ciphers with different key and block sizes. NIST selected three members of this family to serve as variants of AES. These have key lengths of 128, 196 and 256 bits respectively.

This standard supersedes DES and not only is it considered highly secure, it is also the most recommended standard of encryption at the moment.
(*Advanced Encryption Standard*, no date)

5.7 Reverse-engineering mutual authentication

It has been demonstrated that FeliCa's mutual authentication procedure has three main steps:

1. Generation of access keys
2. Main mutual authentication
3. Encrypted channel of communication after mutual authentication

The next three sub-sections will detail the steps and security primitives used by each of these processes and will also discuss how an attacker could take advantage of each of those.

5.8 Access key generation

The figure below shows how the two access keys, KAR and KBR are generated:

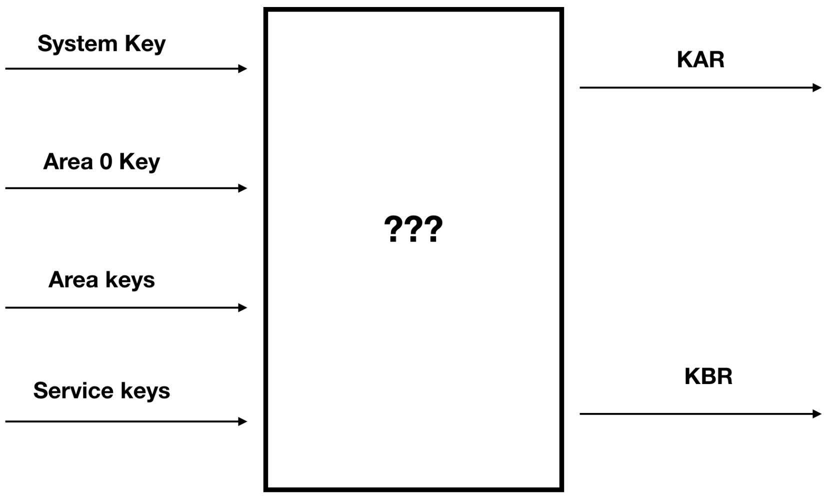


Figure 5.3: Access keys generation procedure

(*CERTIFICATION REPORT No. P165*, 2002)

As can be noticed in 5.3, the exact generation algorithm could not be reverse-engineered, but the input and output of this algorithm are known. The input is made up of the two root keys and the area and service keys of the areas and services that the reader wants to access. The output is formed of the two newly-generated access keys that will be used for mutual authentication.

5.9 Mutual authentication flow

After key generation is done, mutual authentication can commence. This process is nothing more than an implementation of the protocol described in the "High level overview of mutual authentication" sub-section.

A potential attacker's first goal when testing the FeliCa system would be to force mutual authentication in some way. The steps related to the generation and verification of Challenge data 1 and Challenge data 1' are irrelevant from an attacker's point of view. This is because the attacker doesn't care about the reader verifying the card, he just wants the card to acknowledge the identity of the reader. Therefore, just sending 00 bytes as Challenge Data 1 will suffice. Moreover, the verification of Challenge data 1' can be skipped and so the card will assume its own identity has been verified.

The interesting part of the protocol is the generation of Challenge data 2'. This is because the reader needs to generate this data in such a way that it will pass the card's verification. Using information from the Security Target, the JIS document and the section in this report dedicated to reverse engineering the driver DLL, generation of Challenge data 2' can be reconstructed:

$C_2' = E_{KAR,KBR}(C_2)$,
where $E_{KAR,KBR}$ means Triple DES encryption in CBC mode with 2 keys, KAR and KBR

3DES with two keys is an implementation of 3DES where just 2 different keys are used. The third key is usually the same as the first key, so 3DES encryption becomes:

$Ciphertext = E_{K_1}(D_{K_2}(E_{K_1}(Plaintext)))$

Therefore, all the attacker needs to know is the two access keys, KAR and KBR. No evidence was found that shows these keys can somehow be obtained by an attacker. They are not sent in the clear at any point. They are generated using service and area keys that only the manufacturer and the rightful owner of the cards have access to. Also, the 3DES algorithm is considered secure at the moment, so it can be concluded that this part of the protocol has good protection against an attack.

5.10 Encrypted communication after mutual authentication

According to the Security Target document, once mutual authentication has been successfully completed, two new parameters are generated as follows:

- Challenge data 1 becomes session id **ID**
- Challenge data 2 becomes session key **K**

Then every subsequent payload is encrypted as below:

$NewPayload = E_K(ID + OldPayload)$,
where E_K denotes DES encryption in CBC mode with the session key K and
"+" means byte concatenation

(*CERTIFICATION REPORT No. P165*, 2002)

Recall that analysing the communication encryption function from the DLL did not confirm that those two parameters are indeed just Challenge data 1 and Challenge data 2 respectively, but given the findings in the Security Target document, it is highly likely that that is the case. However, what the DLL reverse engineering did confirm is that simple DES encryption is used.

In the next section, the implications of this security flaw will be discussed in greater detail.

5.11 On the security of FeliCa

The findings presented in the previous section make intercepting and decrypting traffic an easy task, which could be performed in one of two ways:

- If the key is indeed Challenge Data 1, then simply use it to decrypt the intercepted traffic
- Otherwise, brute force the 56 bits DES key and use the obtained key to decrypt the traffic

  In any case, this bit of functionality is vulnerable. All the other parts (e.g mutual authentication, key generation) are considered secure as far as this report goes, since the analysis has not yielded a way to break them.

  AES based FeliCa has not been looked at as part of this project, but given the strength of the underlying encryption algorithm, it is probably a secure system.

  The implications of this possible attack to DES FeliCa are not necessarily as grave as they may first seem. This is because, even if an attacker has the decrypted traffic between a card and a reader, there still isn't much that can be done with this information.

  For example, let us say an attacker is able to intercept traffic between an Octopus card and a transport barrier. If the card is recognised by the barrier's reader and it is still valid for travel (whatever that may imply), then the barrier is lifted and the card holder can pass. Let us assume the attacker decrypts the traffic passed between card and reader and is able to find out whatever information the card transmits to the barrier in order to be accepted (e.g this could imply the card sending its current balance and if it is greater than 0 it means it is still valid for travel). When the attacker will try a Replay attack i.e will attempt to craft a FeliCa card that sends the exact same data to the barrier reader, this attempt will fail since mutual authentication has not been performed between the attacker's card and the reader.

Therefore, as grim as the conclusion about DES FeliCa may sound, it doesn't necessarily make any practical implementation vulnerable to an attack that would compromise the system.

Furthermore, DES/3DES cards have been and will continue to be replaced by AES ones on a regular basis, but it is not clear how many potentially vulnerable cards are still in use at this moment. The DES/3DES Octopus card presented under Case Study 2 was still a good example of a card technology that is still in use in Hong Kong.

5.12 Summary of section

In this section, information obtained from standards, Security Target documents and driver code analysis has been used to compile a schematic presentation of how security of DES/3DES FeliCa cards works.

Chapter 6

Discussion

Overall, this project has achieved its initial purpose, which is that of putting together an analysis of the Sony FeliCa smart card system. Detailed technical information has been successfully gathered. Furthermore, a piece of software has been written that allows an user to control the interaction between the reader and a card. Last but not least, diagrams have been compiled that depict the security protocols used for this technology and, based on the diagrams, a short assessment of the strength of the security functions has been carried out.

The main deficiency of this project is the lack of an actual real-world attack implementation against the security of the card. This was mainly due to the lack of time in the end, but the fact that I didn't have any knowledge about the cards' pre-set access keys was also an impediment. It is also worth reiterating that the whole approach was a black-box one, with little to no information about key aspects - such as mutual authentication - known beforehand.

I believe what is left to do is put some more effort into finding out exactly how mutual authentication is performed so that this can be reproduced and then attacked. The first thing I would do once mutual authentication is possible is intercept the traffic that is supposed to be encrypted after this step. Since I am fairly certain the payloads can be easily decrypted, given the lack of strength of the encryption algorithm used (DES), I would then try to recover the plaintext of these payloads. Then, an attack would have been successfully completed and it could be proved that FeliCa DES/3DES is indeed insecure.

I am happy with the results, since it is admittedly more than I was hoping to find out, especially when it comes to the security bits. I am also very content with the collaboration of my supervisor and fellow supervisees, since the discussions we had on a regular basis always helped refreshing my perspectives on the work I was doing. It was a great experience that exposed me to new, challenging aspects of Computer Science and encouraged me to use my already existing knowledge in novel, creative ways.

Chapter 7

Conclusion

The solution described in this report addresses the problem of researching and analysing the FeliCa system in three different ways. Firstly, it documents all technical aspects and inner workings of the technology. Then, it documents and implements the communication protocol. Finally, armed with the knowledge from the previous two steps, a brief overview of the security functions of FeliCa is provided.

Based on the results of this analysis, it can be concluded that DES/3DES based FeliCa is very likely to be vulnerable to at least one type of attack (decrypting traffic that is supposed to be read only by card and reader).

AES based FeliCa has not been looked directly at as part of this report, but given the proven strength of the AES algorithm, it could be argued that cards of this type are secure.

Chapter 8

References

1. Bono, S., et al. (2005) *Security Analysis of a Cryptographically-Enabled RFID Device.* USENIX Security Symposium. Vol. 31. Available at `https://www.usenix.org/legacy/events/sec05/tech/bono/bono.pdf` (Accessed: 26 March 2018)

2. 'What is FeliCa?' (no date). Available at `https://www.sony.net/Products/felica/about/index.html` (Accessed: 27 March 2018).

3. Rieback, M., et al. (2006) *The Evolution of RFID Security.* Available at `http://www.targeted-individuals.co.uk/rfid_implants_36_rfid_security_evolution.pdf` (Accessed: 26 March 2018)

4. 'Radio-frequency identification' (no date) Wikipedia. Available at `https://en.wikipedia.org/wiki/Radio-frequency_identification` (Accessed: 27 March 2018).

5. 'Near-field communication' (no date) Wikipedia. Available at `https://en.wikipedia.org/wiki/Near-field_communication` (Accessed: 27 March 2018).

6. Thrasher, J. (2013) *RFID vs. NFC: What's the Difference?* Available at `https://blog.atlasrfidstore.com/rfid-vs-nfc` (Accessed: 27 March 2018).

7. *FeliCa Card User's Manual Excerpted Edition* (no date). Available at `https://www.sony.net/Products/felica/business/tech-support/st_code.html` (Accessed: 27 March 2018).

8. *FeliCa Technology Code Descriptions* (no date). Available at `https://www.sony.net/Products/felica/business/tech-support/st_usmnl.html` (Accessed: 27 March 2018).

9. *JIS X 6319-4, High Speed proximity cards* (2005).

10. 'Data Encryption Standard' (no date) Wikipedia. Available at `https://en.wikipedia.org/wiki/Data_Encryption_Standard` (Accessed: 29 March 2018).

11. 'Triple DES' (no date) Wikipedia. Available at `https://en.wikipedia.org/wiki/Triple_DES` (Accessed: 29 March 2018).

12. 'Advanced Encryption Standard' (no date) Wikipedia. Available at `https://en.wikipedia.org/wiki/Advanced_Encryption_Standard` (Accessed: 29 March 2018).

13. *CERTIFICATION REPORT No. P165* (2002). Available at `https://www.commoncriteriaportal.org/files/epfiles/CRP165.pdf` (Accessed: 29 March 2018).

Appendix A

Structure of submitted zip file

The submitted zip contains two files:

1. **repo.txt**: address of GIT repository used for the project
2. **files.zip**: archived folder containing all the files and code for this project, as follows:
   (a) **Proposals**: any file with the word proposal in its name is related to the Project Proposal made in the beginning
   (b) **Notes**: the notes.pages file contains all of my notes recorded throughout the project
   (c) **Literature review**: the any file with the word litreview in its name is related to the Literature Review previously submitted for this project
   (d) **Presentation**: the presentation file refer to the project presentation done at the end of first term
   (e) **demo folder**: this folder refers to the project demonstration done at the end of the second term
   (f) **binaries from libraries folder**: all binaries that were retrieved from libraries used by starter kit software
   (g) **bruteforce folder**: a copy of the brute forcer's output files
   (h) **jis folder**: copies of the JIS standards relevant to the project
   (i) **FelicaCommander folder**: contains the client Android application code
   (j) **FelicaCommandSender folder**: contains the server PC software code
   (k) **other slides folder**: powerpoint presentations relevant to the project
   (l) **felica lite tech spec folder**: technical documents for FeliCa Lite

Appendix B

Commands specification

B.1 Polling

a) Purpose: to acquire a card
b) Command packet data breakdown:

- *Opcode*: 00h (1 byte)
- *System code*: 2 bytes (see Codes description). This is the system code of cards the reader wants to acquire.
- *Request code*: 1 byte. Several pieces of information can be requested from cards that respond to this command. This payload is usually 01h, meaning the system code of the card is requested.
- *Time slot*: 1 byte (part of the anti-collision strategy used by FeliCa, which is outside of the scope of this report)

c) Response packet data breakdown:

- *Opcode*: 01h (1 byte)
- *Card IDm*: 8 bytes (see Codes description)
- *Card PMm*: 8 bytes (see Codes description)
- *System code*: 2 bytes - as requested in the command

B.2 Request service

a) Purpose: check if a service is present on a card
b) Command packet data breakdown:

- *Opcode*: 02h (1 byte)
- *Card IDm*: 8 bytes.
- *Number of nodes (services) requested*: 1 byte.
- *List of nodes (services)*: 2*n bytes, where n is the number of nodes. This list contains the service codes for all the services requested. Note a service code is made up of 2 bytes.

c) Response packet data breakdown:

- *Opcode*: 03h (1 byte)
- *Card IDm*: 8 bytes
- *Number of nodes (services) requested*: 1 byte.
- *List of key versions*: 2*n bytes, where n is the number of nodes. This list contains the key versions (2 bytes each) of each service requested. If a service does not exist, its key version will be 00h 00h.

B.3 Read without encryption

a) Purpose: to read blocks of memory from a card
b) Command packet data breakdown:

- *Opcode*: 06h (1 byte)
- *Card IDm*: 8 bytes
- *Number of nodes*: 1 byte
- *List of nodes*: 2*n bytes, n is number of nodes (see previous command)
- *Number of blocks to be read*: 1 byte
- *Block list element (first index of block)*: 1 byte
- *Block number (second index of block)*: 1 byte

c) Response packet data breakdown:

- *Opcode*: 07h (1 byte)
- *Card IDm*: 8 bytes

- *Status flag 1*: 1 byte. This flag is 00h if the operation succeeded . Any other value denotes a failure.
- *Status flag 2*: 1 byte (same as above)
- *Block data*: 16*n bytes, where n is the number of blocks to be read. This is the data being read from the memory blocks.

B.4 Request response

a) Purpose: to check the connection between card and reader is still active. It is usually sent before sending a write, in order to ensure that write can actually happen.
b) Command packet data breakdown:

- *Opcode*: 04h (1 byte)
- *Card IDm*: 8 bytes

c) Response packet data breakdown:

- *Opcode*: 05h (1 byte)
- *Card IDm*: 8 bytes
- *Status flag*: 1 byte. Again, 00h means OK whereas anything else indicates a fault.

B.5 Write without encryption

a) Purpose: to write data directly to memory blocks on the card
b) Command packet data breakdown:

- *Opcode*: 08h (1 byte)
- *Card IDm*: 8 bytes
- *Number of nodes*: 1 byte
- *List of nodes*: 2*n bytes, n is number of nodes
- *Number of blocks to be written to*: 1 byte
- *Block list element (first index of block)*: 1 byte
- *Block number (second index of block)*: 1 byte

- *Block data*: 16*n bytes, where n is the number of blocks to be read. This is the data being written to the memory blocks.

c) Response packet data breakdown:

- *Opcode*: 09h (1 byte)
- *Card IDm*: 8 bytes
- *Status flag*: 1 byte. A status flag of 00h indicates the write has succeeded. Any other value indicates the write has failed.

Appendix C

Service code brute forcer output

Note: The service codes followed by an asterisk (*) belong to services that require authentication.

C.1 Blank FeliCa card

```
Service (area) code: 00:00 - key version: 23:01
Service (area) code: 09:00 - key version: 09:00
Service (area) code: 0B:00 - key version: 0B:00
```

C.2 Octopus card

```
Service (area) code: 00:00 - key version: 07:00 (*)
Service (area) code: 00:08 - key version: 07:00 (*)
Service (area) code: 08:02 - key version: 07:00 (*)
Service (area) code: 08:03 - key version: 07:00 (*)
Service (area) code: 08:07 - key version: 07:00 (*)
Service (area) code: 08:09 - key version: 07:00 (*)
Service (area) code: 08:0A - key version: 07:00 (*)
Service (area) code: 08:0B - key version: 07:00 (*)
Service (area) code: 0A:02 - key version: 07:00 (*)
Service (area) code: 0A:03 - key version: 07:00 (*)
Service (area) code: 0A:07 - key version: 07:00 (*)
Service (area) code: 0A:09 - key version: 07:00 (*)
Service (area) code: 0A:0A - key version: 07:00 (*)
Service (area) code: 0A:0B - key version: 07:00 (*)
Service (area) code: 0C:04 - key version: 07:00 (*)
Service (area) code: 0E:04 - key version: 07:00 (*)
Service (area) code: 10:01 - key version: 07:00 (*)
Service (area) code: 12:01 - key version: 07:00 (*)
Service (area) code: 14:01 - key version: 07:00 (*)
Service (area) code: 17:01 - key version: 07:00
```

Appendix D

Mutual authentication commands specification

D.1 Authentication 1

a) Purpose: to perform the first part of the mutual authentication process
b) Command packet data breakdown:

- *Opcode*: 10h (1 byte)
- *Card IDm*: 8 bytes
- *Number of areas accessed*: 1 byte (n)
- *List of areas codes*: 2*n bytes
- *Number of services accessed*: 1 byte (n)
- *List of services codes:* 2*n bytes
- *Challenge data 1*: 8 bytes

c) Response packet data breakdown:

- *Opcode*: 11h (1 byte)
- *Card IDm*: 8 bytes (see Codes description)
- *Challenge data 1'*: 8 bytes
- *Challenge data 2*: 8 bytes

D.2 Authentication 2

a) Purpose: to perform the second part of the mutual authentication process
b) Command packet data breakdown:

- *Opcode*: 12h (1 byte)
- *Card IDm*: 8 bytes.
- *Challenge data 2'*: 8 bytes

c) Response packet data breakdown:

- *Opcode*: 13h (1 byte)
- *Card IDm*: 8 bytes
- *Communication ID*: ? bytes
- *Issuer ID*: ? bytes
- *Issue Parameter*: ? bytes

Appendix E

How to run the code

Pre-requisites:

1. Hardware:
   (a) An Android phone with an embedded NFC reader
   (b) A PC that can run java 8
   (c) Both the PC and the phone should be connected to the same network. No firewalls or port forwarding should be able to stop any kind of socket communication between them.
   (d) USB cable to send the APK of the Android app to the phone
2. Developer environment (recommended):
   (a) Eclipse IDE: for running the PC Server
   (b) Android Studio: for building the Android APK for the app and sending it to the phone
   (c) Java 8

Running the code:

1. The PC server:
   (a) Load the FelicaCommandSender folder into Eclipse as a new Java project
   (b) Run the class Server.java as a Java application
   (c) If everything is ok, an IP address should be output. Make a note of this address. It will be needed later.
2. The Android app:

   (a) Load the FelicaCommander folder in Android Studio as a new Android Studio project. Go with the default settings when asked.
   (b) Using the code editor, go inside the Communication.java class and edit the lines at the top containing the server's IP address and port to reflect the ones of your already started server. The IP output by the server is the one that needs to be used here. The port can remain unchanged (9800).
   (c) Build and send the APK to the phone connected via USB.

3. The software as a whole:
   (a) With the server fired up, open up the newly installed application on the phone
   (b) Approach a FeliCa card/tag to the phone. Communication should be started automatically and the server's console should now output the user's options.
   (c) Now the user can choose between sending a single command to the card or firing up the brute forcer.
   (d) Note that in order for the brute forcer to work, it needs the input/output files to already be created. They are already present in the zip archive submitted, in the right place (the src folder of the Java project).