Jacob Fenton

Efficient Asymmetric Cryptography for RFID Access Control

Computer Science Tripos – Part II

Fitzwilliam College

April 12, 2018

Proforma

Name:	**Jacob Fenton**
College:	**Fitzwilliam College**
Project Title:	**Efficient Asymmetric Cryptography for RFID Access Control**
Examination:	**Computer Science Tripos – Part II, 2018**
Word Count:	**TODO**
Project Originator:	Dr Markus Kuhn
Supervisor:	Dr Markus Kuhn

Original Aims of the Project

The objective of the project was to produce an access control system that implements an authentication protocol based on asymmetric key cryptography. As well as producing a card application and reader application that implement the authentication protocol, a card-provisioning application that can be used to issue new cards or reprogram existing cards was to be written. The system was required to be able to authenticate a card in less than one second.

Work Completed

I have devised and implemented a basic authentication protocol, which includes writing applications to run on both the card and the reader. This basic protocol is able to authenticate a card in under one second, as specified by the success criteria in the project proposal[1]. As an extension, I've also implemented a more advanced protocol based on the Open Protocol for Access Control Identification and Ticketing with PrivacY (OPACITY) [4] with Full Secrecy (FS), which provides user untraceability and mutual authentication. Finally, I've written the command-line card-provisioning application, which is used for (re)programming cards.

[1] See Appendix A.

Special Difficulties

None.

Declaration

I, Jacob Fenton of Fitzwilliam College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed Jacob Fenton

Date April 12, 2018

Contents

List of Figures

Chapter 1

Introduction

1.1 Smart-cards

In recent years, contactless smart-cards (also known as integrated circuit cards, or ICCs) have become extremely popular for a great number of applications. Perhaps one of the first major applications was by Transport for London, whereby contactless "Oyster cards" could be used in place of paper tickets for travel on buses and the underground railway system. As they've become more affordable and the development ecosystem has expanded, their use has become even more widespread. In the UK, contactless bank cards are commonplace[1], and the University of Cambridge distributes contactless university cards to students, which can be used to gain access to colleges/departments. Some colleges even allow students to pay for food and drink with these cards.

Contactless smart-cards come in two kinds: memory cards and microprocessor cards. Both types have an antenna and some form of memory, with microprocessor cards also having a CPU. Thus, memory cards have no ability to process data on-card. Neither type of card has a battery. Instead, the card is powered by the card reader through radio-frequency induction. As a result, smart-cards are very low power and complex computations take significantly longer than they would on an inexpensive laptop or phone. Communication between the card and card reader also occurs over radio frequency.

1.2 MIFARE Classic

The current University of Cambridge access control system is based on the MIFARE Classic smart-card, a memory smart-card which conforms to ISO 14443-A [2], a standard for contactless ICCs to communicate with a "coupling device" (i.e. a smart-card reader) over radio frequency. There are huge numbers of this particular card in existence — over 200 million are in use today.

The cryptographic protocol used in the card, a scheme named CRPYTO-1, was developed in-house by the manufacturer of the MIFARE Classic, NXP Semiconductors. NXP

[1]Strictly speaking, these are dual interface cards, as they have an electrical contact located on the outside of the card which can be used for a hard connection, as well as the internal antenna which allows for contactless communication.

chose to keep the scheme secret, a practice known as security by obscurity. Such practice is eschewed by the security community because naturally, all cryptographic schemes are bound to have weaknesses, and if researchers (or others) are not able to analyse a scheme, these weaknesses will likely go unnoticed, leaving them open to exploitation by anyone without pure intentions. Furthermore, obscurity does not prevent others from deducing the details of the scheme by observing it in operation and indeed this was the case for CRYPTO-1. In December 2007, a presentation at the Chaos Communication Congress (an annual security conference) by two German researchers, Nohl and Plötz, described a partial reverse engineering of CRYPTO-1, as well as some weaknesses. They managed to do this by reconstructing the card's electronic circuit from photos of the chip. They then verified their reconstruction by eavesdropping on the reader-card communication. Just a few months later, in March 2008, researchers in the Digital Security group at Radboud University Nijmegen revealed a complete reverse engineering of the scheme and were able to clone and manipulate the contents of a MIFARE Classic card. The most serious attack they detailed in their paper can recover the card's cryptographic key in under a second using only a laptop, without any pre- computation.

1.2.1 Use of Symmetric Key Cryptography

Ignoring the fact that the CRYPTO-1 scheme is inherently flawed, NXP's choice to use symmetric key cryptography for the MIFARE Classic was perhaps a misstep. Symmetric ciphers utilize what is called a shared secret, or secret key. Two parties wanting to communicate will first exchange this key over a secure channel and then use it to encrypt/decrypt messages sent between them. In the case of access control cards, this means that a card will store just one key, its own secret key, but that key will be stored in every door reader to which the card has access. This means that if a door reader is compromised, and the attacker is able to retrieve all the keys stored within, then they're able to clone any card which had access to that door. If this door is not in a very specific department, then many people will have access to it, and thus the attacker will have access to a very diverse set of doors — essentially, the entire system is compromised.

Such a weakness does not exist when using a scheme based on asymmetric key cryptography, in which each card has not one but two keys — one public, one private. The private key is known only to the owner and is never sent over any channel, whilst the public key is known to everyone. If two parties wish to communicate, then they encrypt their messages with each other's public keys. The message can now only be decrypted with the recipient's private key, which only the recipient knows. In this case, the door reader contains only a long list of public keys corresponding to all the cards that can access the door. Thus, an attacker who's able to compromise a reader doesn't learn any secret information except for the door's private key. This only allows them to clone that specific door reader and doesn't compromise any other cards or readers in the system.

The tradeoffs here are as follows:

- Symmetric key cryptography is computationally much more simple and, therefore quicker than asymmetric key cryptography.

- Symmetric key cryptography involves *shared* secrets, meaning a wider potential attack vector and more vulnerable system than one using asymmetric key cryptography.

1.2.2 Memory Structure

Before discussing the vulnerabilities of MIFARE Classic, it helps to understand the memory structure. Memory is divided into data blocks of 16 bytes, with blocks being grouped into sectors. The first block of the first sector contains special read-only data. The first 4 bytes contain the unique identifier of the card (UID), followed by a 1-byte *bit count check* (BCC). The rest of the block stores manufacturer-specific data.

The last data block in every sector, known as the *sector trailer*, contains two secret keys A and B which are used for authenticating to a sector. Once authenticated to a sector, the card reader may perform operations on it, depending on the *access conditions* which are also defined in the sector trailer. Access conditions are defined for both keys, to allow different levels of access depending on which key is used for authentication, e.g. key A may have read/write permissions whilst key B may only have read permissions. A diagram of the memory structure of a MIFARE Classic card is shown in Figure 1.1.

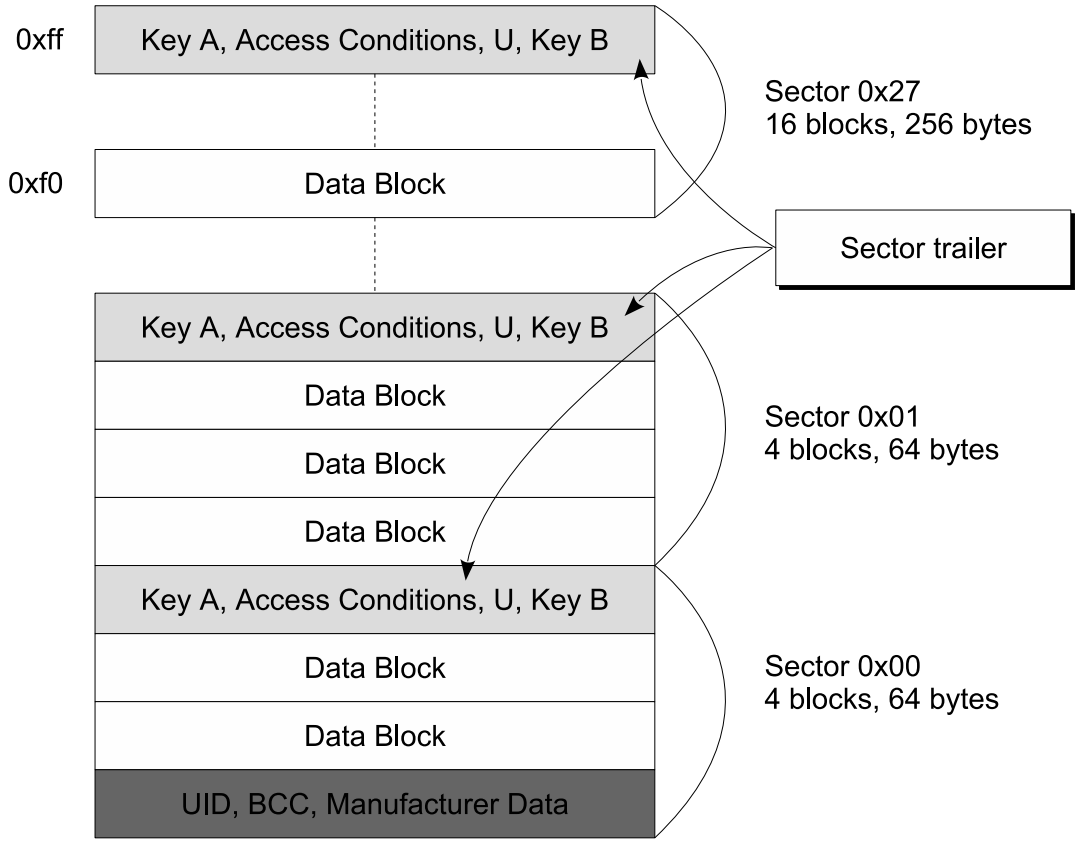

Figure 1.1: MIFARE Classic memory organisation

The sector trailer itself has specific access conditions. Key A is never readable and key B can be configured to be readable or not. In the latter case, the sector is used just for data storage and only key A can be used to authenticate to the sector. Besides the

keys and access conditions, there's one data byte (U) remaining, which has no defined purpose. A diagram of the sector trailer is shown in Figure 1.2.

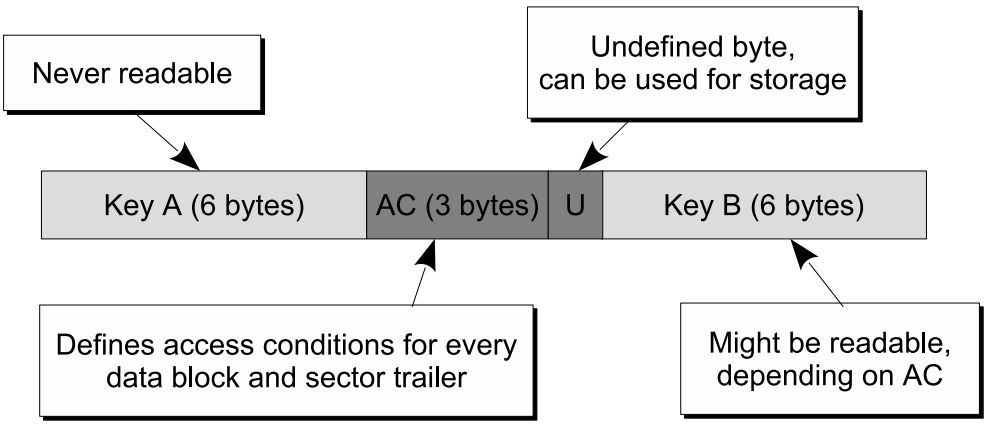

Figure 1.2: Sector trailer

1.2.3 Vulnerabilities

A number of weakness and vulnerabilities have been discovered in the MIFARE Classic system, including brute-forcible keys, a predictable pseudo-random number generator (PRNG), a cloning attack, and two very powerful attacks named the nested authentication attack and the Dark-Side attack. We shall discuss these vulnerabilities in the following paragraphs.

Brute-forcing the sector keys

As stated above, sector keys (A and B) are only 48 bits long, meaning they're vulnerable to a brute-force attack. This involves trying all 2^{48} possible keys one-by-one until the correct key is found. It has been shown that this can be done on dedicated hardware[2] in a matter of hours [6].

Predicting the output of the card's PRNG

A PRNG is supposed to generate a sequence of numbers that are effectively indistinguishable from random or, put another way, an adversary should not be able to guess the next number in the sequence. In MIFARE Classic, the on-card PRNG plays an important role in the sector authentication protocol, CRYPTO-1. Specifically, it turns out that if we can predict the output of the PRNG, we're able to authenticate to sectors for which we do not know either of the keys — this is the nested authentication attack, which is explained in the next paragraph. It was discovered that the output of the MIFARE Classic PRNG depends only on the amount of time that has elapsed since the card was powered up by the reader and that the PRNG has a period of 0.618s. Thus, its output is entirely predictable, making the following attack possible.

[2]FPGAs or GPUs.

Nested authentication attack

This attack allows an adversary to obtain all the keys for every sector on the card if just one key (A or B) is known for any sector. A readily available list of known default keys for MIFARE Classic cards exists and, unfortunately, many systems do not update these keys before issuing cards. When testing my own University of Cambridge card, I discovered that five default keys were in use.

When a reader attempts to authenticate to a sector on a card, the card produces a nonce, generated by the weak on-card PRNG, which it sends to the reader. Both the card and reader then derive some key stream bits using this nonce and the key for the sector (one of A or B). The trick here is that, once the reader has authenticated to a sector, subsequent authentication requests are encrypted. Furthermore, upon receiving an authentication request for a new sector, the card sets the internal state of the cipher to the key for the new sector. Thus, when the card sends the *encrypted* nonce for a subsequent authentication request, since we are able to predict the nonce value, we can discover the key stream bits — the first 32 bits of the sector key — used to encrypt it by XORing the encrypted nonce with our prediction. Now we can easily brute-force the last 16 bits of the key.

Dark-Side attack

This attack allows an adversary to recover any key for any sector without any prior knowledge of any keys. During the authentication protocol, there's a single step where the reader sends encrypted data to the card, along with 8 parity bits. Parity bits are used in communication as a method for error detection/correction. It was noticed that if random data was sent by the reader, then with probability 1/256, the card will respond in an unexpected way. The reason for this is that if any parity bits are wrong, the card won't respond at all. However, if all the parity bits are correct, but the corresponding data sent by the reader is not correct[3], the card responds with a 4-bit error code, 0x5 (NACK), which is encrypted. Since the plaintext is known, the attacker can recover the four key stream bits used to encrypt the NACK. By repeating this procedure many times, the internal state of the cipher can be discovered, allowing the attacker to completely reconstruct the sector key.

Combined attack

The combination of the nested authentication and Dark-Side attacks allows an adversary to completely recover all the keys from any card. They can first run the nested attack to determine if any default keys are in use. If this is not the case, they instead run the Dark-Side attack to recover just one key for any sector. With this one key, they can then rerun the nested attack to recover all other keys.

[3]The details of what a "correct" response is are specific to the authentication protocol. See [5] for more information.

Cloning cards

Once all the keys have been recovered from a card, all the data can be read from it and written to another card, producing a clone. There's one slight issue here, however. As mentioned earlier, the first block of the first sector of a MIFARE Classic card contains a special piece of **read-only** data — the UID. On legitimate MIFARE Classic cards, this value is burned onto the card during manufacturing, and is indeed unchangeable. However, there exist unofficial cards which emulate the MIFARE Classic system and have a programmable UID[4]. With one of these cards, an adversary is able to create an exact clone of any legitimate card.

[4]Such as the Fundan Microelectronics FM11RF08.

Chapter 2

Preparation

2.1 Starting point

This project makes significant use of the material from Security I and Security II. Further, material from Object-Oriented Programming and Further Java was utilised when writing the smart card application that runs on the JavaCard platform, which supports a subset of the Java language.

A previous attempt was made at this project by Denys Natykan, and so his dissertation deserves mention. However, there're rather significant differences in our end products, as I opted to implement a different authentication protocol in order to complete certain extensions.

2.2 Communication between card and reader

The format for communication between a card and a reader is defined by ISO 7816-4 [3]. It states that the basic unit of communication is the application protocol data unit (APDU). There are two categories of APDUs: commands APDUs and response APDUs. The reader is always the party that initiates communications and thus only ever sends command APDUs. Correspondingly, the card always waits for commands from the reader and thus only ever sends response APDUs.

2.2.1 Command APDU

Command APDUs consist of a required 4-byte header and an optional body. The header contains the following four elements, each 1-byte long:

- CLA — indicates the class of command, interindustry or proprietary.
- INS — instruction code, indicates the specific command, e.g. "initiate authentication".
- P1, P2 — instruction parameters, these are command specific.

The optional body contains the following three elements:

- L_c — a single byte encoding the number (N_c) of bytes of command data to follow.
- Command data — N_c bytes of data.
- L_e — a single byte encoding the maximum number (N_e) of response bytes expected.

If the body is included, parts can be left out depending on the requirements, e.g. if the command has no data, but a response is expected, only the L_e field will be present. Similarly, if the command has data but doesn't expect a response, only the L_c and command data fields will be present. A diagram of the structure of a command APDU is shown in Figure 2.1.

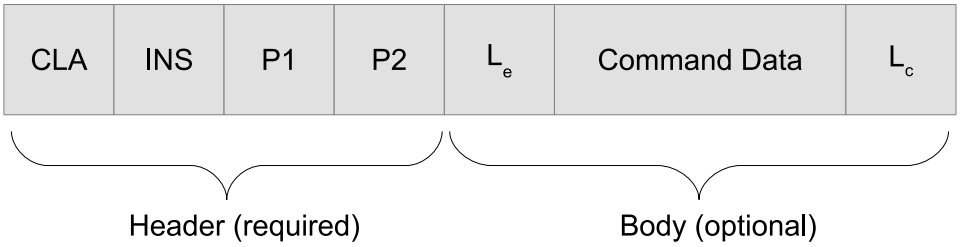

Figure 2.1: Command APDU

2.2.2 Response APDU

Response APDUs consist of an optional body and a required 2-byte trailer. The optional body contains any response data the card wishes to send to the reader, and should not exceed N_e bytes in length. The trailer contains two status bytes, SW1 and SW2, which indicate the command processing status (e.g. 0x9000 indicates no error, command success, whereas 0x6A80 indicates wrong data). A diagram of the structure of a response APDU is shown in Figure 2.2.

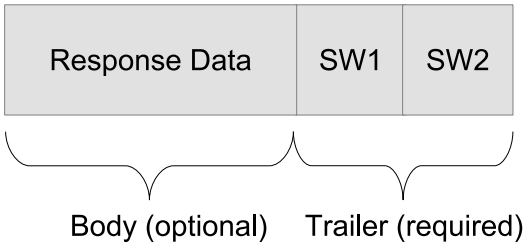

Figure 2.2: Response APDU

2.3 JavaCard

The JavaCard platform allows Java-based applets to be run securely on smart-cards. The choice to use a smart-card that implemented JavaCard was made for a few reasons. Firstly, the development ecosystem is very mature — Oracle make available extensive documentation and resources, perhaps most importantly of which are detailed API references.

2.4 Resources

Before I was able to begin the implementation stage, I ordered the following items:

- Identiv SCL3711 — A USB smart-card reader. I had originally intended to use this reader, but discovered that, despite the claims of the manufacturer, it was not compatible with my operating system (OSX 10.11, El Capitan).
- ACS ACR122T — alternative USB smart-card reader that was compatible.
- NXP J3A040 — a programmable smart-card with 40K of EEPROM memory, implementing JavaCard 2.2.2 and Global Platform 2.1.1, and conforming to ISO 14443. I chose this card because it is relatively cheap and it implements many of the cryptographic algorithms that I needed in order to implement an authentication protocol based on asymmetric cryptography[1]. Note that cards may support some algorithms but only for certain key sizes. For example, the J3A040 only supports ECDSA keys of up to 192 bits. Details of what algorithms and key sizes a card supports can be found by using JCAlgTest [1].

[1] E.g. support for ECDSA (and AES, which is used in the advanced protocol that I implemented).

Chapter 3

Implementation

3.1 Verbatim text

3.2 Tables

3.3 Simple diagrams

3.4 Adding more complicated graphics

Chapter 4

Evaluation

4.1 Printing and binding

4.2 Further information

Chapter 5

Conclusion

Bibliography

[1] JCAlgTest. https://github.com/crocs-muni/JCAlgTest.

[2] ISO/IEC 14443. Identification cards — Contactless integrated — circuit(s) cards — Proximity cards. Standard, International Organization for Standardization, April 2000.

[3] ISO/IEC 7816-4. Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange. Standard, International Organization for Standardization, January 2005.

[4] ActivIdentity. Open protocol for access control identification and ticketing with privacy, July 2011. https://www.securetechalliance.org/resources/pdf/OPACITY_Protocol_3.7.pdf.

[5] Nicolas T. Courtois. The dark side of security by obscurity, March 2009. https://eprint.iacr.org/2009/137.pdf.

[6] Cihangir Tezcan. Brute force cryptanalysis of MIFARE classic cards on GPU, January 2017. https://www.researchgate.net/publication/315848177_Brute_Force_Cryptanalysis_of_MIFARE_Classic_Cards_on_GPU.

Appendix A

Project Proposal

Computer Science Tripos – Part II – Project Proposal

Efficient Asymmetric Cryptography for RFID Access Control

J. Fenton, Fitzwilliam College
Originator: Dr M. Kuhn
11 October 2017

Project Supervisor: Dr M. Kuhn
Director of Studies: Dr R. Harle
Project Overseers: Dr S. Holden & Dr N. Krishnaswami

Introduction

The current university access control system is based on the MIFARE Classic smart card, which conforms to ISO 14443 Type A, a standard for contactless integrated circuit cards to communicate with a "coupling device" (i.e. a smart card reader) over radio frequency. There are huge numbers of this particular card in existence — over 200 million are in use today.

The cryptography used in the card, a scheme named CRYPTO-1, was developed in-house by the manufacturer of the MIFARE Classic, NXP Semiconductors. NXP chose to keep the scheme secret, a practice known as security by obscurity. Such practice is eschewed by the security community because naturally, all cryptographic schemes are bound to have weaknesses and if researchers (or others) are not able to analyse a scheme, then they cannot provide advice as to how flaws within said scheme can be fixed. Furthermore, obscurity does not prevent others from deducing the scheme by observing it in operation and indeed this was the case for CRYPTO-1. In December 2007, a presentation at the Chaos Communication Congress (an annual security conference) by two German

researchers, Nohl and Plötz, described a partial reverse engineering of CRYPTO-1, as well as some weaknesses. They managed to do this by reconstructing the card's electronic circuit from photos of the chip. They then verified their reconstruction by eavesdropping on the reader-card communication. Just a few months later, in March 2008, researchers in the Digital Security group at Radboud University Nijmegen revealed a complete reverse engineering of the scheme and were able to clone and manipulate the contents of a MIFARE Classic card. The most serious attack they detailed in their paper can recover the card's cryptographic key in under a second using only a laptop, without any precomputation. NXP tried to obtain an injunction to prevent publication of the paper but were unsuccessful.

Ignoring the fact that the CRYPTO-1 scheme is inherently flawed, NXP's choice to use symmetric key cryptography for the MIFARE Classic was perhaps a misstep. Symmetric ciphers utilize what is called a shared secret, or secret key. Two parties wanting to communicate will first exchange this key over a secure channel and then use it to encrypt/decrypt messages sent between them. In the case of access control cards, this means that a card will store just one key, its own secret key, but that key will be stored in every door reader to which the card has access. This means that if a door reader is compromised, and the attacker is able to retrieve all the keys stored within, then they're able to clone any card which had access to that door. If this door is not in a very specific department, then many people will have access to it, and thus the attacker will have access to a very diverse set of doors — essentially, the entire system is compromised. Such a weakness does not exist when using a scheme based on asymmetric key cryptography, in which each card has not one but two keys — one public, one private. The private key is known only to the owner and is never sent over any channel, whilst the public key is known to everyone. If two parties wish to communicate, then they encrypt their messages with each other's public keys. The message can now only be decrypted with the recipient's private key, which only the recipient knows. In this case, the door reader contains only a long list of public keys corresponding to all the cards that can access the door. Thus, an attacker who's able to compromise a reader doesn't learn any secret information except for the door's private key. This only allows them to clone that specific door reader and doesn't compromise any other cards or readers in the system.

The aim of the project is to produce an access control system that uses asymmetric key cryptography to authenticate smart cards. The system should act as a replacement for the existing MIFARE Classic system.

Starting point

The project will make significant use of the material from Security I and Security II — I have already studied the lecture notes for Security II, although I have set aside some time in my plan for recap. Further, material from Object-Oriented Programming and Further Java will be utilised when writing the smart card application, which runs on the JavaCard platform and supports a subset of the Java language.

A previous attempt was made at this project by Denys Natykan, and so his dissertation

must be mentioned as a starting point. However, I already anticipate significant differences between our end products as I intend to use a rather different authentication protocol.

Substance and structure of the project

The objective of the project is to produce an access control system that implements an authentication protocol based on asymmetric key cryptography. As well as producing a card application and reader application that implement the authentication protocol, I intend to write a card-provisioning application that can be used to issue new cards or reprogram existing cards.

I intend to compare at least two different digital signature algorithms (DSAs) for speed in my evaluation, and it's possible that one or more of these algorithms won't be implemented by the JavaCard SDK, in which case I will have to implement them myself.

Given that smart cards are low power, I expect that I may have to spend time optimising the protocol, so that authentication happens within the required time.

Success citeria and evaluation

- An authentication protocol must be chosen.
- The protocol must be implemented in two separate applications — one to run on the card, the other on the reader.
- A command-line application must be written for provisioning and reprogramming cards.
- The system must be able to authenticate a card in less than one second.
- The system should be tested to ensure it operates as the protocol dictates it should.
- The dissertation must be planned and written.

Possible extensions

- Mutual authentication of both card and controller so that only authorised readers (i.e. university door controllers) are able to communicate with cards.
- Provide user untraceability as a feature of the authentication protocol.
- Ensure the system is resilient to cloning.
- Implement a GUI for the card-issuing application.

Timetable

Weeks 1 to 2

Initial research period. I will familiarise myself with existing authentication protocols and either select one of them to use, either in full or as a guideline, else I will design one myself. Familiarisation with the JavaCard SDK and the GlobalPlatform API.

Weeks 3 to 4

Implement a very basic challenge-response application on the smart card. Gain a deeper understanding of the J3A040 smart card, specifically the memory structure and the implications of this for fast authentication.

Weeks 5 to 10

Implement the chosen authentication protocol on the card and reader. Implement the card-provisioning application to run on computer.

Weeks 11 to 12

Time reserved for testing the system and sorting out any leftover bugs in the system.

Weeks 13 to 16

Reserved for dealing with bugs. If the base system is in good working order, then this time can be used to implement extensions. I'm currently undecided as to which extensions will be prioritised — my choice will depend on available time.

Weeks 17 to 20

Perform evaluation of the system. Begin writing dissertation.

Weeks 21 to 23

Time reserved for handling any bugs that escaped notice earlier in the process. This time can be used for writing the dissertation if there's nothing to be fixed.

Weeks 24 to 25

Finish writing initial draft of the dissertation.

Weeks 26 to 28

Time left to send dissertation draft to supervisor for review (this may happen a couple of times) and make changes. Finalise dissertation and submit electronically.

Resources declaration

- NXP J3A040 — a programmable smart card supporting JavaCard SDK and GlobalPlatform API.
- SCL3711 — a USB smart card reader.
- JavaCard SDK and GPShell for programming smart cards.
- My own MacBook Pro and Lenovo Yoga 2 Pro for writing applications, documentation and dissertation. I plan to use Git for source control, and will regularly push to a remote Bitbucket repository to avoid significant loss in the event that I experience a hard drive failure.