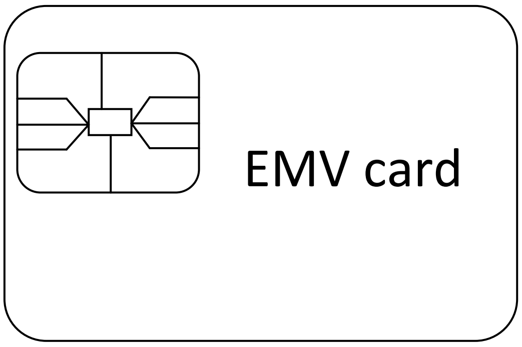
EMV (Chip and PIN) Project

Student: Khuong An Nguyen

Supervisor: Professor Chris Mitchell

Year: 2009-2010

Full Unit Project



Contents

Figures

Tables

1. Introduction

This chapter explains the motivation for this project. Initially, an introduction about electronic payment is looked at, as it is the basic concept the project is built on. Then, the scope of the project is reviewed and the chapter concludes by outlining eleven further chapters to be discussed later on, along with each chapter's general ideas.

1.1 Electronic payment

Electronic payment is now widely used across the globe as a new means of replacing old cash payment. Not only does electronic payment offer better security protection for transactions, but it also introduces a new concept of a payment system which allows merchants to personalise the financial environment. However, along with the introduction of an electronic payment system, criminals have adapted themselves to exploit the weaknesses of those systems. Information in magnetic stripe cards has been easily duplicated, cards forged without much effort. Thus, a new level of security for electronic payment was demanded which brought to attention the invention of the EMV standard. EMV has been implemented to enhance stronger protection for electronic payment transactions, introducing new combinations of hardware protection, as well as a sophisticated software mechanism to guarantee tamper-free resistance for credit and debit cards.

1.2 Scope of project

This project was designed to study the technologies inside EMV, and learn how EMV is implemented in real life. The project is concluded by the creation of the software reader which is capable of accessing and analysing information obtained from an actual EMV card. This project report is systematically organised into eleven chapters, which clearly reflects the development process of the project, from learning and understanding the smart card and EMV theories into implementing these as the software application, documentation reports and further testing procedures.

1.3 Contents of report

Chapter 1: **Introduction**
This chapter provides motivation for this project, outlines the seven chapters to be discussed further on, along with each chapter's general ideas. The introduction serves as a guideline to refer to when developing software.

Chapter 2: Definitions and abbreviations
This section explains all technical terms and abbreviations used in this project report in alphabetical order.

Chapter 3: Project overview
This chapter gives a complete understanding of the nature of the project. While it does not provide any technical discussion, the chapter is written to deliver readers a general

outlook at the whole project, introduces each project components and confirms the objectives to be achieved at the end. Further discussions behind the project components including architecture design and technical design are detailed in chapter 7 and chapter 8.

Chapter 4: Background

This chapter collates all relevant background material needed for this project, including a brief introduction about smart card, EMV standard, 'Chip and PIN' implementation, as well as a thorough discussion about the system entities and project objectives to be achieved. After a swift overview of the main concepts, a detailed description of internal structure of EMV is mentioned, concentrating on how EMV organises its file system structure as this is crucial for EMV software development. This project takes a closer look at how to initiate a payment transaction. Two protection mechanisms are taken into account including Card Authentication Methods to prove that the presented card is authentic and Cardholder Verification Methods to prove that the customer is genuine.

Chapter 5: **Command analysis**

This chapter outlines and explains all APDU commands used to communicate between software system and ICC. Initially, an introduction about APDU is discussed, outlining two different APDU commands. The chapter continues by looking at each C-APDU commands used in this project and the steps to construct them. A comprehensive analysis of R-APDU information concludes the chapter.

Chapter 6: EMV communication protocol construction

This chapter describes how to construct a communication protocol used to select the payment application. Initially, a theoretical approach is discussed. The solution is largely based on knowledge obtained from organisation of EMV file system discussed in chapter 4. The chapter is concluded by the display of algorithm showing construction steps.

Chapter 7: **High level design**

From a high-level user perspective, all software functionalities are described in detailed, including how they share information and interact with each other.

Chapter 8: **Detailed design**

This chapter takes a closer look at low level design, showing how each software functionality and software class is implemented in Java style pseudo codes.

Chapter 9: **Software testing**

When the software is finished and first prototype is produced, a detailed testing plan is scheduled. This plan includes stress testing to ensure the software is working correctly and is operating as intended. Three testing cases are Black-box testing, White-box testing and Integration Testing. For each test case, a variety of circumstances and data are input, all results are carefully logged and documented. In the case of testing failure, each case is examined, the codes are amended and tests are re-applied

Chapter 10: **Experimental results**

After software quality is verified, a detailed document report is produced based on results of applying software in real life EMV cards. This document is intended for reference purposes only, therefore some parts of sensitive personal information are hidden. There are four reports obtained from testing UK Chip and PIN NatWest debit card, HSBC debit card, Abbey (Santander) MasterCard and Barclays VISA card, as well as two international cards from Thailand and Vietnam. Each report is carefully documented and compared. This chapter aims to produce a real life perspective when applying the project and also underpins the results obtained to match theoretical expectation.

Chapter 11: **Conclusion and future work**

Finally, a summary report is produced to conclude the project. This report is intended by the project writer to summarise what he learned when doing this project. A possible future work plan is included to keep track of what should be implemented in the near future. Rather than repeating what the project does, this chapter serves as an individual summary of work in which the project writer expresses personal experiences when conducting this project, covering both technical and non-technical difficulties and solutions.

Appendix A: Software manual

This appendix aims to provide the software user with a complete guide detailing software operations, regardless of technical ability. All instructions are written simply to aid users when running the software with trouble-free.

Appendix B: Full source code listing

This appendix lists all source codes written in Java language. The codes are organised into different classes. Each class performs a particular function. There is a main class integrating all other classes to form a complete programme. For simplicity, all programming comments are removed to keep the codes tidy.

Appendix C: CRC cards

This appendix includes all original Class Responsibility Collaborator cards (CRC cards) produced handwritten from scratch when the high level of the software system was designed. The purpose of those CRC cards was to provide the software writer a useful approach to design an object-oriented system. They sketch all basic functionalities of each class and the relationship amongst those classes.

Bibliography: This section lists all references used when this project was carried out including internet sources, books and slides in their natural orders.

2. Definitions and abbreviations

2.1 Definitions

Byte	byte is the unit used to measured data information in computer. One byte contains exactly eight bits
Command	a message sent from terminal to ICC and from ICC back to terminal
Compromise	when a system has been compromised, it can no longer be trusted as the security system does not work properly as when functioning normally
Certification Authority	a trusted third party who binds a user's identity to a public key by issuing a digital certification
Digital signature	asymmetric cryptography application, which allows receivers to check the integrity of the message, ensures that information is authentic
Issuing bank	the bank which issues a credit/debit card to provide their services to the customers
Magnetic stripe	the black line at the back of a credit and debit card. Encrypted Information is stored within three tracks
Padding	adding some bits at the end of data to increase the length of data to expected length
PIN pad	device allowing customer to enter their PIN physically. It provides ten digit button from zero to nine, along with some necessary buttons such as ENTER, CLEAR, CANCEL
Plaintext	a normal unencrypted text in ASCII format
Payment transaction	the process when money is debited from a customer account who wants to buy goods, services ... and debited money is credited back to merchant account. Before the transaction happens, some security checks are applied to protect the seller and buyer
RSA	stands for Rivest-Shamir-Adleman, this is a cryptosystem involving a pair of public key and private key
Tag	an identification in hexadecimal form to represent a particular data object in smart card

Terminal	a combination of both reader device to exchange data with ICC and the software reader to process exchanged information
Track 2	the second flow of information on magnetic stripe card
Unattended terminal	a type of electronic merchant terminal which works through cardholder and does not need attendance by card issuer representative

2.2 Abbreviations

ADF	Application Definition File
AEF	Application Elementary File
AFL	Application File Locator
AID	Application Identifier
AIP	Application Interchange Profile
APDU	Application Protocol Data Unit
ATR	Answer to Reset, a sequence returned when ICC is reset
CAM	Card Authentication Methods
CDA	Combined DDA/Application Cryptogram Generation
CVM	Cardholder Verification Methods
C-APDU	Command Application Protocol Data Unit, this command is sent by Terminal to smart card
DDA	Dynamic Data Authentication
DF	Dedicated File
EF	Elementary File
FCI	File Control Information
ICC	Integrated Circuit Card, this is another technical name for smart card
ISO	International Organisation for Standardisation, this is an international recognised set of rules in which many applications conform
MF	Master File, this is a special DF located at top of EMV tree
PCI	Payment Card Industry, this is a standard involving all payment entities in card industry such as Credit card, Debit card, ATM, POS
PDOL	Processing Options Data Object List
POS	Point of Service, this implies ATM, Cashpoint
PIN	Personal Identification Number, this is a four-digit number used to identify cardholder during electronic payment transaction
PSE	Payment System Environment, a folder grouping all payment applications in EMV
R-APDU	Response Application Protocol Data Unit, this is the command returned by smart card in response to C-APDU
RFU	Reserved for Future Use, this term is used to refer to the bits which are currently unallocated in specification and intended to be modified in the future

SDA	Static Data Authentication, an offline authentication method
SFI	Short File Identifier
SW1	Status Byte 1, this is part of R-APDU which shows the status of C-APDU
SW2	Status Byte 2, this is part of R-APDU which shows the status of C-APDU
TLV	Tag Length Value

3. Project overview

This chapter is dedicated to give a complete understanding about the nature of this project with no specific technical discussion, to provide inspectors a better top-down overview. Initially, a brief introduction about the project nature from the project writer's point of view is discussed. Then, three components making up the project are introduced, along with their roles in the system. More details about those components are explained in the high level design chapter. As the project makes use of supporting hardware and software, they are introduced in the subsequent section. At the end of the chapter, a list of all project objectives is outlined to confirm which aspects of EMV are to be addressed in this project.

3.1 Project nature

This project is the combination of hardware and software aspects. Since the hardware involved in this project is pre-made, and can be easily bought in the market, all discussions are focused on how to make it function and combine it with the software system, rather than the internal structure of the hardware. Furthermore, because of the nature of a theoretical analysis project, there will be many discussions on how to analyse data obtained from a smart card, as well as details on how to construct communication commands in this report.

3.2 Project components

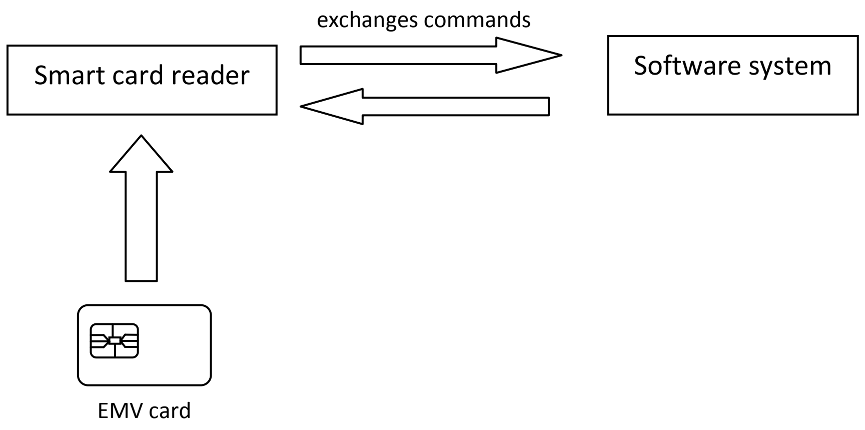


Figure 1: Project system components

There are three main parties involved in the project, as seen in figure 1. After being inserted into a 'smart card reader', the EMV card communicates indirectly with 'software system' using the facility provided by the reader device. The role of the smart card reader is restricted as a messenger who relentlessly sends and receives messages between EMV card and the software system. Software system, on the other hand, sends requests to EMV card, receives and analyses response messages. From the industry's point of view, 'smart card reader' and 'software system' form the merchant terminal side, while EMV card alone forms ICC side. Furthermore, aiding to the development of the project, there are supporting software and hardware, which do not participate in the complete system, yet provide support to verify system's reliability, and proving theoretical correctness.

3.2.1 Software system

This project uses PC/SC (Personal Computer/Smart Card) framework which provides the computers ability to communicate with smart card and smart card readers on a Windows platform. 'Windows Smartcard Resource Manager' comes with all Windows versions since Windows 2000. It allows any PC/SC compliant reader device to work immediately after plugging-in without the need for driver installation.
Software reader is written in Java language, utilising a set of Java API provided by JACCAL. JACCAL reads all PC/SC APIs and translates them to work under Java environment.
Figure 2 incorporates all elements of software system. This software system serves as 'software part' of terminal side.

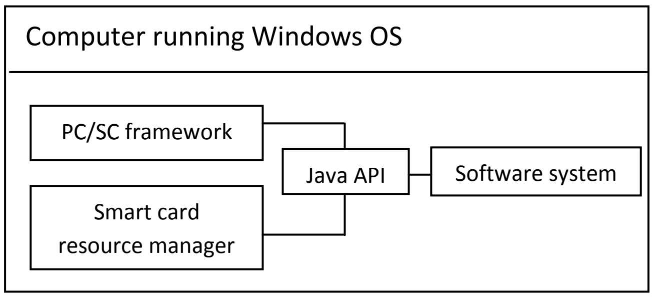


Figure 2: Software system overview

3.2.2 Smart card reader

The reader device is a piece of hardware equipped with contact surface to exchange signals with ICC. The device tested in this project is Gemplus manufactured by Gemalto company as seen in figure 3, which is also PC/SC workgroup member. However, any PC/SC compliant smart card reader will work.
This reader device serves as 'hardware part' of terminal side.

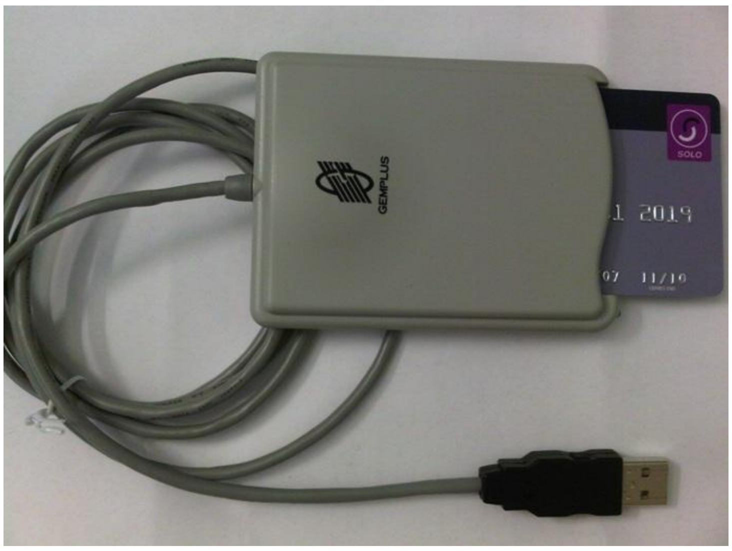


Figure 3: Smart card reader used in this project

3.2.3 EMV Credit/Debit card

EMV Card is the object which possesses information that the software reader needed to obtain and analyse. Figure 4 shows such an EMV card. In this project scope, EMV Card is rather an independent token, which cannot be modified by outside factors. However, it can react to suspicious interactions, protect itself in critical circumstances such as blocking access to sensitive information when being attacked. It also responds to challenges to verify and authenticate genuine users.

Figure 4: NatWest EMV MasterCard
This is only a sample card, the picture is officially licensed by National Westminster at www.natwest.com

3.2.4 Supporting software and hardware

This project uses MSR206 magnetic stripe reader to read information from magnetic stripe surface at the back of an EMV card to compare with duplicated track 2 information in the Chip. The picture of the device is captured in figure 5.

Figure 5: Magnetic stripe reader used in this project

The software used to read information from magnetic stripe card is MSR206DEMO provided along with the device.

3.3 Project objectives

There are six main purposes of this project to be achieved

- Understanding how merchant terminal communicates with ICC.

- A protocol of communication between EMV card and software system.
- Overview of CAMs and CVMs including descriptions, functionalities and their roles in electronic payment transaction.
- Analysis of information retrieved from EMV Chip and PIN card.
- Process of plaintext offline PIN verification.
- Development of software implementing theories listed above.

3.4 Project development plan

In terms of software development, the waterfall model is used to develop this project. The development process is divided into the six following stages:

1) Requirements analysis
   At this earliest stage, all project requirements are carefully analysed to make sure all details are fully understood correctly. A quick draft of software, hardware, and solution ideas is constructed to help the project writer imagine further about the requirements. All early difficulties about project requirements are thoroughly discussed with project supervisor before the actual project is continued.
2) Research
   Relevant information about EMV, card authentication, cardholder verification and PIN verification is collected from different sources of books, Internet and articles. The aim of this stage is to identify what has been done before and what already existed, which helps create a background chapter and decides whether the project will develop independent functionalities or enhance existing features. A log of this stage is reflected by the bibliography at the end of this report.
3) Modelling
   Based on all collated theories, a high level design of the system is built, concentrating on showing the users what main functionalities the system provides. This stage is reflected in the high level design chapter.
4) Coding
   Based on the outlined modelling of the system, all functionalities of the system are implemented in Java environment. This stage is carefully discussed in the detailed design chapter.
5) Testing
   This stage outlines three separate testing procedures including black box testing, white box testing, integration testing and system testing, used to verify the completeness of the first prototype of the system. This stage maintains a list of bugs, and showing how well the developer is removing them. A detailed testing plan is discussed in the software testing chapter.
6) Documenting
   This is the final stage to summarise the project. A report is produced to collate all aspects of the project.

4. Background

This chapter describes in detail the theories behind this project. It begins with an overview of EMV – the main concept studied in this project. The 'Chip and PIN' implementation of EMV in the United Kingdom, along with advantages and security issues are discussed. This chapter carries on with an insight introduction about EMV file system organisation as this is the only means for the outside world to communicate with ICC. The format which a general smart card uses to organise its file system will be looked at first, as EMV adapts a similar style. Having good background about EMV structure forms the next section about EMV transaction. The chapter is continued with the introduction of two important EMV protection mechanisms – Card Authentication Methods and Cardholder Verification Methods. An overview of both processes is explained, including an introduction for each method. The chapter is concluded by a good description and guideline to apply plaintext offline PIN verification in a real life EMV card.

4.1 EMV (Chip and PIN) introduction

This part provides an introduction about smart card, ISO 7816 and EMV (Chip and PIN) standards. Some disadvantages and security plots are also discussed. The purpose of this section is to outline the key features of the popular concepts related to this project, including smart card and EMV to be used further on as the project is developed.

4.1.1 Smart card introduction

Smart card is a plastic card which has the size of a credit card or the size of a mobile phone SIM card (cut-down smart card) used in the United Kingdom, Europe and many countries in the world. Smart card is also known as Chip card or integrated circuit card (ICC). Smart card has been widely accepted as a replacement for old magnetic stripe card.

The major security threat of un-secured static data used by magnetic stripe card is that they can be easily accessed and duplicated. This has been fully addressed through the introduction of a microprocessor inside the smart card. Sensitive personal information is securely protected by the Chip and the internal structure of smart card is not public, thus making it very hard to forge. With the ability to perform calculation, microchip also offers cryptographic functions (RSA, DES, Triple-DES) which significantly help the fight against fraudsters. Figure 6 describes the lifecycle of smart card from the manufacturing stage.

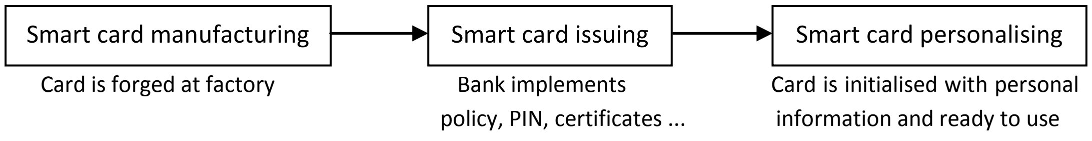

Smart card manufacturing
Card is forged at factory
Smart card issuing
Bank implements policy, PIN, certificates ...
Smart card personalising
Card is initialised with personal information and ready to use

Figure 6: Smart card life cycle

There are two types of smart card: Contact card and Contactless card

- Contact card must be inserted into a reader. Communication is performed via eight contact points on ICC (this is sketched in Figure 7). Reader device supplies needed power for operation.

- Contactless card does not require a reader. Communication is performed through an antenna in the card. Power is supplied from battery inside ICC.

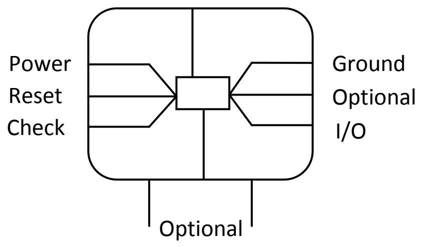


Figure 7: ICC contact surface

4.1.2 EMV (Chip and PIN) introduction

ISO 7816 standard defines properties and physical appearance of every smart card. ISO 7816-4 specifies smart card organisation and six basic commands used for communication. EMV (Europay, MasterCard & VISA) standard is based on ISO 7816. EMV is produced by three companies; Europay, MasterCard and VISA. It defines the logical and physical interface between the merchant terminal and payment smartcard. However, EMV standard is enhanced with security protections for the electronic payment system. EMV cards are Contact card. The first EMV Contactless card has been recently introduced.

In the United Kingdom, the implementation of EMV standard is known as 'Chip and PIN'. Figure 8 shows the 'Chip and PIN' logo, widely seen in the United Kingdom. The term 'Chip and PIN' is a combination of two discrete objects: 'Chip' and 'PIN'

- 'Chip' is the microprocessor inside smart card (this is discussed in chapter 4.1.1). Chip stores and protects information in internal memory. During a transaction, Chip proves that exchanged information is correct, and presented card is authentic
- 'PIN' (Personal Identification Number) is a four digit number that is known only by the cardholder. During an electronic transaction, PIN is physically entered by the customer using a PIN pad to prove that customer is indeed the genuine cardholder. The PIN pad used at Point of Service must qualify PCI standard (Payment Card Industry) which guarantees the confidentiality of a PIN during payment transaction. PIN is also securely stored and well-protected inside the Chip.

Chip and **PIN**

Figure 8: Chip and PIN logo
This logo is officially licensed by Chip and PIN UK at www.chipandpin.co.uk

Since ‘Chip and PIN’ was introduced in the United Kingdom, fraudulent transactions have significantly decreased which underlines one of the biggest advantages of switching from magnetic stripe card to ‘Chip and PIN’ card. However, there are other benefits to be noted:

- ‘Chip and PIN’ eliminates the need for a customer signature at Point of Service.
- ‘Chip and PIN’ provides a flexible electronic payment environment. Smart card issuers can write their own payment applications.
- ‘Chip and PIN’ decreases transaction processing time, allowing card authentication and card verification to be done online or offline.

The following diagram describes how a transaction is performed with Chip and PIN card, emphasising the role of PIN in the process:

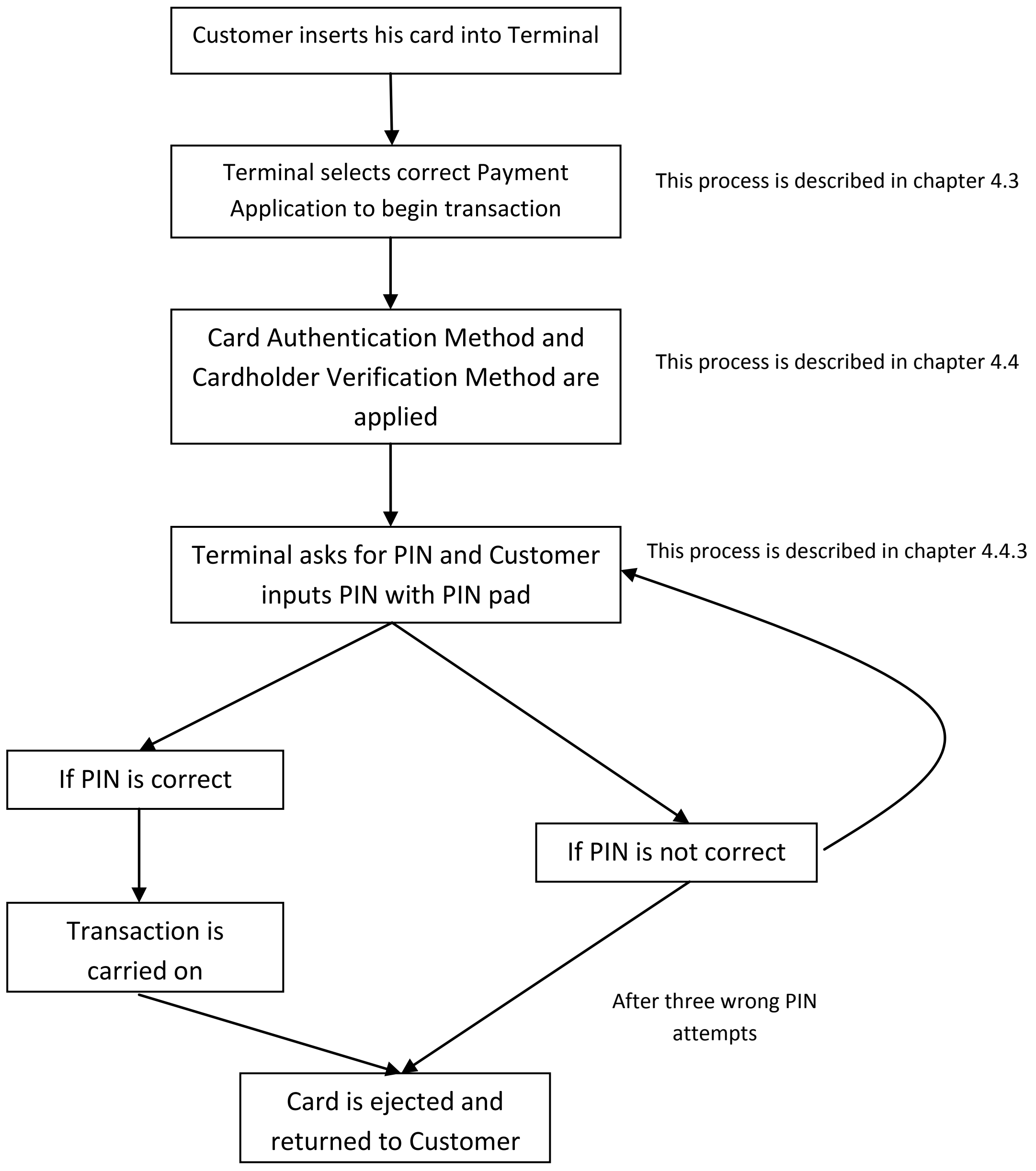

Customer inserts his card into Terminal
Terminal selects correct Payment Application to begin transaction
This process is described in chapter 4.3
Card Authentication Method and Cardholder Verification Method are applied
This process is described in chapter 4.4
Terminal asks for PIN and Customer inputs PIN with PIN pad
This process is described in chapter 4.4.3
If PIN is correct
If PIN is not correct
Transaction is carried on
After three wrong PIN attempts
Card is ejected and returned to Customer

Figure 9: Transaction with ‘Chip and PIN’ card

4.2 File system structure overview

This section provides an insight description into EMV, and concentrates on how EMV organises its file structure. Firstly, it would be easier to discuss the file system of general smart card, since EMV adapts a similar type of file system from ISO 7816-4. Each file system type along with access type is discussed. The purpose of this part is to detail sufficiently the internal structure of EMV background for protocol implementation in the next section.

4.2.1 General smart card file system overview

According to ISO 7816-4 standard, ICC organises its file system structure into two types: **DF** (Dedicated file) and **EF** (Elementary file). The whole system structure is viewed as a Tree. The root of the tree is a DF called **Master File** (MF).

DF can be seen as a folder. It stores one or many EFs, even other sub-DFs. Access Control can be implemented onto DF to prevent access to inside applications.

There are two ways for Terminal to access a DF -

- using FID (2 bytes Fixed File Identifier): Terminal must know the file system structure of ICC beforehand.
- using AID (16 bytes Application ID): Terminal does not need to know whole file system structure of ICC in advance.

This project uses AID option since terminal does not know the file structure inside ICC. FID is suitable for banks to implement their ATMs as they are also card issuers.

EF stores Card Application information. Each EF can contain other sub-EFs. However, EF cannot contain any DF.

There are two EF types -

- Internal EFs: this EF type is used exclusively by ICC and cannot be interfered by Terminal. They store sensitive information such as cryptographic function, PIN, ...
- Working EFs: this EF type can be accessed by Terminal.

There are two ways to access an EF -

- using FID: exactly same as the DF. Terminal must know whole ICC file system structure beforehand.
- using SFI (Short File Identifier): 5 bits number from 1 to 30. EF can be accessed indirectly using SFI without selecting DF which contains it initially.

4.2.2 EMV file system overview

EMV adapts ISO 7816-4 standard, thus it is compatible with ISO 7816-4 file system structure. There are three EMV file systems types -

- **ADF**: Application Definition File, this file type is adapted from DF.
- **DDF**: Directory Definition File, this file type is adapted from DF.

- **AEF**: Application Elementary File, this file type is adapted from EF.

Each ADF or DDF has a table of contents which stores information about that folder such as Entry Point. This table of contents is called AFL (Application File Locator).

ADF can contain many AEFs, but ADF cannot contain other DDFs. A branch of ADF tree can be seen in Figure 10. Each ADF has a unique AID as index and can be accessed with this AID. AEF can also be accessed with SFI. SFI is just an index number from 1 to 30. AEFs with similarities are grouped into one ADF. For example, all applications stored inside PSE are payment related system applications.

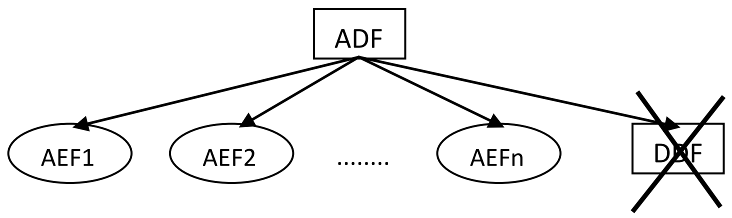


Figure 10: ADF tree structure

DDF can contain many ADFs. Each DDF has a unique AID as index and can be accessed with this AID. DDF can also contain other DDFs. A branch of DDF tree can be viewed in Figure 11. Each DDF maintains a list of all AIDs of other sub DDFs and ADFs inside this DDF.

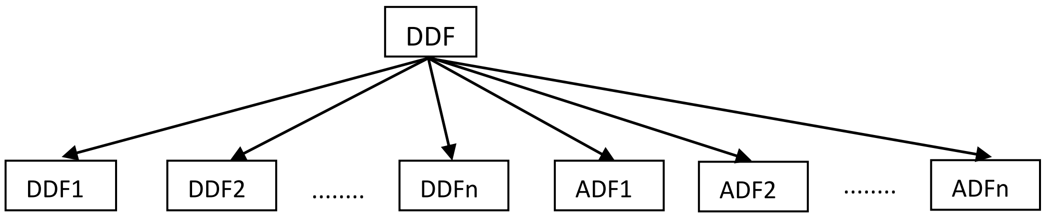


Figure 11: DDF tree structure

All EMV applications are stored in one DDF folder. This folder has AID ‘1PAY.SYS.DDF01’ and is called Payment System Environment (PSE).

4.3 EMV payment transaction

An EMV payment transaction needs to be initialised between merchant terminal and ICC, so that information can be exchanged. Terminal is the active entity to request a transaction creation. It sends a list of special instructions about the business environment and terminal restriction so ICC can review before agreeing to proceed with the transaction. Upon request, ICC will review those conditions and sends back a message for the transaction to go ahead. An agreement must be made between these two entities for transaction creation. There are two ways a transaction can be formed -

- Standard way: terminal does not specify any business instruction. In this case, ICC will set up the transaction in default.
- Personalised environment: terminal sends a list of business instructions which ICC adapts to set up a correct EMV transaction. The list can contain information about language environment and terminal restrictions.

Those special instructions are encapsulated in a data object called Processing Options Data Object List (PDOL). The structure of PDOL is as followed:

- PDOL begins with tag 9F 38
- Terminal type is 9F 35
- Terminal capabilities is 9F 33
- Terminal country code is 9F 1A
- Merchant category code is 9F 15
- The data field begins with tag 83

4.4 Payment transaction protection mechanism

To increase payment transaction security, EMV provides two protection mechanisms: Card Authentication Methods (CAMs) and Cardholder Verification Methods (CVMs).

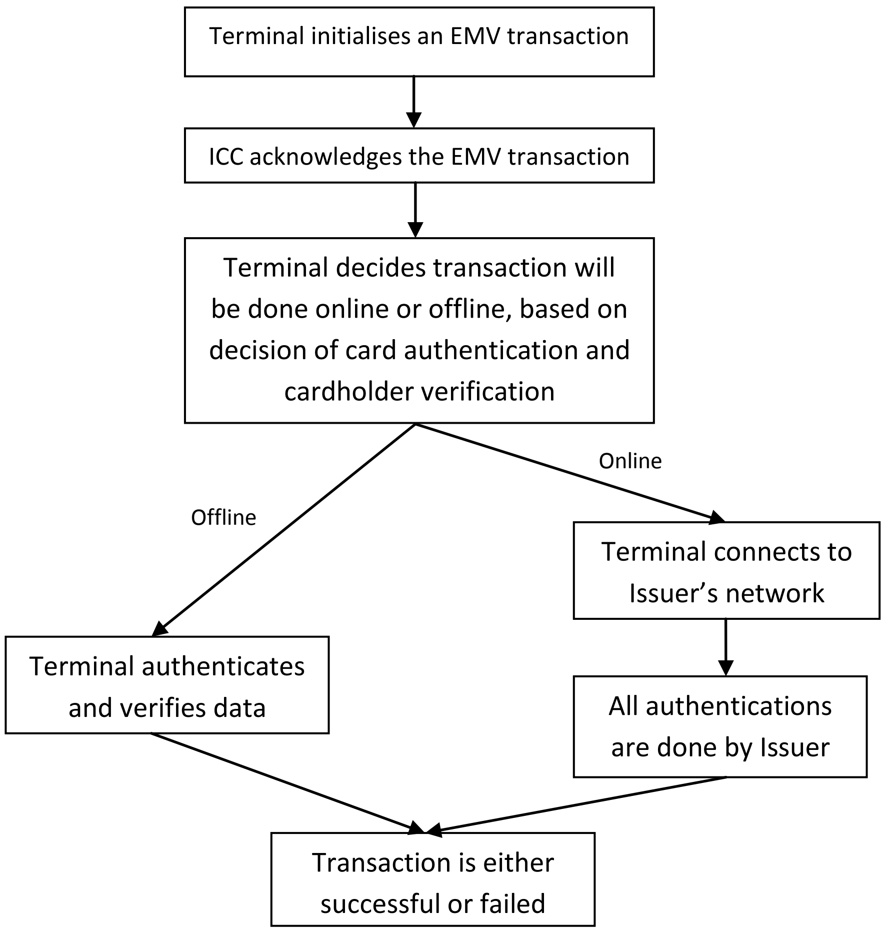

Terminal initialises an EMV transaction
ICC acknowledges the EMV transaction
Terminal decides transaction will be done online or offline, based on decision of card authentication and cardholder verification
Offline
Online
Terminal connects to Issuer's network
Terminal authenticates and verifies data
All authentications are done by Issuer
Transaction is either successful or failed

Figure 12: CAMs and CVMs during payment transaction

Figure 12 demonstrates the role of card authentication and cardholder verification which affects the decision of terminal during payment transaction. While CAMs makes sure that the presented card at merchant terminal is authentic, CVMs guarantees the customer in possession of the card is the genuine holder. This section shows how to identify which CAM and CVM the EMV supports, as well as their roles in the process.

4.4.1 Card Authentication Methods

There are two Card Authentication Methods (CAMs) types: offline CAM and online CAM. Online CAM requires Internet or phone connection, thus does not require much work from either terminal or ICC. ICC does not store needed information for authentication, encryption processes. All authentications are done in Issuer's online network. Offline CAM does not require online connection, but terminal has to do all authentication processes. In the scope of this project, we only consider offline CAM. There are two types of offline CAM, both based on digital signature -

- Offline static CAM (SDA): With this CAM, only terminal performs cryptographic operations.
- Offline dynamic CAM (DDA): With this CAM, both terminal and ICC perform cryptographic operations.

There are five elements in ICC which supports offline static CAM -

- Issuer Public Key Certificate (tag 90): this is issued by CA.
- Certification Authority Public Key Index (tag 8F): Index of Issuer Public Key Certificate.
- Issuer Public Key Exponent (tag 9F32).
- Issuer Public Key Remainder (tag 92).
- Signed Static Application Data (tag 93).

There are eight elements in ICC which supports offline dynamic CAM

- ICC Public Key Certificate (tag 9F46).
- ICC Public Key Exponent (tag 9F47).
- ICC Public Key Remainder (tag 9F48).
- Issuer Public Key Certificate (tag 90): this is issued by CA.
- Certification Authority Public Key Index (tag 8F): this is the Index of Issuer Public Key Certificate.
- Issuer Public Key Exponent (tag 9F32).
- Issuer Public Key Remainder (tag 92).

4.4.2 Cardholder Verification Methods

Cardholder verification process makes sure that the person doing payment transaction is indeed the genuine cardholder. In order for the process to carry on, ICC must support at least one Cardholder Verification Method (CVM). ICC's CVM can be found by analysing AIP (Application Interchange Profile).

There are eight types of CVMs

- *No CVM required*: if card suggests this method, it simply does not matter if the person presenting the card is the real owner. The obvious advantage of this method is the fluency of transaction.
- *Online Enciphered PIN verification*: a PIN is needed from user. Terminal encrypts this PIN with symmetric encryption before sending it to issuer's network for verification.
- *Enciphered offline PIN verification*: a PIN is required from user. Terminal encrypts this PIN with RSA and sends to ICC. Chip must be able to perform RSA operations. It decrypts PIN and compares with the reference one in the internal memory.
- *Plaintext offline PIN verification*: a PIN is needed from user. However, it is sent straight to ICC without any encryption. ICC compares reference PIN stored in internal memory with user PIN.
- *Signature verification*: this method prompts cardholder to provide a paper signature, which must match the one on the back of the card. This method requires a human to witness the verification.
- *Plaintext offline PIN verification and signature verification*: this method is simply a combination of plaintext offline PIN verification and signature verification.
- *Enciphered offline PIN verification and signature verification*: this method is a combination of enciphered offline PIN verification and signature verification.
- *Fail CVM processing*: this method forces terminal to terminate cardholder verification process.

CVM list (tag '8E') can be found with the following format (according to EMV Book 3 – Table 40). There are three objects in CVM list

- X (4 bytes in binary format): this is a number set by Issuer to determine currency.
- Y (4 bytes in binary format): this is a number set by Issuer to determine currency.
- Two-byte cardholder verification rules.
    - The structure of first byte is described in table 1.

Table 1

CVM list

	b8	b7	b6	b5	b4	b3	b2	b1
Fail CVM processing	X	-	0	0	0	0	0	0
Plaintext PIN verification	X	-	0	0	0	0	0	1
Enciphered online PIN verification	X	-	0	0	0	0	1	0
Plaintext PIN verification and Signature verification	X	-	0	0	0	0	1	1
Enciphered offline PIN verification	X	-	0	0	0	1	0	0
Encipher PIN verification and Signature verification	X	-	0	0	0	1	0	1
Signature verification	X	-	0	1	1	1	1	0
No CVM needed	X	-	0	1	1	1	1	1

Bit b8 is reserved for future use (RFU).

If bit b7 is 0, Terminal terminates verification process if current CVM fails.

If bit b7 is 1, Terminal tries next CVM (if existed) when current CVM fails.

- Second byte is seen as 2-digit hexadecimal number. It describes the condition in which CVM is applied -
  - 00: CVM can be applied without restriction.
  - 01: CVM can be applied for all transactions with cash or cash-back.
  - 02: CVM can only be applied for transactions without cash or cash-back.
  - 03: CVM can always be applied if terminal supports this CVM type.
  - 04: CVM can be applied if customer purchases with cash.
  - 05: CVM can be applied if customer purchases with cash-back.

If CVM list does not exist in ICC, terminal terminates cardholder verification process. If CVM exists, terminal will perform each CVM rule according to the order it appears in the CVM list. If one CVM rule fails, terminal continues with the next CVM rule until at least one is successful or the list is finished.

4.4.3 Plaintext offline PIN verification

This section discusses plaintext offline PIN verification method provided by the cardholder verification protection mechanism. Initially, some advantages and disadvantages of the method are looked at. The section moves on with a description of the verification procedure. At the end of this section, a guideline is specified showing how software system implements a practical application to perform real plaintext offline PIN verification on real life EMV smart card.

- Advantages and disadvantages

Plaintext offline PIN verification is one of the eight cardholder verification methods. It is considered the most cost-effective verification method to prove the authenticity of the cardholder based on Chip and PIN. The idea of this method is the use of 4-digit PIN which is only known by the cardholder and the Chip to perform verification. This 4-digit PIN however is transferred openly directly from customer input to ICC without encryption. Terminal does not perform any cryptographic operation and does not require to equip cryptographic facilities. Thus the obvious advantage is the fluency of network and the terminal cost saving. However, the major drawback as implied is the lack of security. Therefore, this type of verification should only be used at unattended terminals (such as ATM machines).

- Verification procedure

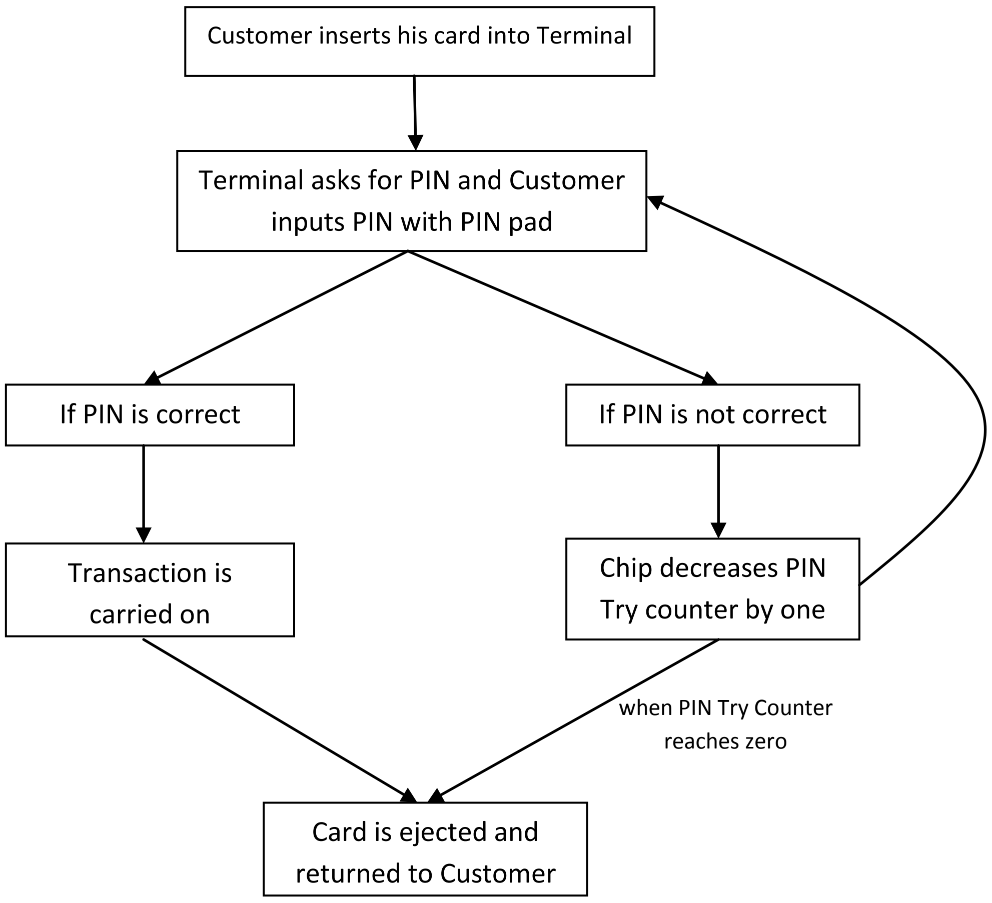

Customer inserts his card into Terminal
Terminal asks for PIN and Customer inputs PIN with PIN pad
If PIN is correct
If PIN is not correct
Transaction is carried on
Chip decreases PIN Try counter by one
when PIN Try Counter reaches zero
Card is ejected and returned to Customer

Figure 13: PIN verification during transaction

Figure 13 illustrates plaintext offline PIN verification procedure. During a payment transaction, cardholder enters this 4-digit via a PIN pad provided at Point of Service. The entered PIN is not encoded in any way and is sent directly to ICC. ICC compares this PIN with the reference PIN stored in its internal memory when the card is personalised. If the user's PIN is correct, transaction occurs normally. Otherwise, terminal prompts for PIN re-entry and decreases PIN try counter number by 1. If this number reaches zero, card is blocked and no more PIN attempts is allowed.

Step-by-step process for terminal to process plaintext offline PIN verification:

- If bit 5 in the first byte of AIP is not zero, terminal knows this card does support cardholder verification. Otherwise, terminal will terminate the process and signals 'ICC does not support CVM'.
- Terminal checks if CVM list exists. This is done by checking the existence of tag '8E'. If it does not exist, terminal terminates process and signals 'ICC does not support CVM'.
- Terminal analyses the second byte of cardholder verification rule, it makes sure all conditions are met.

- Terminal sees if the 6 right-most bits of the first byte of cardholder verification rule is **000001.** If it is not, it means ICC does not support plaintext Offline PIN verification, terminal stops.
- In the next step, terminal identifies how many times cardholder is allowed to input PIN. This number is stored inside PIN Try Counter object data. In order to retrieve this data object, terminal sends GET DATA command to ICC. This command is described in chapter 5.2.3. ICC will respond with R-APDU containing PIN Try Counter number, it begins with tag '9F 17' followed by its length.
- If R-APDU's Status Word is '90 00' and PIN Try Counter is greater than 0, terminal prompts customer to enter 4-digit PIN with PIN pad.
- Before sending this PIN to ICC, terminal needs to encode it into proper PIN block format as described in table 2

  Table 2

  PIN block format

C	N	P	P	P	P	P/F	P/F	P/F	P/F	P/F	P/F	P/F	P/F	F	F

  C (Control Field): 4 bit value 0010 (2 in hexadecimal)

  N (PIN length): 4 bit value from 0100 to 1100 (4 to C in hexadecimal)

  P (PIN digit): 4 bit value from 0000 to 1001 (0 to 9 in hexadecimal)

  P/F (PIN/Filler): this is determined by PIN length

  F (Filler): 4 bit value 1111 (F in hexadecimal)

  PIN can have any length from 4 digits to 12 digits. Each digit varies from ZERO to NINE
- Terminals then applies VERIFY command with constructed PIN to ICC. This command is described in chapter 5.2.4. ICC will compare this PIN with the actual PIN it stores.
- ICC responses with R-APDU.
  - If Terminal inputs a wrong PIN, Status Word will be '63 Cx' with x is the number of PIN Try Counter left. If x returns to ZERO, ICC blocks the Card and no more CVM can be applied. If Terminal still inputs PIN after this point, ICC returns Status Word '69 83'.

If entered PIN matches stored PIN on ICC. R-APDU has Status Word '90 00'.

5. Command analysis

This chapter is dedicated to explain in detail all communication commands involved in the project, because this is a very important means used by the software system to communicate with smart card. Almost eighty percent of the project is carried out by understanding commands, creating commands and analysing commands. This is referred to intensively when writing the software system and producing the detailed design document.

5.1 Application Protocol Data Unit

Application Protocol Data Unit (APDU) is defined by ISO 7816 standard. Information is encapsulated in this data unit before being transferred to the other entity. There are two APDU types: Command APDU (C-APDU) and Response APDU (R-APDU).

5.1.1 Command APDU

Command APDU (C-APDU) is the message sent by terminal to ICC to request a particular action. There are two distinct parts of a C-APDU command: Header (4 bytes) and Body (any arbitrary length) as seen in table 3.

Table 3

C-APDU command structure

Header				**Body**		
CLA	INS	P1	P2	L_c field	Data field	L_e field

- CLA: class where the command lies.
- INS: the command itself.
- P1: command first parameter.
- P2: command second parameter, this is usually instruction.
- L_c: (1 bytes or 3 bytes): number of bytes of Data field.
- Data field: a string to be sent to ICC.
- L_e: (0, 1, 2 or 3 bytes) identifies the maximum length (in bytes) in Data field to be received later in R-APDU.

5.1.2 Response APDU

Response APDU (R-APDU) is the reply message sent by ICC back to Terminal in response to a C-APDU command. Table 4 describes the structure of R-APDU. There are two distinct parts: Body with length $L_r \leq L_e$ (specified in C-APDU) and Trailer (2 bytes).

Table 4

R-APDU command structure

Body	**Trailer**	
Data field	SW1	SW2

- Data field: a string returned by ICC.
- SW1 and SW2 (Status Word): show the result of applying C-APDU command, whether the command is successful or failed, and the reason of failure. Some

examples of status word: 90 00 means command is successful, 6A 82 means unknown parameters.

5.2 EMV commands

There are sixteen commands defined by ISO 7816-4 standard to interact with EMV card:

- **READ BINARY**, **WRITE BINARY**, **UPDATE BINARY** and **ERASE BINARY** commands are used to interact with file.
- **READ RECORD**, **WRITE RECORD**, **APPEND RECORD** and **UPDATE RECORD** commands are used to interact with records inside application.
- **GET DATA** and **PUT DATA** commands are used to interact with primitive data objects.
- **SELECT** command is used to choose an application.
- **VERIFIY** command is used to initiate the comparison process of the data sent by terminal and reference data stored in ICC.
- **INTERNAL AUTHENTICATE** and **EXTERNAL AUTHENTICATE** commands are used to initiate the authentication data computation process.
- **GET CHALLENGE** command is used for security related procedure.
- **MANAGE CHANNEL** command is used to open or close channels.

However, each card operating system can define its own commands with APDU. For example, a bank can include a BLOCK command to prevent access to a whole card. In the scope of this project, four ISO 7816-4 standard commands are used, plus another EMV specialised command. The four ISO 7814-4 standard commands are actually overridden under EMV specification to adapt EMV environment, however they have the same inherited features. They are SELECT, READ RECORD, VERIFY and GET DATA commands. The specialised EMV command is GET PROCESSING OPTIONS used to initiate an EMV transaction.

5.2.1 SELECT command

SELECT command is used to perform application selection. This command is wrapped in C-APDU protocol to send from terminal to ICC. ICC will respond with R-APDU containing FCI with all information about selected application.

For demonstration purpose, terminal will select Payment System Environment (PSE) with AID ‘1PAY.SYS.DDF01’. Here are the steps to build SELECT command:

Header analysis:

- CLA is **00**. This is set by default by ISO 7816-4.
- INS is **A4**. This is set by default by ISO 7816-4.
- P1: the first parameter of SELECT command is seen as 8-bit (b8, b7, b6, b5, b4, b3, b2, b1) specified the method terminal wants SELECT command to perform, such as select by name, select by full directory, etc

  Terminal chooses select by name because the AID of PSE is known. According to ISO 7816-4, this 8-bit group is **0 0 0 0 0 1 0 0**

However, because P1 only accepts one single value of 2-digit hexadecimal, terminal converts 00000100 into **04h**

Thus P1 = 04

- P2: the second parameter of SELECT command is also seen as 8-bit (b8, b7, b6, b5, b4, b3, b2, b1) as described in table 5 specifying the record terminal wants to read. According to ISO 7816-4, terminal should only be interested in bit b2 and b1.

Table 5

SELECT command parameter

	B8 b7 b6 b5 b4 b3	b2 b1
Read First Record	X X X X X X	1 0
Read Last Record	X X X X X X	1 1
Read Next Record	X X X X X X	1 0
Read Previous Record	X X X X X X	1 1

Bit b3 to bit b8 are undefined and considered reserved for future use (RFU) [ISO 7816-4].

Terminal wants to select first record. **Thus P2 = 00**

At the moment, C-APDU Header is:

00 A4 04 00

Body analysis:

- Data field: this is where terminal puts the AID ‘1PAY.SYS.DDF01’, terminal needs to convert this string into hexadecimal: **31 50 41 59 2E 53 59 53 2E 44 44 46 30 31**
- L_c: this is the number of bytes of Data field. Because ‘1PAY.SYS.DDF01’ has 14 bytes, thus **L_c = 0Eh**
- L_e: this is the number of expected bytes from R-APDU command. If left blank, R-APDU can return any length of bytes

C-APDU Body is:

0E 31 50 41 59 2E 53 59 53 2E 44 44 46 30 31

Merge Header field and Body field together, terminal forms completed C-APDU SELECT command to select PSE application from ICC:

00 A4 04 00 0E 31 50 41 59 2E 53 59 53 2E 44 44 46 30 31

5.2.2 READ RECORD command

READ RECORD command is used to read contents inside an application. This command requires SELECT command to be performed beforehand. Terminal uses precisely the same general format of C-APDU command, this time with different parameter values

The steps to construct C-APDU for READ RECORD command -

- CLA: **00**, this parameter is defined by default by ISO 7816-4
- INS: **B2**, this parameter is defined by default by ISO 7816-4
- P1: this is Record number
- P2: is seen as 8-bit (b8, b7, b6, b5, b4, b3, b2, b1) as described in table 6.
- L_c: empty

- Data field: empty
- L_e: 00 (this allows R-APDU to return any length)

Table 6

READ RECORD command parameter

	b8 b7 b6 b5 b4	b3 b2 b1
Read Record P1	- - - - -	1 0 0
Read all Record from P1 to End	- - - - -	1 0 1
Read all Record from End to P1	- - - - -	1 1 0

Bit b8 to bit b2 are the 5 right most bits of SFI (Short File Identifier). If SFI has more than 5 bits, only 5 rightmost bits are kept, the rest are discarded. Now, SFI is merged with the final 3 bits to form the complete parameter.

To demonstrate the idea, assuming SFI is 08h. Terminal converts it into 8 bits: 0000 1000. As only 5 leftmost bits are kept, terminal discards the 3 rightmost bits, hence SFI reads: 00001. Now, assuming terminal wants to read first record P1 only, the three bits representation are 100. By merging all together, terminal has: 00001100. Finally, it converts this string back to hexadecimal **0Ch**.

Full C-APDU command in this example will read:

00 B2 01 0C

5.2.3 GET DATA command

GET DATA command is used by ICC to obtain a data object being controlled by ICC. These data objects are directly managed and frequently updated by ICC, thus are different from static records and files hard-coded into the Chip during personalised stage of smart card manufacturing. An example of such primitive data object is the ‘PIN try counter’, this object holds the value of remaining PIN tries customer can perform on the card.
Here are the steps to build C-APDU for GET DATA command -
- CLA: **80**. This is default by EMV specification.
- INS: **CA**. This is default by EMV specification.
- P1 P2: **9F 17**. This is PIN Try Counter parameter, according to EMV specification
- L_c: empty
- Data: empty
- L_e: 00. This allows R-APDU command to return any arbitrary length.

Thus, C-APDU command should read:

80 CA 9F 17 00

5.2.4 VERIFY command

VERIFIY command is used to initiate the comparison process of the data sent by terminal and reference data stored in ICC.

The steps to build C-APDU for VERIFY command -
- CLA: 00, according to ISO 7816-4 standard.
- INS: 20, according to ISO 7816-4 standard.

- P1: 00, according to ISO 7816-4 standard.
- P2: this depends on type of data to be sent.
- L_c: length of P2 parameter.
- Data: the data sent to ICC for comparison.
- L_e: empty, so R-APDU can have arbitrary length.

5.2.5 GET PROCESSING OPTIONS command

This command is used by terminal to initialise an EMV transaction with ICC. Here are the steps to build C-APDU for GET PROCESSING OPTIONS command -

- CLA: 80, according to EMV specification – Book 3.
- INS: A8, according to EMV specification – Book 3.
- P1: 00, this is RFU as specified by EMV specification – Book 3.
- P2: 00, this is RFU as specified by EMV specification – Book 3.
- L_c field: 02, this is the length of Data field in this case.
- Data field: 83 00, this is where PDOL is specified, followed by tab 83. Terminal in this project will ignore PDOL as it does not have information about the business environment set up at Point of Service. (Referring to chapter 4.3 on PDOL discussion).
- L_e field: empty, so it accepts any length returned by R-APDU.

So C-APDU for GET PROCESSING OPTIONS should read:

80 A8 00 00 02 83 00

5.3 Response command analysis

This section describes how to analyse information returned by ICC under different circumstances. Each case is explained with an example obtained from real EMV card.

5.3.1 Payment application R-APDU analysis

This section shows how to analyse payment application information received from ICC when terminal looks into PSE folder. The following sequence shows an example of R-APDU command returned by ICC in a NatWest MasterCard:

$$\text{70 1A 61 18 4F 07 } \underbrace{\text{A0 00 00 00 04 10 10}}_{\text{Application ID}} \text{ 50 0A } \underbrace{\text{4D 41 53 54 45 52 43 41 52 44}}_{\text{Application label}} \text{ 87 01 01}$$

Terminal looks for tag '4F' as it indicates AID (Application ID). Terminal sees that this tag only appears once in the sequence, thus the EMV card only contains one payment application. Next, terminal focuses on getting information about this application. It sees the next byte is after '4F' is 07h, which means the next 7 bytes 'A0 00 00 00 04 10 10' indicates AID. The following tag '50' indicates the application label, which has 0Ah bytes. Because 0A in hexadecimal is equal to 10 bytes in decimal thus the next 10 bytes will indicate the application label: 4D 41 53 54 45 52 43 41 52 44. By translating this hexadecimal sequence to ASCII format, terminal gets human-readable string MASTERCARD, which surprisingly is also the type of EMV card. The next tag '87' indicates application priority, which follows by 01h means the next only byte contains priority level. Terminal sees that this application has highest priority level 1.

5.3.2 GET PROCESSING OPTIONS R-APDU analysis

This section shows how to analyse information received from GET PROCESSING OPTIONS command. In response to this command, ICC will set up an EMV transaction and returns R-APDU command containing two important fields called AIP (Application Interchange Profile) and AFL (Application File Locator). These two fields contain information about Card Authentication Methods and Cardholder Verification Methods. Body of R-APDU will have the following structure. || symbol means concatenation.

Tag ‘80’ || Length of AIP + AFL || AIP || AFL

AIP contains two bytes, in which the first byte contains CAMs and CVMs status, while the second byte is reserved for future (RFU). If terminal divides first byte into 8 bit, it can extract useful information about CAMs and CVMs.

- Bit 7: if value of bit 7 is 1, it means ICC supports offline data authentication SDA. If value is 0, it means ICC does not support offline authentication SDA.
- Bit 6: if value of bit 6 is 1, it means ICC supports offline data authentication DDA. If value is 0, it means ICC does not support offline authentication DDA.
- Bit 5: if value of bit 5 is 1, it means ICC supports cardholder verification, if value is 0, it means ICC does not support cardholder verification.
- Bit 2: if value of bit 2 is 1, it means ICC supports combined offline data authentication DDA and application cryptogram generation.

AFL contains one or many 4 byte group AEF which contains public information needed for terminal to complete the transaction.

- Byte 1 identifies SFI.
- Byte 2 identifies first record number.
- Byte 3 identifies last record number.
- Byte 4 identifies number of records which have offline data authentication.

An example of R-APDU sequence obtained from a NatWest MasterCard:

$$\text{80 0A }\underbrace{\text{5C 00}}_{\text{AIP}}\ \underbrace{\text{08 01 01 00}}_{\text{AFL1}}\ \underbrace{\text{10 01 04 01}}_{\text{AFL2}}$$

Terminal sees that two bytes AIP is ‘5C 00’. By dividing first byte 5C into 8 bits: 0101 1100, terminal can analyse status of CAMs and CVMs:

- Bit 7 is 1, which means ICC supports offline data authentication SDA.
- Bit 6 is 0, which means ICC does not support offline data authentication DDA.
- Bit 5 is 1, which means ICC supports cardholder verification.
- Bit 2 is 0, which means ICC does not support combined offline data authentication DDA and application cryptogram generation.

Next, terminal sees that there are two groups of AFLs. Terminal will analyse the first AFL: 08 01 01 00. The five most significant bits of first byte 08 indicates SFI values, in this case it is 00001. The second byte 01 shows that the first record number is 01. The third byte shows that the last record number is 01, in which terminal can conclude that there is only one record for this SFI. Finally, the last byte is 00, which means this record does have offline data authentication.

Next, terminal analyses the second AFL: 10 01 04 01. The five most significant bits of first byte 08 indicates SFI values, in this case it is 00010. The second byte 01 shows that the first record number is 01. The third byte shows that the last record number is 04, in which terminal can conclude that there is four records for this SFI. Finally, the last byte is 01, which means only first record has offline data authentication.

5.3.3 Cardholder verification R-APDU analysis

This section shows how to analyse cardholder verification methods information obtained from CVM list. The following R-APDU sequence shows an example of CVR (Cardholder verification rule) obtained from CVM list:

$$\underbrace{42\ 01}_{\text{CVR1}}\ \underbrace{41\ 03}_{\text{CVR2}}\ \underbrace{1E\ 03}_{\text{CVR3}}\ \underbrace{02\ 03}_{\text{CVR4}}\ \underbrace{1F\ 03}_{\text{CVR5}}$$

Terminal knows the each CVR is a group of two bytes, thus it will process each CVR accordingly. For demonstration purposes, terminal will analyse the first three CVRs only.

1) For CVR1 ‘42 01’, the first byte 42 indicates the cardholder verification method and the second byte 01 indicates the rule in which the method is applied. Terminal converts first byte from hexadecimal into 8 bits, therefore it gets 01000010, then it discards the two leftmost bits of the sequence to obtain 000010. This is the representative sequence for enciphered online PIN verification method, according to Table1 - CVM list. The condition in which this method is applied is 01, which indicates it is performed if unattended cash at point of service.
2) For CVR2 ‘41 03’, the first byte 41 indicates the cardholder verification method and the second byte 01 indicates the rule in which the method is applied. Terminal converts first byte from hexadecimal into 8 bits, therefore it gets 01000001, then it discards the two leftmost bits of the sequence to obtain 000001. This is the representative sequence for plaintext offline PIN verification method, according to Table1 - CVM list. The condition in which this method is applied is 03, which indicates it is always performed if the terminal supports this method.
3) For CVR2 ‘1E 03’, the first byte 1E indicates the cardholder verification method and the second byte 01 indicates the rule in which the method is applied. Terminal converts first byte from hexadecimal into 8 bits, therefore it gets 00011110, then it discards the two leftmost bits of the sequence to obtain 011110. This is the representative sequence for signature verification method, according to Table1 -

CVM list. The condition in which this method is applied is 03, which indicates it is always performed if the terminal supports this method.

5.3.4 Personalised information R-APDU analysis

This section shows how to analyse personalised information received from one record on ICC. The following sequence shows an example of R-APDU:

$$\text{70 36 5F 20 0E } \underbrace{\text{4E 47 55 59 45 4E 2F 4B 48 55 4F 4E 47 20}}_{\text{name on card}}$$

$$\underbrace{\text{57 13 54 54 ** ** ** ** 36 94 D1 00 22 01 15 79 20 09 01 89 1F}}_{\text{track 2 data}} \text{ 9F 1F 0D 31 35 37 39 32}$$

30 30 39 30 31 38 39 31

The terminal looks for tag ‘5F 20’ which indicates name on card. This tag is followed by OEh which indicates the next 14 bytes belongs to name on card content: 4E 47 55 59 45 4E 2F 4B 48 55 4F 4E 47 20. By translating this sequence into ASCII format, terminal gets meaningful name on card ‘NGUYEN/KHUONG’.

Tag ‘57’ found in this sequence is a duplicated of track 2 information in magnetic stripe. A quick check with magnetic stripe reader at the back of EMV card shows the same result for track 2. Terminal sees that the sixteen digits followed are the actual sixteen digits on card. For security purpose, the eight digits in the middle are hidden: 54 54 ** ** ** ** 36 94. This is followed by a deliminator D to separate between sixteen digits and expiry date: 10/02 (February-2010).

6. EMV communication protocol construction

This chapter describes how to construct a communication protocol to be used to select payment application. Initially, an approach is thought of, based on current organisation of EMV file system discussed in chapter 3. Then, some essential theories are reviewed to prepare for construction process. The chapter is followed by three construction steps. This process makes use of SELECT and READ RECORD commands described in chapter 5.

6.1 Approach discussion

Before started, terminal must always know beforehand at least one existing application on ICC. However, as there are only a limited number of applications currently allowed to run on EMV Card, a sensible approach is performing an exhaustive search to verify the existence of each application. Notice that at the time after ICC resettles, it is not possible to tell what applications ICC supports. Terminal chooses PSE as the application to select as this is a popular DDF on mostly all EMV Card. This DDF contains all EMV payment related applications.

6.2 Protocol algorithm

Before introducing the algorithm, some EMV theories are reviewed. When a Smartcard is inserted into a Terminal, it is in hibernation state and cannot be used yet. Thus, Terminal will send an electric shock signal to wake ICC up. This is called 'Reset process'. In response to this event, ICC will reply with an ATR string. Right after terminal initialises communication with ICC, terminal position is at EMV Root Master File (MF). In other words, it is at the top of the file system tree. However, it does not know how many ADFs master file has, as well as their locations (at what byte).

The algorithm pseudo code will be provided with further explanation in the detailed design chapter. Four steps of the algorithm are processed in their natural orders unless being re-directed by instruction -

- **Step 1**: PSE confirmation and preparation
  Terminal applies SELECT command (building instruction described in chapter 5.2.1) to select PSE folder on ICC. We know PSE's AID is '1PAY.SYS.DDF01', therefore C-APDU should read:

      00 A4 04 00 <14 bytes hexadecimal of 1PAY.SYS.DDF01>
  = **00 A4 04 00 0E 31 50 41 59 2E 53 59 53 2E 44 44 46 30 31**

  ICC should reply with Status Word '90 00' (successful command) confirming the existence of PSE on ICC and the SFI of this PSE folder. Any Status Word different from '90 00' either means PSE does not exist or it is inaccessible. Process is terminated in other cases.

- **Step 2**: READ record inside
  After PSE folder is selected, terminal can read the content inside using READ RECORD command (building instruction described in chapter 5.2.2). However, because READ

RECORD command demands precisely the number of bytes to read, terminal has to find out the size of first record. Terminal puts 0 bytes as size parameter in READ RECORD command and ICC will reply the actual size of the first record. The C-APDU command should read:

00 B2 01 0C 00

Because the record size is different for each ICC, let us assume it is 03 bytes for explanation purposes. The R-APDU command received from ICC should read:

6C 03

Here, R-APDU's first Status Word points out the requested size is wrong (6C), and second Status Word 03 is the correct size it should be. Now, terminal can use READ RECORD command to get first record. The C-APDU command should read (notice the change in final two bytes since first READ RECORD command)

00 B2 01 0C 03

At this stage, terminal must get a successful R-APDU '90 00' along with record details. Terminal will apply READ RECORD command again, this time asking ICC to provide the next record. At this stage, there are two possibilities:

- If the next record is an ADL, advances to step 3.
- If the next record is a DDF, advances to step 4.

Otherwise, terminal keeps repeating step 2 until R-APDU returns '**6A 83'**, which means 'Record not found'. In this case, terminal is confident all records in the same folder have been read. Process is exited normally.

- **Step 3**: ADF process
  This step will process ADF file only. Terminal simply performs a SELECT command again with this new ADF. In order words, terminal performs step 1 again with the new AID of current ADF. After this step finishes, terminal returns to previous higher level of the tree, at the point it left off to traverse deeper, and continue from then.

- **Step 4**: DDF process
  This step will process DDF file only. We simple perform a SELECT command again with this new DDF. Step 1 is called with new SFI of current DDF. After this step finishes, terminal returns to previous higher level of the tree, at the point it left off to traverse deeper, and continue from then.

This algorithm is known as Depth First Search (DFS). This is the most efficient algorithm used to traverse a Tree. As we know, EMV file system is viewed as a Tree, this is the best approach. A more detailed Java style pseudo code of the algorithm is listed in the detailed design chapter. The following example will summarise this chapter:

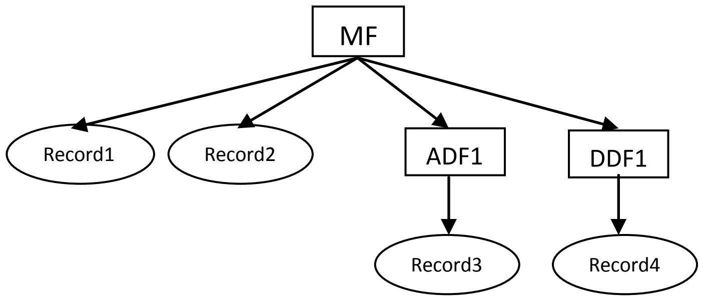


Figure 14: EMV file system example

The example figure above shows that EMV has four records. Record1 and Record2 are located right under MF, while Record3 is inside ADF1 folder and Record4 is under DDF1 folder. The algorithm starts at MF, step 1 is performed thus terminal reads the first record, then proceeds to the next item. Terminal realises the current item is indeed another record (Record2) so step 1 is repeated again. However, the next found item is an ADF1, according to the first branch, the algorithm goes to step 2, and the third record is read. Since there are no more records left under ADF1, algorithm returns to previous higher level of the tree, which is ADF1. Now it proceeds to the next item and finds DDF1, according to the second branch, algorithm goes to step 3, the fourth record is read. Algorithm again goes back to previous higher level of the tree. But now there are no more records left. Algorithm is finished, all records have been read.

7. High level design

This chapter sketches the logical structure of all components in the system at a high level design approach, from a general system overview to every detail inside each component. Initially, an overview of all user accessible functionalities is mentioned. Next, the architecture structure of the whole system is specified, followed by the architecture of involving hardware and software system. Then, a further step to look into software system is introduced by the design of each important Java class, with the aid of UML class diagrams. The chapter continues with description of the connection and interaction amongst those classes, featuring how information is passed and exchanged. The flow of data is illustrated by a sequence diagram. Next, a data dictionary is compiled to provide an insight of what type of data the system is handling. Finally, a screenshot of each main graphical user interface is captured to illustrate a better view of complete software. There is also a Class Responsibility Collaborators cards (CRC cards) section in Appendix C, which was used extensively when developing the high level design of the software system.

7.1 System functionalities overview

This section introduces all functionalities of the software system, which is accessible to the user. The software system incorporates six clear functionalities.

1) Smart card recognition.
2) Payment application finding.
3) Card authentication methods and cardholder verification methods survey.
4) Plaintext offline PIN verification application.
5) Personalised information collecting.
6) Report generation.

The first functionality of the system is the ability to recognise an inserted smart card in the reader device. The recognition process includes the tracing back of card vendor, checking working status and setting up a connection between terminal and ICC. The second functionality is the ability of locating payment applications existing on ICC and information of each application. Next, as one of the project objectives to address, the third functionality is designed to survey and learn information about card authentication methods and cardholder verification methods. Using gathered information from cardholder verification methods, terminal will decide if plaintext offline PIN verification can be performed in current ICC. This purely depends on the capability of ICC, and this makes the fourth functionality – practical plaintext offline PIN verification. The fifth functionality is designed to collect and analyse all personalised information in the card such as name on card, sixteen digits on card, expiry date, effective date, BIC and IBAN. Finally, a crucial functionality of survey software is the ability to generate a user-readable report. This makes the sixth functionality of the system.

7.2 General system architecture

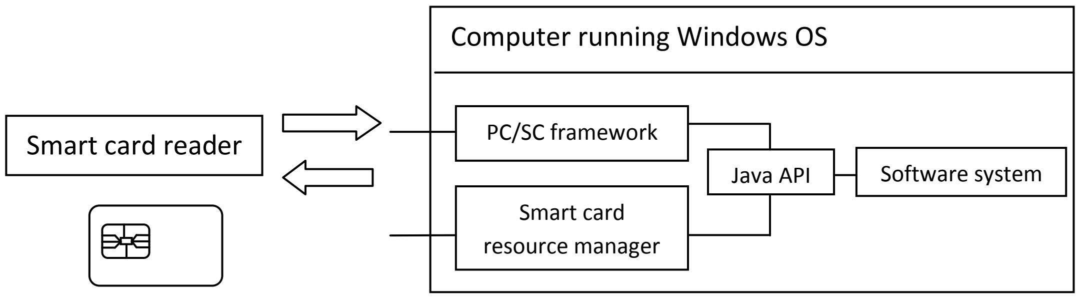


Figure 15: General system architecture

Figure 15 illustrates how the hardware component and software component interact with each other. PC/SC framework and smart card resource manager are provided under Windows operating system only. They handle the communication signals between PC/SC compatible smart card reader and any software system running under Windows. The software programming environment is Java. Java API provides a handy set of libraries to write software system under Java environment.

7.3 Hardware architecture

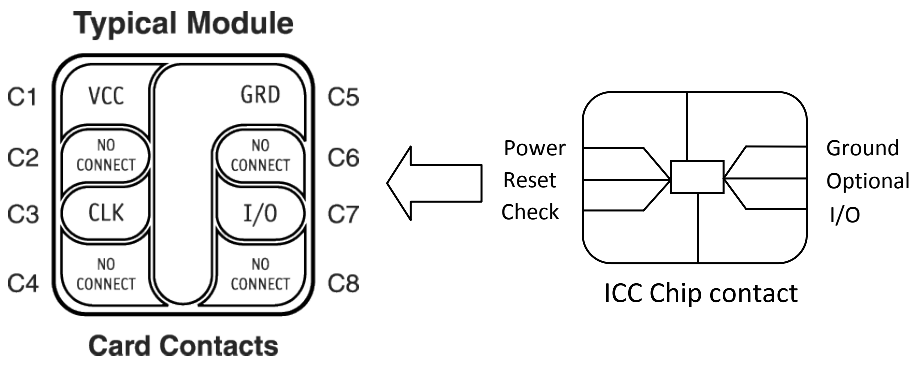


Figure 16: Smart card reader contact surface

This picture is taken from www.smartcardbasics.com website

Firstly, it is worth remembering that the smart card reader used in this project is bought from the market. The reader contact surface and smart card surface are illustrated in figure 16. The reader has a USB port connector, thus requires a USB port on the other side of the software system. The Chip part on the smart card must make contact with the reader surface on smart card reader. Electricity for smart card operations is supplied at Power point. Reset point is where reader sends signals to reset microprocessor inside smart card Chip. Data communication is performed at I/O point. This is only half duplex transportation, so at any time, reader is either sending or receiving information to/from Chip.

7.4 Software system architecture

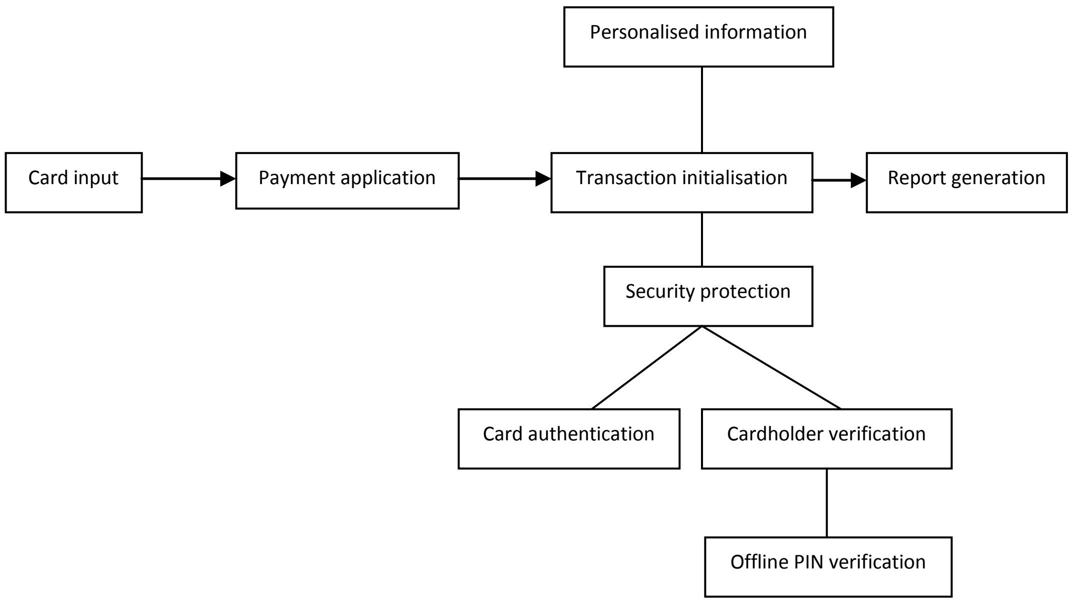


Figure 17: Software system architecture

Figure 17 illustrates the visible software components of the system to user. Each box represents the logical link amongst different functionalities of the system. User begins with ‘Card input’ page. This is where a connection is established between smart card reader and smart card. After that, control is redirected to ‘Payment application’ page. From here, user can indentify if there is a payment application on smart card. There are provided functions which allow all existing payment applications to be displayed and allows user to choose which application to start a new transaction. After a payment application is chosen, a new EMV transaction is initialised. Control is then switched to ‘Transaction initialisation’ page. When a transaction has been established, user can view personalised information which was hard-coded into the chip when manufactured or user can survey the two protection mechanism ‘card authentication’ and ‘cardholder verification’. Card authentication page displays supporting information of three card authentication methods, along with their data elements. Cardholder verification provides the same functionality, but includes a new practical function which performs real offline PIN verification with presented smart card. Finally, control is passed on to ‘Project generation’ page where user can export all surveyed information onto a text file. The system will ensure confidentiality for user information. Part of sensitive information such as Credit Card number and PIN number will be hidden. This information cannot be reversed and reports should be deleted after a certain period of time.

7.5 Sequence diagram

Figure 18 is an example of sequence diagram from a new session is created when EMV card is inserted until it is finished.

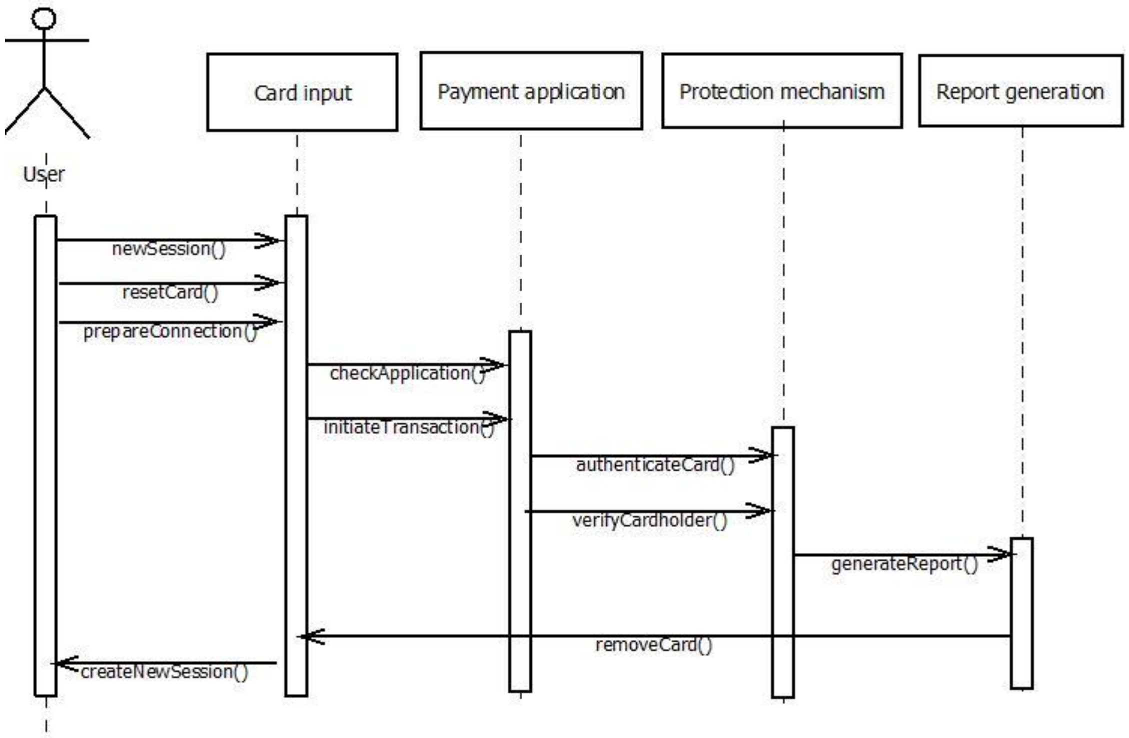
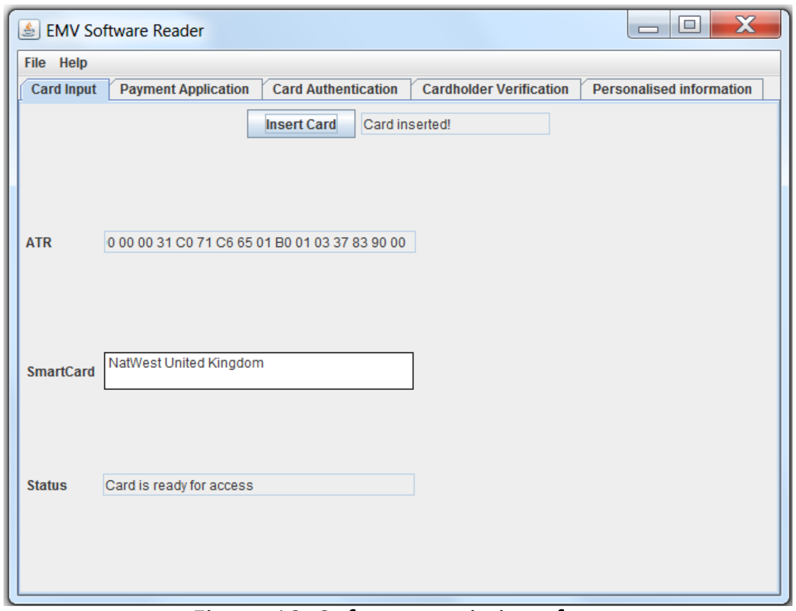


Figure 18: Session sequence diagram

7.6 User Interface

This section provides screenshots of all main pages of the software system, along with their descriptions.

7.6.1 Card input interface



Card inserted!

ATR 0 00 00 31 C0 71 C6 65 01 B0 01 03 37 83 90 00

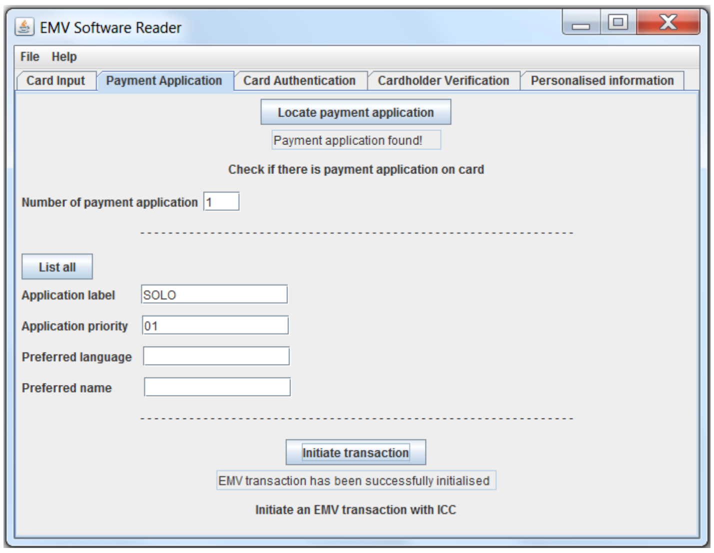
SmartCard NatWest United Kingdom

Status Card is ready for access

Figure 19: Software main interface

Figure 19 gives an example of first stage of system software, when user inserts a UK NatWest card into card reader and presses ‘Inset Card’ button.

7.6.2 Payment application interface



Payment application found!

Check if there is payment application on card

Number of payment application: 1

Application label: SOLO

Application priority: 01

Preferred language:

Preferred name:

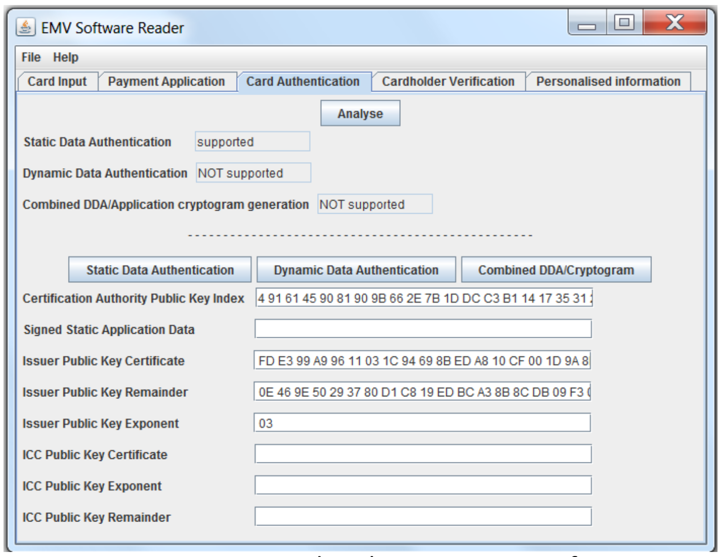
EMV transaction has been successfully initialised

Initiate an EMV transaction with ICC

Figure 20: Payment application interface

Figure 20 shows how payment application locating process can be performed. By clicking on ‘Locate payment application’ button, system will check if there is a payment application found on card. Pressing the ‘List all’ button will list all applications along with their attributes. In the figure, there is only one payment application found on the card, thus there is only one column for the application attributes. Finally, pressing ‘Initiate transaction’ will create an EMV transaction using the payment application found in the previous step. The status will be updated accordingly.

7.6.3 Card authentication interface



Static Data Authentication: supported

Dynamic Data Authentication: NOT supported

Combined DDA/Application cryptogram generation: NOT supported

Certification Authority Public Key Index: 4 91 61 45 90 81 90 9B 66 2E 7B 1D DC C3 B1 14 17 35 31

Signed Static Application Data:

Issuer Public Key Certificate: FD E3 99 A9 96 11 03 1C 94 69 8B ED A8 10 CF 00 1D 9A 8

Issuer Public Key Remainder: 0E 46 9E 50 29 37 80 D1 C8 19 ED BC A3 8B 8C DB 09 F3

Issuer Public Key Exponent: 03

ICC Public Key Certificate:

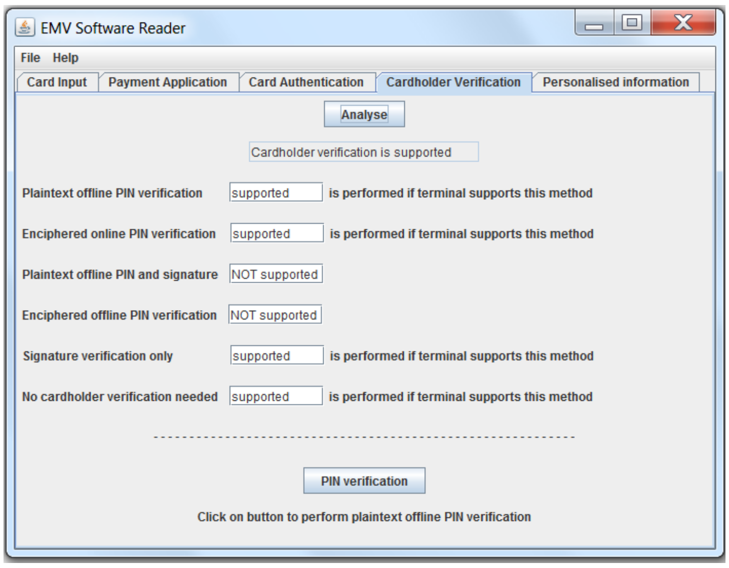
ICC Public Key Exponent:

ICC Public Key Remainder:

Figure 21: Card authentication interface

Figure 21 depicts the card authentication user interface. By pressing the 'Analyse' button, the system will tell what authentication method the ICC supports. Furthermore, user can explore additional data elements of each supported method by pressing the button of that method.

7.6.4 Cardholder verification interface



Cardholder verification is supported

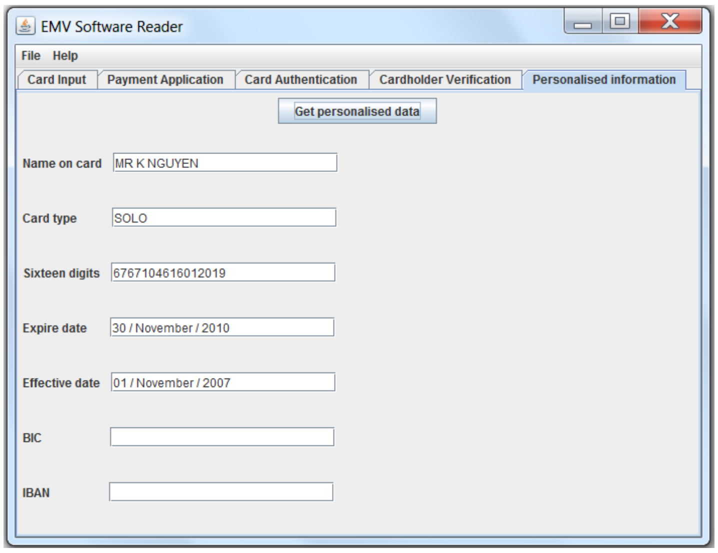
Plaintext offline PIN verification	supported	is performed if terminal supports this method
Enciphered online PIN verification	supported	is performed if terminal supports this method
Plaintext offline PIN and signature	NOT supported	
Enciphered offline PIN verification	NOT supported	
Signature verification only	supported	is performed if terminal supports this method
No cardholder verification needed	supported	is performed if terminal supports this method

Click on button to perform plaintext offline PIN verification

Figure 22: Cardholder verification interface

Figure 22 shows cardholder verification user interface. Similarly to card authentication interface, by pressing the 'Analyse' button, the system will update current supporting status of each verification method, and the condition in which the method can be applied.

7.6.5 Personalised information interface



Name on card	MR K NGUYEN
Card type	SOLO
Sixteen digits	6767104616012019
Expire date	30 / November / 2010
Effective date	01 / November / 2007
BIC	
IBAN	

Figure 23: Personalised information interface

Figure 23 shows the user interface of personalised information. By pressing the 'Get personalised data' button, all personalised information are collected.

7.6.6 Offline PIN verification interface

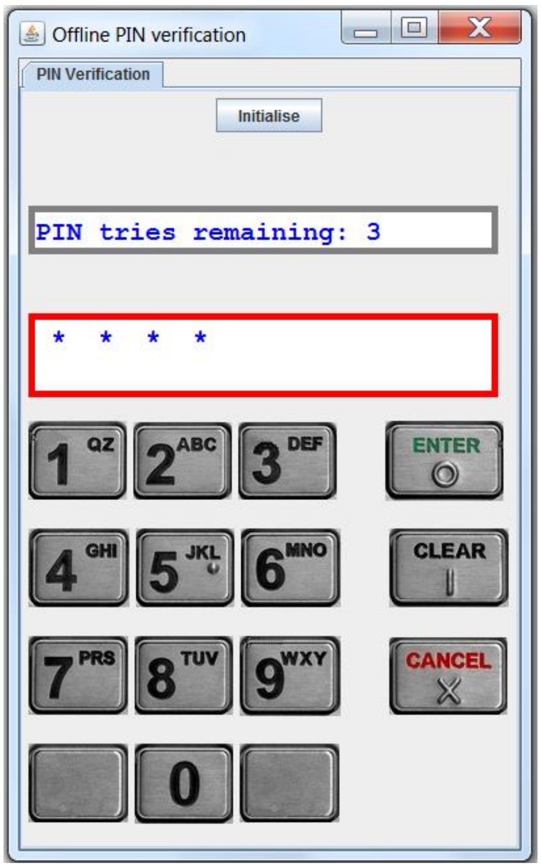


Figure 24: Offline PIN verification interface

Figure 24 shows offline PIN verification interface. The interface has been designed to look like the PIN pad found in the ATM machine. Besides ten digit keys from zero to nine to input PIN digits, users are provided with ENTER key to send the PIN to ICC for verification, CLEAR key to erase screen and re-enter PIN, and CANCEL key to 'eject' card and finish current transaction. The top grey screen shows the status of the current process, if the user enters a wrong PIN, the number will decrease. If such a number reaches one, it will signal the users a last try. 'Initialise' button will trigger terminal to get ready for offline PIN verification process.

8. Detailed design

This chapter is an expansion of the high level design chapter. Six functionalities of the software will be looked at very low level details in Java style pseudo code. Initially, a brief introduction about Java is given as this is the programming language used for pseudo code in this chapter. Next, each functionality of the system is explained in Java style pseudo code. The chapter is concluded by the pseudo code of the Java algorithm used in the project.

8.1 Java programming

Java language is originally famous for their portability and multi-platform. Yet for a long period of time, Java library has expanded to accommodate a variety of different functions. Furthermore, as a feature of open-source programming language, Java allows developers to write their own library functions. The software system utilises a set of Java APIs provided by JACCAL. In the subsequent sections, Java style pseudo code is used to demonstrate the low level implementation of each class of the software system.

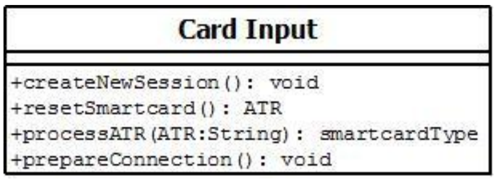
8.2 Classes specification

This section explains the design of important classes of software system, along with their functionalities. The order of class discussion also logically matches the flow of control and information illustrated in the software architecture part. Here is the list of seven important classes to be explained in this section and their contributions to six functionalities of the system listed in high level design chapter 7.1:

1) Card input class incorporates smart card recognition functionality.
2) Payment application class forms payment application finding functionality.
3) Transaction initialisation class does not incorporate an independent function, but it contributes to all survey functionalities in the software.
4) Card authentication class performs card authentication methods survey functionality.
5) Cardholder verification class performs cardholder verification methods survey functionality.
6) Plaintext offline PIN verification class performs plaintext offline PIN verification application.
7) Personalised information class incorporates personalised information collecting functionality.
8) Report generation class incorporates report generation functionality.

8.2.1 Card input class

Table 7
Card input class



Card Input
+createNewSession(): void +resetSmartcard(): ATR +processATR(ATR:String): smartcardType +prepareConnection(): void

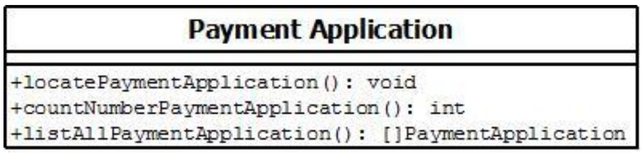
Table 7 illustrates card input class. This class creates a fresh session for each detected smart card by using *createNewSession()* method. The creation process uses SessionFactory object provided by JACCAL. It provides an instance on each request. The whole session prevails inside this instance. When smart card is no longer used, the session will be terminated by Java garbage collection mechanism. This simply involves closing the connection between reader and software, and clearing the occupied memory used by software variables. At the beginning, software system uses *open()* method to send an instruction to reset the ICC, if successful, smart card will reply with an Answer To Reset sequence (ATR). This ATR sequence will be casted to String format with *toString()* method provided by Java for analysis purposes. This ATR sequence is unique for each card vendor, thus makes it possible to trace back the vendor with *processATR()* function. This function simply looks up the ATR sequence with the format specified in smartcard_list text file. Since this pre-made text file is published under GNU license, the original format will be prevailed and the software system will adapt to smartcard_list format. Finally, software system ensures the connection is correctly established by checking the status of ICC with *prepareConnection()* function.

The following Java style pseudo code demonstrates the solution ideas:

```
public class CardInput
{

        public void createNewSession()
        {
                f = SessionFactory.getInstance();
                // open a new session with EMV card
                se = f.createSessions();
        }

        public ATR resetSmartcard()
        {
                for(int i = 0; i < se.length; i++)
                {
                        // reset EMV card
                        Atr atr = se[i].open();
                        atrValue = atr.toString();
                }
                return atrValue;
        }

        public void processATR(String atr)
        {
                // process this ATR sequence to trace back card vendor

                FileReader dataIn = new FileReader ("smartcard_list.txt");
                BufferedReader f = new BufferedReader(dataIn);

                String line;
                while (true)
                {
                        while (true)
                        {
                            line = f.readLine();
                            if (!line.isEmpty() && line.charAt(0) >= '0'
                              && line.charAt(0) <= '9') break;
                            if (line.equals("# do not delete")) break;
                        }
                        if (line.equals("# do not delete")) break;

                        if (atrCard.equals(line))
                            while (true)
                            {
                                line = f.readLine();
                                line = line.trim();
                                if (line.isEmpty()) break;
                                System.out.println(line);
                                cardManufacturer += line + "\n";
                            }
                }
        }

        public void prepareConnection()
        {
                // prepare connection between software and ICC
        }

}
```

8.2.2 Payment application class

Table 8
Payment application class



Payment Application
+locatePaymentApplication(): void +countNumberPaymentApplication(): int +listAllPaymentApplication(): []PaymentApplication

Table 8 describes payment application class and its methods. This class handles application selection procedure. Firstly, it verifies if there exists a payment application container in the card with *locatePaymentApplication()* function. This function makes use of SELECT command described in chapter 5.2.1. If there is such container, the software system will select the folder by calling *countNumberPaymentApplication()* function. This function makes use of READ RECORD command described in chapter 5.2.2 and returns the total number of applications inside this folder as an integer number. Finally, *listAllPaymentApplication()* function will look inside each payment application and record its attributes such as application label, application priority and application preferred name. The method returns all recorded information as an array of payment applications to be used to create a transaction in the next part.

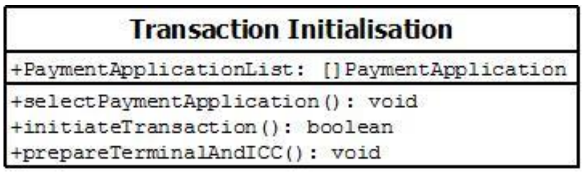
The following Java style pseudo code demonstrates the solution ideas:

```
public class PaymentApplication
{
        public void locatePaymentApplication()
        {
                capdu = new ApduCmd ("00 A4 04 00 0E 31 50 41 59 2E 53
                59 53 2E 44 44 46 30 31");

                rapdu = se[0].execute(capdu);
                if (rapdu.getStatusWord().isSuccess())
                        tx1.setText("Payment application found!");
                else
                {
                        tx1.setText("Payment application NOT found!");
                        return;
                }

                capdu = new ApduCmd("00 B2 01 0C 00");
                rapdu = se[0].execute(capdu);
                String size = NumUtil. hex2String (rapdu.getStatusWord().
                getSw2());

                // concat with byte length obtained above to get PSE data
                capdu = new ApduCmd("00 B2 01 0C" + size);
                rapdu = se[0].execute(capdu);

                processApplication(rapdu.toString());

        }

        public int countNumberPaymentApplication()
        {
                // counter number of applications
        }


        public paymentApplication[] listAllPaymentApplication()
        {
                // returns list of payment applications
        }

}
```

8.2.3 Transaction initialisation class

Table 9

Transaction initialisation class



```
Transaction Initialisation
+PaymentApplicationList: []PaymentApplication
+selectPaymentApplication(): void
+initiateTransaction(): boolean
+prepareTerminalAndICC(): void
```

When a list of all payment transactions has been compiled, an EMV transaction can be initialised. The reason a transaction must be created is because that is the only way information can be surveyed. Under no other circumstances, ICC agrees to release information without transaction establishment. By looking at the application priority found in the payment application list, software system can decide which application would be used for creating a transaction. In practice, the one with the highest priority (level 1) is selected by calling *selectPaymentApplication()* function. This function makes use of SELECT command. The active entity in this procedure is software system, which requests a transaction to be created, thus sending a list of environmental conditions, business restrictions to the passive entity – smart card using *initiateTransaction()* function. On receiving the list, ICC adapts to it and responds with either an agreement to proceed with the transaction or some minor tweaks needed in order to continue. If transaction is successfully initialised, this function returns boolean value 'true', otherwise it returns 'false' to signal failure. Software system will react to this agreement and the transaction is finalised with *prepareTerminalAndICC()* function. These functions are encapsulated in transaction initialisation class described in table 9.

The following Java style pseudo code demonstrates the solution ideas:

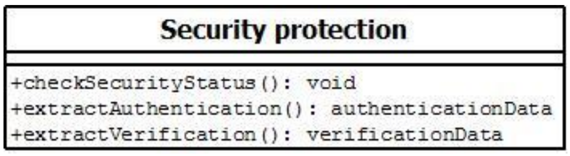
```
public class TransactionInitialisation
{

        public void selectPaymentApplication()
        {
                // scan through the array of payment application
                // select payment application with highest priority
                capdu = new ApduCmd("00 A4 04 00 07" + AID);

                rapdu = se[0].execute(capdu);
                capdu = new ApduCmd("80 A8 00 00 02 83 00");
                rapdu = se[0].execute(capdu);
        }

        public boolean initiateTransaction()
        {
                // check if transaction has been initiated successfully
                if (rapdu.statusWord.isSuccess())
```

```
                        transaction = true;
                else
                        transaction = false;

                return transaction;
        }


        public void prepareTerminalAndICC()
        {
                // returns list of payment applications
        }

}
```

8.2.4 Security protection class

Table 10

Security protection class



```
Security protection
+checkSecurityStatus(): void
+extractAuthentication(): authenticationData
+extractVerification(): verificationData
```

Table 10 illustrates security protection class and its methods. This class is only performed after a transaction is initiated. Based on current security and business environment, transactions may be required to be acknowledged online via issuer's network or locally offline between merchant terminal and ICC. *checkSecurityStatus()* function of the class will gather environment status and verify this. Next, *extractAuthentication()* and *extractVerification()* functions take responsibility to extract relevant security information for card authentication and cardholder verification procedures, and then converts them into the correct data format for authentication and verification processes to be continued.

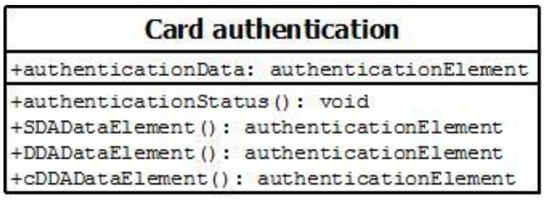
The following Java style pseudo code demonstrates the solution ideas:

```
public class SecurityProtection
{

        public void checkSecurityStatus()
        {
                // gather security information
                // check existence of CVM list and AIP
        }

        public authenticationData extractAuthentication()
        {
                // passing the data on for authentication procedure

        }

        public verificationData extractVerification()
        {
                // passing the data on for verification procedure
        }
}
```

8.2.5 Card authentication class

Table 11

Card authentication class



```
Card authentication
+authenticationData: authenticationElement
+authenticationStatus(): void
+SDADataElement(): authenticationElement
+DDADataElement(): authenticationElement
+cDDADataElement(): authenticationElement
```

Table 11 illustrates card authentication class. This class handles card authentication processes. On receiving authentication data from Security protection class, software can analyse the authentication status to find out the support status of Static Data Authentication, Dynamic Data Authentication and combined Dynamic Data Authentication/application cryptogram generation by calling *authenticationStatus()* method. This method simply analyses the second bit, third bit and seventh bit of AIP sequence as described in chapter 4.4.1. Those methods supported by ICC will be further analysed for their data elements by *SDADataElement()* method, *DDADataElementMethod()* and *cDDADataElement()* method respectively. Each of the three methods returns authenticationElement which contains data elements found on ICC.

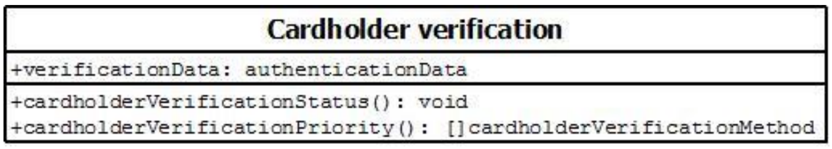
The following Java style pseudo code demonstrates the solution ideas:

```
public class CardAuthentication
{

        public void authenticationStatus()
        {
                // convert hex to Dec
                int decimal = hexToDec(AIP);
                // convert Dec to Binary
                st = Integer.toBinaryString(decimal);
                // append 0 at the beginning to restore 8 bit as
                toBinaryString cut all zero at front
                while (st.length() < 8)
                        st = "0" + st;
                if (st.charAt(1) == '1') SDA = true; else SDA = false;
                if (st.charAt(2) == '1') DDA = true; else DDA = false;
                if (st.charAt(6) == '1') cDDA = true; else cDDA = false;
        }

        public authenticationElement SDAElement()
        {
                int index, length;

                // get CertificationAuthorityPublicKeyIndex, tag 8F
                index = AFL.indexOf("8F");
                if (index != -1)
                {
                        length = hexToDec(AFL.substring(index+2,index+4));

                        // multiple 2 since it is 2 bytes per hexa
                        CertificationAuthorityPublicKeyIndex  =  AFL.substring
                        (index + 4, index + 4 + length * 2);
                }

                // get Issuer Public Key Certificate, tag 90
                index = AFL.indexOf("90");
                if (index != -1)
                {
                        length = hexToDec(AFL.substring(index+2,index+4));
                        // multiple 2 since it is 2 bytes per hexa
                        IssuerPublicKeyCertificate = AFL.substring(index + 4,
                        index + 4 + length * 2);
                }

                //Signed Static Application Data, tag 93
                index = AFL.indexOf("93");
                if (index != -1)
                {
                        length = hexToDec(AFL.substring(index+2,index+4));
                        // multiple 2 since it is 2 bytes per hexa
                        SignedStaticApplicationData  =  AFL.substring(index  +
                        4, index + 4 + length * 2);
                }

                // Issuer Public Key Remainder, tag 92
                index = AFL.indexOf("92");
                if (index != -1)
                {
                        length = hexToDec(AFL.substring(index+2,index+4));
                        // multiple 2 since it is 2 bytes per hexa
                        IssuerPublicKeyRemainder = AFL.substring(index + 4,
                        index + 4 + length * 2);
                }
```

```
                // Issuer Public Key Exponent, tag '9F32'
                index = AFL.indexOf("9F32");
                if (index != -1)
                {
                        length = hexToDec(AFL.substring(index+4,index+6));
                        // multiple 2 since it is 2 bytes per hexa
                        IssuerPublicKeyExponent = AFL.substring(index + 6,
                        index + 6 + length * 2);
                }
        }

        public authenticationElement DDAElement()
        {
                // retrieve DDA data elements
        }

        public authenticationElement cDDAElement()
        {
                // retrieve combined DDA data elements
        }

}
```

8.2.6 Cardholder verification class

Table 12
Cardholder verification class



Cardholder verification
+verificationData: authenticationData
+cardholderVerificationStatus(): void +cardholderVerificationPriority(): []cardholderVerificationMethod

Table 12 describes cardholder verification class. This class controls cardholder verification procedure. Similarly to Card authentication class, Cardholder verification class receives protection data from Security protection class. It analyses this data to check the status of seven different methods including plaintext offline PIN verification, enciphered online PIN verification, plaintext offline PIN verification and signature, enciphered offline PIN verification, enciphered offline PIN and signature, signature verification only and no cardholder verification needed using *cardholderVerificationStatus()* method. Those statuses can be determined by checking CVM list. The condition in which the method is applied is also recorded by the function. Furthermore, each EMV card has a different priority order in which each cardholder verification method is applied after another. *cardholderVerificationPriority()* function will analyse this priority order and returns an array containing cardholder verification methods in their decreasing order of priority.

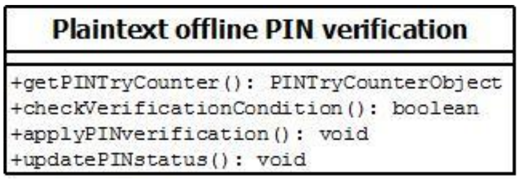
The following Java style pseudo code demonstrates the solution ideas:

```
public class CardholderVerification
{

        public void cardholderVerificationStatus()
        {
                String CVR = "", rule = "", condition = "";
                int index, length;

                // get CVM list, tag 8E
                index = AFL.indexOf("8E");
                if (index != -1)
                {
                        length = hexToDec(AFL.substring(index+2,index+4));
                        // multiple 2 since it is 2 bytes per hexa
                        CVR = AFL.substring(index+4,index+4+length * 2);

                        // ignore the first 8 bytes of X and Y
                        CVR = CVR.substring(16);
                }

                index = 0;
                while (index < CVR.length())
                {
                        rule = CVR.substring(index, index + 2);
                        condition = CVR.substring(index + 2, index + 4);

                        // convert to 8 bit binary & trim the 6 bit right most
                        rule = hexToBin(rule).substring(2);

                        if (condition.equals("00"))
                                condition = "will always be performed";
                        else
                        if (condition.equals("01"))
                                condition = "is performed if unattended cash";
                        else
                        if (condition.equals("02"))
                                condition  =  "is  performed  if  not  unattended
                                cash,  not  manual  cash  and  not  purchase  with
                                cashback";
                        else
                        if (condition.equals("03"))
                                condition  =  "is  performed  if  terminal  supports
                                this method";
                        else
                        if (condition.equals("04"))
                                condition  =  "is  performed  if  customer  pays
                                with manual cash";
                        else
                        if (condition.equals("05"))
                                condition = "is performed if transaction with
                                cashback";
                        else
                        condition = "";

                        if (rule.equals("000001"))
                        {
                        Plaintext_offline_PIN_verification = "supported";
                        cPlaintext_offline_PIN_verification = condition;

                        }
                        else
                        if (rule.equals("000010"))
                        {
                        Enciphered_online_PIN_verification = "supported";
                        cEnciphered_online_PIN_verification = condition;

                        }
                        else
                        if (rule.equals("000011"))
                        {
                        Plaintext_offline_PIN_and_signature = "supported";
                        cPlaintext_offline_PIN_and_signature = condition;
                        }
                        else
                        if (rule.equals("000100"))
                        {
                        Enciphered_offline_PIN_verification = "supported";
                        cEnciphered_offline_PIN_verification = condition;
                        }
                        else
                        if (rule.equals("000101"))
                        {
                        Enciphered_offline_PIN_and_signature = "supported";
                        cEnciphered_offline_PIN_and_signature = condition;
                        }
                        else
                        if (rule.equals("011110"))
                        {
                        Signature_verification_only = "supported";
                        cSignature_verification_only = condition;
                        }
                        else
                        if (rule.equals("011111"))
                        {
                        No_cardholder_verification_needed = "supported";
                        cNo_cardholder_verification_needed = condition;
                        }
```

```
                        index += 4;
                }

        }

        public []cardholderVerificationMethod cardholderVerificationPriority()
        {
                // returns list of cardholder verification methods
        }

}
```

8.2.7 Plaintext offline PIN verification

Table 13
Plaintext offline PIN verification



Plaintext offline PIN verification

```
+getPINTryCounter(): PINTryCounterObject
+checkVerificationCondition(): boolean
+applyPINverification(): void
+updatePINstatus(): void
```

Table 13 describes plaintext offline PIN verification class. This class performs offline PIN verification application if the card supports this method. Initially, the terminal needs to decide if customer is allowed to enter PIN, and if possible, how many attempts they are allowed to do so. getPINTryCounter() method obtains PIN Try counter object in ICC with GET DATA command. After all required information has been satisfied and *checkVerificationCondition()* method return 'true', terminal prompts user to input 4-digit PIN. This PIN is sent clear to ICC for verification by *applyPINverification()* method. The method firstly encodes the 4-digit PIN in correct format described in chapter 4.4.3. After that, the verification procedure is carried out independently by ICC. If customer PIN does not match reference PIN, ICC will decrease PIN Try counter, if customer PIN matches reference PIN, transaction is allowed to continue. ICC will inform terminal the verification result and terminal updates the status with user through *updatePINstatus()* method.

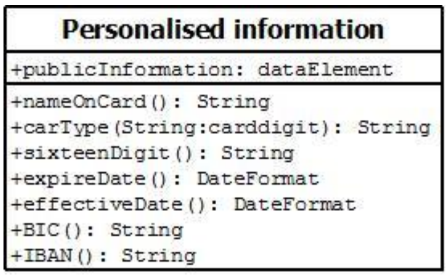
The following Java style pseudo code demonstrates the solution ideas:

```
public class PlaintextOfflinePINVerification
{

        // extract the PIN Try Counter number
        public int getPINTryCounter()
        {
                StringBuilder s = new StringBuilder();
                try{
                System.out.println("GETTING PIN TRY COUNTER");
                capdu = new ApduCmd("80 CA 9F 17 00");
        rapdu = se[0].execute(capdu);

        System.out.println("Try again with new length");
        capdu = new ApduCmd("80 CA 9F 17 04");
        rapdu = se[0].execute(capdu);

        s = new StringBuilder(rapdu.toString());

        for (int i=0; i < s.length(); i++)
                if (s.charAt(i) == ' ')
                        s.deleteCharAt(i);

        } catch (CardException e)
        {
                System.out.println("Error   when   getting   PIN   try
                counter!");
        }
        String st = s.toString();
        return  Integer.parseInt(st.substring(st.indexOf("9F17")  +  6,
        st.indexOf ("9F17") + 8));
}

// send PIN to ICC
public boolean applyPINverification(String pin)
{
        try{
        System.out.println("APPLYING VERIFY COMMAND");

        String command = "00 20 00 80 08 24" + pin;
        while (command.length() < 26) //14 + 12 = 26
                command += "F";
        System.out.println(command);
        capdu = new ApduCmd(command);
        rapdu = se[0].execute(capdu);
        if (rapdu.toString().substring(5, 10).equals("90 00"))
                return true;
        } catch (CardException e)
        {
                System.out.println("Error when sending PIN to ICC");
        }
        return false;
}


public boolean checkVerificationCondition()
{
        // check if users can go on with PIN verification
}
```

```
            public void updatePINstatus()
            {
                    // informs user if entered PIN matches or not
            }

}
```

8.2.8 Personalised information class

Table 14
Personalised information class



```
Personalised information
+publicInformation: dataElement
+nameOnCard(): String
+carType(String:carddigit): String
+sixteenDigit(): String
+expireDate(): DateFormat
+effectiveDate(): DateFormat
+BIC(): String
+IBAN(): String
```

This Personalised information class provides methods to obtain personalised information as described in table 14. This is public information needed to complete a transaction. All of them can be accessed by their specific tags, for example, name on card is signalled by tag '5F 20', followed by the length of content and then the real content of cardholder name. Because of the same access pattern, the following Java style pseudo code will show the methods to obtain four public pieces of personalised information only including name on card, card sixteen digits, BIC and IBAN.

The following Java style pseudo code demonstrates the solution ideas:

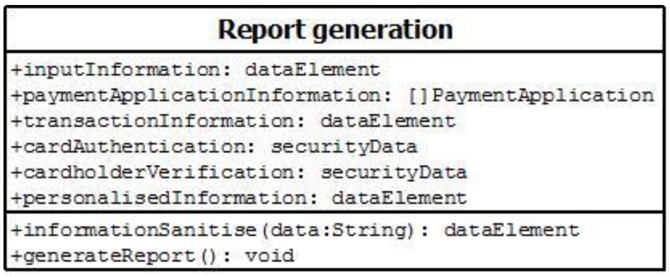
```
public class PersonalisedInformation
{

            public String nameOnCard()
            {
                    // get Card Holder name, tag 5F 20
                    if (st.indexOf("5F20") != -1)
                    {
                            int nameLength = hexToDec (st.substring (st.indexOf
                            ("5F20") + 4, st.indexOf("5F20") + 6 ));
                            nameOnCard = st.substring(st.indexOf("5F20") + 6,
                            st.indexOf("5F20") + 6 + nameLength * 2);

                            // covert the hexa string nameOnCard will give a
                            meaningful ASCII name
                            nameOnCard = hexToASCII(nameOnCard);
                    }
                    return nameOnCard;
```

```
        }

        public String sixteenDigit()
        {
                // 16 digits
                index = st.indexOf("5A");
                if (index != -1)
                {
                        sixteenDigit = st.substring(index + 4, index + 20);
                }
                return sixteenDigit;
        }

        public String BIC()
        {
                // get BIC, tag 5F 54
                if (st.indexOf("5F54") != -1)
                {
                        int nameLength = hexToDec(st.substring (st.indexOf
                        ("5F54") + 4, st.indexOf("5F54") + 6 ));
                        BIC = st.substring(st.indexOf("5F54") + 6, st.indexOf
                        ("5F54") + 6 + nameLength * 2);
                }
        }

        public String IBAN()
        {
                // get IBAN, tag 5F 53
                if (st.indexOf("5F53") != -1)
                {
                        int nameLength = hexToDec(st.substring(st.indexOf
                        ("5F53") + 4, st.indexOf("5F53") + 6 ));
                        IBAN = st.substring(st.indexOf("5F53") + 6,
                        st.indexOf("5F53") + 6 + nameLength * 2);
                }
        }

}
```

8.2.9 Report generation class

Table 15

Report generation class



Report generation
+inputInformation: dataElement +paymentApplicationInformation: []PaymentApplication +transactionInformation: dataElement +cardAuthentication: securityData +cardholderVerification: securityData +personalisedInformation: dataElement
+informationSanitise(data:String): dataElement +generateReport(): void

The Report generation class provides a facility to generate a user-readable report via its methods as described in table 15. Sensitive information such as Credit Card number, PIN number are handled with care and asterisk mark will be used to conserve information integrity and prevent information to be reversed. Information is collected from a global variable set by other classes throughout survey processes. Because of the similarity amongst output procedures, only preliminary input information, payment applications, card authentication and personalised information survey are displayed. *informationSantitise()* method receives data from other facilities and dispose the stored memory after generating a report for security reasons. Local information will be cleaned automatically by Java garbage collector facility.

The following Java style pseudo code demonstrates the solution ideas:

```
public class ReportGeneration
{

            public void informationSanitise(String data)
            {
                    // dispose used occupied memory by data
            }

            public void generateReport()
            {
                    PrintWriter f = new PrintWriter("report.txt");
                    f.println("*** EMV card survey report");

                    f.println("Card is reseted, the ATR value returned is " +
                    atrValue);

                    if (!cardManufacturer.isEmpty())
                            f.println("Card manufacturer is "+cardManufacturer);
                    else
                            f.println("Cannot trace back card manufacturer");


                    f.println("* * * * * * * * * * * * * * * * * * * * * * *");
                    f.println("*    Card payment applications survey     *");
                    f.println("* * * * * * * * * * * * * * * * * * * * * * *");
                    f.println();
                    f.println("Payment application folder exists in this card");
                    if (numberOfApplications == 1)
                            f.println("There is only one payment application on
                            this card");
                    else
                            f.println("There are " + numberOfApplications + "
                            payment applications on this card");

                    f.println("The list of all payment applications:");
```

```
f.println("- Application label is " + applicationLabel);
f.println("- Application priority is " + applicationPriority);
if (!applicationName.isEmpty())
        f.println("- Application preferred name is " +
        applicationName);
if (!applicationLanguage.isEmpty())
        f.println("- Application preferred language is " +
        applicationLanguage);


f.println("* * * * * * * * * * * * * * * * * * * * * *");
f.println("*   Card authentication methods survey    *");
f.println("* * * * * * * * * * * * * * * * * * * * * *");
f.println();
f.println("Card authentication methods checking: ");
if (SDA)
        f.println("- Static data authentication is supported");
else
        f.println("- Static data authentication is NOT
        supported");

if (DDA)
        f.println("- Dynamic data authentication is
        supported");
else
        f.println("- Dynamic data authentication is NOT
        supported");

if (cDDA)
        f.println("- Combined dynamic data authentication
        and application cryptogram generation is supported");
else
        f.println("- Combined dynamic data authentication
        and application cryptogram generation is NOT
        supported");

if (!CertificationAuthorityPublicKeyIndex.isEmpty())
        f.println("- Certification Authority Public Key Index is "
        + CertificationAuthorityPublicKeyIndex);
if (!IssuerPublicKeyCertificate.isEmpty())
        f.println("- Issuer Public Key Certificate is " +
        IssuerPublicKeyCertificate);
if (!SignedStaticApplicationData.isEmpty())
        f.println("- Signed Static Application Data is " +
        SignedStaticApplicationData);
if (!IssuerPublicKeyRemainder.isEmpty())
```

```
                    f.println("- Issuer Public Key Remainder is " +
                    IssuerPublicKeyRemainder);
                if (!IssuerPublicKeyExponent.isEmpty())
                    f.println("- Issuer Public Key Exponent is " +
                    IssuerPublicKeyExponent);


                f.println("* * * * * * * * * * * * * * * * * * * * * * *");
                f.println("*     Personalised information survey       *");
                f.println("* * * * * * * * * * * * * * * * * * * * * * *");
                f.println("Cardholder name is " + nameOnCard);
                f.println("Card type is " + cardType);
                // Hide away 8 digits of card for security reason
                f.println("Card sixteen digits are " + sixteenDigit.substring(0,
                4) + "****" + sixteenDigit.substring(8, 12) + "****");
                f.println("This card is effective from " + beginDate + " / " +
                beginMonth + " / " + beginYear);
                f.println("This card will expire on " + expireDate + " / " +
                expireMonth + " / " + expireYear);
                if (!BIC.isEmpty())
                    f.println("The bank\\'s BIC is " + BIC);
                if (!IBAN.isEmpty())
                    f.println("The bank\\'s IBAN is " + IBAN);

                f.close();
        }

}
```

8.2.10 Graphical User Interface class

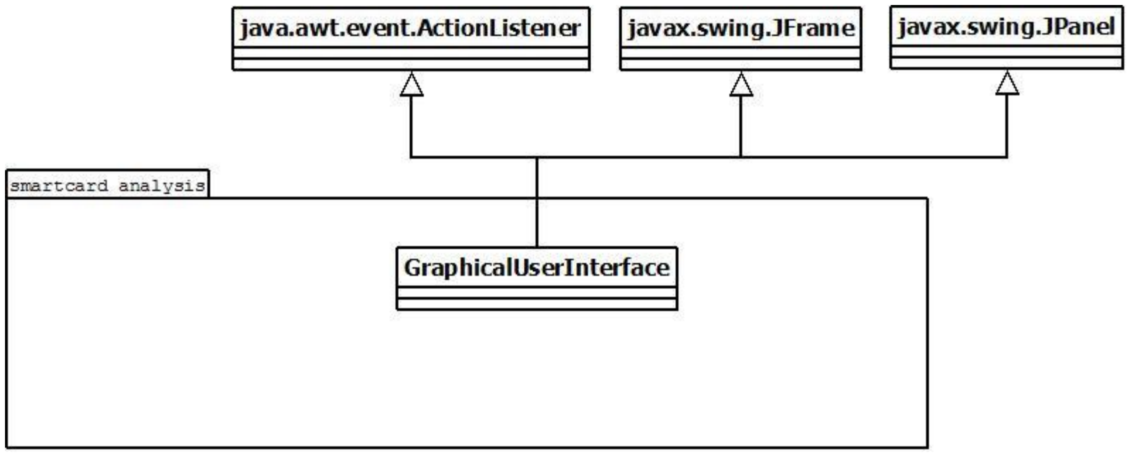


Figure 25: Graphical User Interface class

This class provides user interface for software system. It inherits three standard Java classes from Java library including JFrame and JPanel classes for buttons, labels, text fields, text boxes display, ActionListener for mouse clicking event, text boxes triggers. Figure 25 describes this class.

8.3 Data dictionary

This part gives a brief overview of the data type used in the software system.

- PaymentApplication: this data type stores information about a particular payment application including label, priority, preferred name, and preferred language.
- AuthenticationData: this data type stores information used for card authentication procedure.
- VerificationData: this data type stores information used for cardholder verification procedure.
- DateFormat: this data type stores information for expiry date and effective date in dd/mm/yy format.
- String: this is standard Java data type representing a sequence of characters.
- int: this is standard Java data type representing an integer number.
- boolean: this is standard Java data type representing two logical value true or false.
- PINTryCounterObject: this is special data type specified by ICC. It contains a number specified how many times a PIN can be entered.

8.4 Java algorithm

This section describes the Java algorithm used throughout the solution to look into EMV file system tree for application file and folder. This algorithm is described in more detail in chapter 6 – EMV communication protocol construction.

The Algorithm we use above is called Depth First Search (DFS). This is the most efficient algorithm used to traverse a Tree. As we know, EMV file system is viewed as a Tree, this is the best approach.

The process will begin when we call Depth_First_Search_EMV(PSE folder). The Java style pseudo code of the algorithm is as followed:

```
// Perform Depth First Search with folder root and index (AID or SFI)
Depth_First_Search_EMV( root MasterFile, integer index )
{
        // perform SELECT command for this folder (root) with given AID
        SELECT(root, inde);
        // perform READ RECORD command for each record inside root
        READ_RECORD(root);

        while (Status_Word != '6A 83')     // keep repeating until no more
        records found
        {
              Save_Record(record);         // save this record into a list
```

```
                // if an ADF is found, jump to that ADF (one-level deeper on
                the tree)
                if (record == ADF)
                        Depth_First_Search_EMV(record, AID);
                else
                // if a DDF is found, jump to that DDF (one-level deeper on
                the tree)
                if (record == DDF)
                        Depth_First_Search_EMV(record, SFI);
        }
}
```

9. Software Testing

This chapter demonstrates the results of the physical tests carried out on the system during testing phrase. The testing plan is systematically divided into four main parts -

- Black box testing: this type of testing considers each class of the program as a black box. It tests every function and methods of each class based on its attributes. This type of testing is called ‘black box’ since it only applies to input forms and method signatures, it does not test internal structure of each class.
- White box testing: In contrast to black box testing, white box testing is executed based on knowledge of the internal code of the program. The main advantage of white box testing is the ease of fixing logical erroneous since each test case can spot exactly what lines of code cause the problems. However, in return, for white box testing to be continued, a thorough understanding of program source code is required. It is also worth noting that any small alternation of code would require an appropriate change to white box testing to reflect the modification.
- Integration testing: this testing uses bottom-up approach. The test cases are carried out as the software is examined by user. Each methods of each class are tested in the way data flows.
- System testing: this type of testing verifies the completed system against the requirements and functionality to ensure it matches the specification. This type of testing is similar to the way black box testing approaches. Internal structure of the program is not considered.

9.1 Black box testing

Each black box test procedure contains three separate test cases: Normal test, Extreme test and Erroneous test. Normal test cases are easy tests, assuming users following precisely what software specification has written. Stress tests concentrate on testing boundary input values. These are the very maximum and very minimum input values software can handle. And finally, Erroneous tests challenges the software ability to handle unexpected situations caused by user carelessness. Software system will be tested in a variety of circumstances when input data are totally wrong or inconsistent with what have been written in specification.

Card input class

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
1	void createNewSession()	N/A as no data is parsed through this method	A new session is created	Test successful. A new session is created
2	ATR resetSmartcard()	N/A as no data is parsed through this method	Smart card is reset and an ATR sequence is returned	Test successful. Smart card is reset and an ATR sequence is returned
3	smartcardType processATR(String ATR)	Normal:- ATR = “2A 07 0C 1C” Extreme:- ATR = “” (empty string) ATR = “48 55 4F 4E 47 20 ... 54 60 58 ” (256 bytes in length) Erroneous:- N/A	Normal:- processATR function returns correct smart card type Extreme:- processATR function returns correct smart card type Erroneous:- processATR	Test successful. processATR function looks up ATR sequence in smart_card_list text file and returns correct smart card type if found.

			function returns correct smart card type	
4	void prepareConnection()	N/A as no data is parsed through this method	A connection is created between software system and ICC	Test successful. A connection is created between software system and ICC

Payment application class

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
5	boolean locatePaymentApplication()	N/A as no data parsed through this method	This function returns 'true' if a payment application is found, or 'false' if no payment application is found.	Test successful
6	int countNumberPaymentApplicat ion()	N/A as no data parsed through this method	This function returns total number of payment application found on card.	Test successful
7	PaymentApplication[] listAllPaymentApplication()	N/A as no data parsed through this method	This function returns a list of all payment applications found on card.	Test successful

Transaction initialisation class

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
8	void selectPaymentApplication()	N/A as no data parsed through this method	This function selects payment application with highest priority.	Test successful
9	boolean initiateTransaction()	N/A as no data parsed through this method	This function returns ‘true’ if a transaction is successfully created, and ‘false’ otherwise.	Test successful
10	void prepareTerminalAndICC()	N/A as no data parsed through this method	This function configures terminal to adapt to conditions specified by ICC in response to transaction creation.	Test successful

Security protection class

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
11	void checkStatusStatus()	N/A as no data parsed through this method	This function checks security status of ICC and decides if card authentication and	Test successful

			cardholder verification will be performed.	
12	authenticationData extractAuthentication()	N/A as no data parsed through this method	This function returns authentication data retrieved from security data.	Test successful
13	verificationData extractVerification()	N/A as no data parsed through this method	This function returns verification data retrieved from security data.	Test successful

Card authentication class

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
14	void authenticationStatus()	N/A as no data parsed through this method	This function updates the status of three authentication methods	Test successful
15	authenticationElement SDADataElement()	N/A as no data parsed through this method	This function returns SDA data elements	Test successful
16	authenticationElement DDADataElement()	N/A as no data parsed through this method	This function returns DDA data elements	Test successful

17	authenticationElement cDDAElement()	N/A as no data parsed through this method	This function returns cDDA data elements	Test successful

Cardholder verification class

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
18	void cardholderVerificationStatus()	N/A as no data parsed through this method	This function updates the status of eight verification methods	Test successful
19	[]cardholderVerificationMetho d cardholderVerificationPriority()	N/A as no data parsed through this method	This function returns the priority order in which verification methods are applied in ICC.	Test successful

Plaintext offline PIN verification

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
20	PINTryCounterObject getPINTryCounter()	N/A as no data parsed through this method	This function obtains PIN Try counter object from ICC.	Test successful
21	boolean checkVerificationCondition()	N/A as no data parsed through this method	This function returns ‘true’ if offline PIN verification can	Test successful

			be performed and ‘false’ otherwise.	
22	void applyPINverification()	N/A as no data parsed through this method	This function collects PIN from user and sends it clear to ICC.	Test successful
23	void updatePINstatus()	N/A as no data parsed through this method	This function receives result from ICC and update status to inform user.	Test successful

Personalised information

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
24	String nameOnCard()	N/A as no data parsed through this method	This function displays cardholder name.	Test successful
25	String cardType(String carddigit)	N/A as no data parsed through this method	This function displays card type.	Test successful
26	String sixteenDigit()	N/A as no data parsed through this method	This function displays sixteen card digit.	Test successful
27	DateFormat expireDate()	N/A as no data parsed	This function retrieves and	Test successful

		through this method	returns card expire date.	
28	DateFormat effectiveDate()	N/A as no data parsed through this method	This function retrieves and returns card effective date.	Test successful
29	String BIC()	N/A as no data parsed through this method	This function retrieves and returns BIC number.	Test successful
30	String IBAN()	N/A as no data parsed through this method	This function retrieves and returns IBAN number.	Test successful

Report generation

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
31	void informationSanitise(String data)	Normal:- data = “4967908712456543” Extreme:- N/A Erroneous:- N/A	Normal:- data = “4967****1245****” Extreme:- N/A Erroneous:- N/A	Test successful
32	void generateReport()	N/A as no data parsed through this method	This function generates a text report.	Test successful

9.2 White box testing

Each white box test case has three following subtests -

- Condition testing: this sub-test finds all IF statements within the program, and ensures the true/false values are correct.
- Loop testing: this sub-test locates all loops (FOR loop, WHILE loop, DO-WHILE loop) in the program, makes sure they run precisely the number of times they are designed to run, spotting infinite loops.
- Logic testing: this sub-test verifies that all methods and functions execute in a way developers intend.

Card input class

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
1	Method resetSmartcard	Normal:- System is asked to reset a NatWest MasterCard.	Normal:- resetSmartcard() returns sequence '3B 6E 00 00 00 31 C0 71 C6 65 01 B0 01 03 37 83 90 00'	Test successful
2	Method processATR	Normal:- System is asked to returns card vendor based on ATR sequence of NatWest MasterCard	Normal:- processATR() returns 'NatWest United Kingdom'	Test successful

Payment application class

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
3	Method locatePaymentApplication	Normal:- System is asked to locate payment application folder on a NatWest MasterCard Erroneous:- System is asked to locate payment application folder on a Vietcombank debit card	Normal:- locatePaymentApplication() returns 'true' Erroneous:- locatePaymentApplication() returns 'false'	Test successful
4	Method countNumberPaymentApplication	Normal:- System is asked to count how many payment application existing on a NatWest MasterCard	Normal:- countNumberPaymentApplication() return 1	Test successful
5	Method listAllPaymentApplication	Normal:- System is asked to return a list of all payment applications on a NatWest MasterCard	Normal:- listAllPaymentApplication() returns [Application name: SOLO, Application priority: 1]	Test successful

Transaction initialisation class

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
6	Method selectPaymentApplication	Normal:- System is asked to select a payment application on a HSBC VISA	Normal:- the FOR loop in selectPaymentApplication() goes through payment application list once, and application with highest priority is selected.	Test successful
7	Method initiateTransaction	Normal:- System is asked to initiate an EMV transaction	Normal:- initiateTransaction() returns ‘true’	Test successful

Cardholder verification class

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
8	Method cardholderVerificationPriority	Normal:- System is asked to return a list of cardholder verification methods of a NatWest MasterCard in priority order.	Normal:- functions returns 1.Plaintext offline PIN verification 2. Signature verification only 3.Enciphered online PIN verification 4. No cardholder verification needed	Test successful

Plaintext offline PIN verification

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
9	Method checkVerificationCondition	Normal:- System is asked if plaintext offline PIN verification can be performed on HSBC VISA card.	Normal:- checkVerificationCondition() returns ‘true’	Test successful
10	Method applyPINverification	Normal:- System is asked to apply PIN ‘1234’ with HSBC VISA card	Normal:- applyPINverification() apply PIN in the correct order. PIN Try counter object is obtained firstly, then all conditions must be satisfied before PIN will be sent to ICC.	Test successful

Personalised information

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
11	Method nameOnCard	Normal:- NatWest card owner ‘Khuong Nguyen’ puts his card in reader, and uses system to obtain information.	Normal:- nameOnCard() returns ‘Khuong Nguyen’	Test successful

12	String cardType(String carddigit)	Normal:- Sixteen card digits ‘6767456716013478’ is passed through cardType()	Normal:- cardType() returns ‘SOLO’	Test successful
13	Method sixteenDigit	Normal:- a NatWest MasterCard is inserted into reader and user wants to obtain sixteen digits on card.	Normal:- sixteenDigit() returns‘6767456716013478’	Test successful
14	DateFormat expireDate()	Normal:- a NatWest MasterCard with expiry date 11/10 is inserted into reader.	Normal:- expireDate() returns ‘30 / November / 2010’	Test successful
15	DateFormat effectiveDate()	Normal:- a NatWest MasterCard with effective date 11/07 is inserted into reader.	Normal:- effectiveDate() returns ‘01 / November / 2007’	Test successful

9.3 Integration testing

Integration testing uses bottom-up approach. The test cases are carried out as the software is examined by user. Each methods of each class are tested in the way data flows.

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
1	System initialises new session with smart card reader and processes inserted ICC preliminary information.	User inserts a NatWest EMV card into smart card reader, and presses 'Insert card' button.	Card input class runs createNewSession() method, resetSmartcard() method returns an ATR sequence and processATR() method processes this sequence.	Test successful. All functions are executed properly in their orders.
2	System initialises new session with smart card reader and processes inserted ICC preliminary information.	User inserts a Vietcombank card into smart card reader, and presses 'Insert card' button.	Card input class runs createNewSession() method, resetSmartcard() method returns an ATR sequence and processATR() method processes this sequence.	Test successful. All functions are executed properly in their orders.
3	System looks into ICC for payment application folder	User presses 'Locate payment application' button, with NatWest card inserted.	Payment application class executes locatePaymentApplication() method to verify payment application folder. Then, countNumberPaymentAppli	Test successful. All functions are executed properly in their orders.

			cation() method is run to count how many payment applications inside.	
4	System looks into ICC for payment application folder	User presses ‘Locate payment application’ button, with Vietcombank card inserted.	Payment application class executes locatePaymentApplication() method to verify payment application folder. Then, countNumberPaymentAppli cation() method is run to count how many payment applications inside.	Test failure. System informs no payment application has been found. This is because Vietcombank card is not EMV card.
5	System looks into payment application folder and lists all payment applications inside with their information	User presses ‘List all’ button	Payment application class executes listAllPaymentApplication() method and a list of all payment applications is returned. At the same time, their information are shown on the interface.	Test successful. All functions are executed properly in their orders.
6	System creates a new EMV transaction	User presses ‘Initiate transaction’ button	Transaction initialisation class receives the list of payment applications from Payment application class. Then, selectPaymentApplication()	Test successful. All functions are executed properly in their orders.

			method is executed to select the payment application with highest priority and initiateTransaction() method create the new transaction.	
7	System updates the authentication status of three card authentication methods	User presses ‘Analyse’ button under Card Authentication tab	Card authentication class runs authenticationStatus() method and updates SDA status to supported and DDA, cDDA statuses to NOT supported.	Test successful. All functions are executed properly in their orders.
8	System shows all data elements belonging to static offline data authentication	User presses ‘Static Data Authentication’ button	Card authentication class runs SDADataElement() method and retrieves all possible data elements	Test successful. All functions are executed properly in their orders.
9	System shows all data elements belonging to dynamic offline data authentication	User presses ‘Dynamic Data Authentication’ button	Card authentication class runs DDADataElement() method and retrieves all possible data elements	Test successful. All functions are executed properly in their orders.
10	System shows all data elements belonging to combined dynamic data authentication/application cryptogram generation.	User presses ‘Combined DDA/Cryptogram’ button	Card authentication class runs cDDADataElement() method and retrieves all possible data elements	Test successful. All functions are executed properly in their orders.
11	System updates the verification status of eight	User presses ‘Analyse’ button under Cardholder Verification	Cardholder verification class runs	Test successful. All functions are executed

	cardholder verification methods	category	cardholderVerificationStatus() method and updates eight methods' status.	properly in their orders.
12	System launches a new interface, which simulates an ATM keypad for user to perform offline PIN verification	User presses 'PIN verification' button	Plaintext offline PIN verification class is called. A separated interface is created to perform offline PIN verification	Test successful. Plaintext offline PIN verification interface is displayed.
13	System collects preliminary information from ICC to begin offline PIN verification process	User presses 'Initialise' button under Offline PIN verification category	Plaintext offline PIN verification class runs getPINTryCounter() method to obtain PIN Try counter object, and checks the verification condition before allowing user to enter PIN.	Test successful. All functions are executed properly in their orders.
14	System sends user PIN clear to ICC for verification	User presses 'ENTER' button under Offline PIN verification category	Plaintext offline PIN verification class runs applyPINVerification() method to send the PIN clear to ICC.	Test successful. All functions are executed properly in their orders.
15	System erased wrongly entered PIN and allows user to re-enter	User presses 'CLEAR' button under Offline PIN verification category	PIN is clear on screen.	Test successful. PIN is clear from screen.
16	System terminates current offline PIN verification process.	User presses 'CANCEL' button under Offline PIN verification category	Plaintext offline PIN verification class is exited.	Test successful. Plaintext offline PIN verification class is exited.

	System displays all personalised information found in ICC	User presses 'Get personalised data' button	Personalised information class executes all functions to query personalised information from ICC.	Test successful. All personalised information are displayed on screen.
17	System auto-generates a user-friendly report, using collected information during the survey	User presses 'Generate report' button under File menu	Report generation class gets information obtained throughout the survey process and creates a text report.	Test successful. A text report is generated properly.

9.4 System testing

System testing verifies the completed system against the requirements and functionality to ensure it matches the specification. This type of testing is similar to the approach of black box testing. Internal structure of the program is not considered.

TEST ID	DESCRIPTION	TEST DATA	EXPECTED OUTCOME	ACTUAL OUTCOME
1	EMV card is inserted and card vendor is traced back	User inserts an HSBC debit card into smart card reader, and presses 'Insert card' button.	Card vendor is returned as 'HSBC United Kingdom'	Test successful
2	EMV card is inserted and card vendor is traced back	User inserts a Vietcombank card into smart card reader, and presses 'Insert card' button.	Card vendor is returned as 'Vietcombank Vietnam'	Test failure. System does not display any vendor information.

3	No process is performed when there is no card resided in smart card reader	User does not insert any card in smart card reader, and presses 'Insert card' button to obtain preliminary information.	Card status does not change from 'Waiting for card'. Nothing happens.	Test successful
4	A session must be started before any other functionalities can be performed	'Analyse' button, or 'Locate payment application' button, or 'Initiate transaction' button, or 'Get personalised data' button is pressed before 'Insert card' button	System refuses to perform the intended function, since no session between ICC and reader has been established yet.	Test failure. System crashes and a CardIO exception is thrown.
5	An EMV transaction must be created before card authentication, or cardholder verification, or PIN verification can be performed	'Analyse' button, or 'PIN verification' button, or 'Get personalised data' button is pressed before 'Initiate transaction' button	System refuses to perform the intended function, since no EMV transaction was created. An attempt to get information from ICC will be refused by ICC.	Test failure. System does not crash but no information is returned.
6	Card is cold-ejected by user in the middle of procedure	User inserts card into reader, and performs all steps normally until a transaction is initialised. Then, user hard-ejected the card without notifying the system. Then	System recognises the missing of ICC in the middle of the process and terminates the current session.	Test failure. System's behaviours are unexpected and depend on which combination of buttons user press.

		user continues to operate the system by analysing card authentication information.		
7	User inputs wrong PIN number two continuous times	The correct PIN of card is ‘1234’. User enters PIN ‘1111’, then PIN ‘2222’.	System informs in red colour at status bar that ‘There is only one time left to input PIN’.	Test successful
8	User inputs wrong PIN number three continuous times	The correct PIN of card is ‘1234’. User enters PIN ‘1111’, then PIN ‘2222’, then PIN ‘3333’.	System informs in red colour at status bar that ‘No more PIN tries is allowed’ and system does not send PIN verification instruction to ICC any more, even user asks to.	Test successful
9	Card has been blocked, and user still tries to enter PIN	At the moment, card does not allow any PIN tries (PIN Try counter number is zero). User still enters PIN ‘1234’.	System does not send PIN to ICC and informs user no more PIN tries is allowed.	Test successful
10	Transaction is forced to finish early	User presses ‘CANCEL’ button in the middle of transaction.	System terminates current transaction and ejects the card	Test successful

11	User just inputs 3-digit PIN and wants to starts over again	User presses ‘CLEAR’ button.	System erases screen and allows user to re-enter new PIN	Test successful
12	User inputs too short or too long PIN	User inputs PIN ‘12’ or PIN ‘0123456789000’ and press ‘ENTER’ button	System does not accept PIN with less than 4 digits or more than 12 digits. User is prompted to re-try.	Test successful

9.5 System testing failures analysis

This part explains four test failures found in system testing. Each failed test case will be analysed and fixed -

- Test case 2: The reason system does not display any information about card vendor as smartcard_list text file does not cover Vietcombank card. This issue can be fixed by adding Vietcombank ATR to exception list to be handed separately by the program.

- Test case 4: System crashes since no session between ICC and software terminal is established. From a programmer perspective, this problem can be fixed by adding a boolean flag to 'Insert card' button. When this button is pressed, flag value is updated to 'true'. And before performing any analysis operations, software system checks the flag value first. However, this will slow down the system a little since it always has to check the flag value.

- Test case 5: System crashes since no EMV transaction has been initialised. This problem is similar to test case 4, and can be fixed in the same manner by adding another boolean flag to 'Initiate transaction' button.

- Test case 6: System can either crash or returns meaningless information, depending on which combination of buttons user presses after cold-ejecting the card. This issue is a bit tough to fix from the operator's perspective since there is no flag indictor in either terminal or ICC to indicate the current state of ICC. However, since the software system does not know when user will suddenly eject the card, therefore it requires an indicator boolean flag for every single operation of the project. This is obviously not reasonable, since it will massively slow down the whole system. Thus for the time being, only important operations such as card authentication analysis request, and cardholder verification analysis request will have an indicator flag.

10. Experimental results

This chapter shows text results obtained by using 'Generate report' function of the software on four different UK Chip and PIN cards manufactured by NatWest, HSBC, Barclays, Abbey, and two international cards from Vietnam and Thailand.

10.1 NatWest card survey

```
*** EMV card survey report 1

Card is reset, the ATR value returned is 3B 6E 00 00 00 31 C0 71 C6 65 01 B0 01 03 37 83
90 00

Card manufacturer is NatWest United Kingdom


* * * * * * * * * * * * * * * * * * * * * *

*    Card payment applications survey *

* * * * * * * * * * * * * * * * * * * * * *

Payment application folder exists in this card

There is only one payment application on this card

The list of all payment applications:

- Application label is SOLO
- Application priority is 01


* * * * * * * * * * * * * * * * * * * * * * *

*   Card authentication methods survey *

* * * * * * * * * * * * * * * * * * * * * * *

Card authentication methods checking:

- Static data authentication is supported
- Dynamic data authentication is NOT supported
- Combined dynamic data authentication and application cryptogram generation is NOT
supported

The data element for static data authentication found on this card:

- Certification Authority Public Key Index is 800
```

```
70 81 C0 8F 01 04 9F 32 01 03 92 24 A4 3C EF 04 93 DB 55 39 4A 49 27 94 5B 5D 0E 46 9E
50 29 37 80 D1 C8 19 ED BC A3 8B 8C DB 09 F3 04 91 61 45 90 81 90 9B 66 2E 7B 1D DC
C3 B1 14 17 35 31 2A 01 75 D9 38 AA 8C 84 53 38 3D 0F F7 9A 76 82 5D 0E EC 31 50 58 7D
7C 5D 78 8B 60 72 6B B4 13 AE E4 1D 14 A9 B0 A6 3F 0E 17 D8 B0 49 A1 A5 6F BB 2B F8
1C 37 80 78 39 68 05 A9 37 56 C2 FE A4 58 2A 70 0A 89 C4 1C 76 E8 D9 6F 06 57 0C A6 5F
FD E3 99 A9 96 11 03 1C 94 69 8B ED A8 10 CF 00 1D 9A 8D FE 87 1A 80 89 D9 AC 2C 20
CD 20 B9 E3 FA 0B 9A 53 D4 48 33 36 DC 62 48 DD B7 A6 C3 64 E2 17 9A 70 23 5F 20 0B
4D 52 20 4B
```

- Issuer Public Key Certificate is
90 9B 66 2E 7B 1D DC C3 B1 14 17 35 31 2A 01 75 D9 38 AA 8C 84 53 38 3D 0F F7 9A 76
82 5D 0E EC 31 50 58 7D 7C 5D 78 8B 60 72 6B B4 13 AE E4 1D 14 A9 B0 A6 3F 0E 17 D8
B0 49 A1 A5 6F BB 2B F8 1C 37 80 78 39 68 05 A9 37 56 C2 FE A4 58 2A 70 0A 89 C4 1C 1C
76 E8 D9 6F 06 57 0C A6 5F FD E3 99 A9 96 11 03 1C 94 69 8B ED A8 10 CF 00 1D 9A 8D FE
87 1A 80 89 D9 AC 2C 20 CD 20 B9 E3 FA 0B 9A

- Issuer Public Key Remainder is
A4 3C EF 04 93 DB 55 39 4A 49 27 94 5B 5D 0E 46 9E 50 29 37 80 D1 C8 19 ED BC A3 8B
8C DB 09 F3 04 91 61 45

- Issuer Public Key Exponent is 03

* *

* Cardholder verification methods survey *

* *

Verification methods checking:

- Plaintext offline verification Plaintext is supported. This method is performed if terminal supports this method
- Enciphered online PIN verification is supported. This method is performed if terminal supports this method
- Plaintext offline PIN and signature is NOT supported.
- Enciphered offline PIN verification is NOT supported.
- Enciphered offline PIN and signature is NOT supported.
- Signature verification only is supported. This method is performed if terminal supports this method
- No cardholder verification needed is supported. This method is performed if terminal supports this method

The priority order of CVM in this card is:

1. Plaintext offline PIN verification

2. Signature verification only
3. Enciphered online PIN verification
4. No cardholder verification needed

* *

*   Personalised information survey   *

* *

Cardholder name is MR K NGUYEN
Card type is SOLO
Card sixteen digits are 6767****1601****
This card is effective from 01 / November / 2007
This card will expire on 30 / November / 2010

10.2 HSBC card survey

*** EMV card survey report 2

Card is reset, the ATR value returned is 3B 6E 00 00 00 31 C0 71 D6 65 94 E8 03 40 00 83 90 00

Card manufacturer is HSBC United Kingdom

* *

*   Card payment applications survey *

* *

Payment application folder exists in this card

There is only one payment application on this card

The list of all payment applications:

- Application label is SOLO
- Application priority is 01

* *

*   Card authentication methods survey *

* *

Card authentication methods checking:

- Static data authentication is supported
- Dynamic data authentication is NOT supported
- Combined dynamic data authentication and application cryptogram generation is NOT supported

The data element for static data authentication found on this card:

- Certification Authority Public Key Index is C5 F2 01

-   Issuer Public Key Certificate is
90 A1 CE B7 C3 09 54 6A 4F AD B2 CA 61 3E 94 62 61 83 68 9A 68 0E B3 7C E8 68 DF BC 78
59 CC F3 B3 1F E9 F2 CE 51 D9 C4 C1 AD 57 30 8B 30 B8 8B 12 4E 73 2F 96 07 DA EE 30 1D
6D 19 B6 53 85 22 44 78 2C 93 68 DA 5F 66 C8 95 F7 AB 2F 1B D0 72 EE 0B 34 30 EF DC 1C
09 4C 6B 06 F7 B0 A7 63 AD 62 BC 05 DC 67 A5 4B 4F 35 73 65 BE EA 33 B8 5F 4D F6 EB
2A 53 10 5B 98 7A 56 64 C6 80 E2 93 41 A7

* *

* Cardholder verification methods survey *

* *

Verification methods checking:

- Plaintext offline verification Plaintext is supported. This method is performed if terminal supports this method
- Enciphered online PIN verification is supported. This method is performed if terminal supports this method
- Plaintext offline PIN and signature is NOT supported.
- Enciphered offline PIN verification is NOT supported.
- Enciphered offline PIN and signature is NOT supported.
- Signature verification only is supported. This method is performed if terminal supports this method
- No cardholder verification needed is supported. This method is performed if terminal supports this method

The priority order of CVM in this card is:

1. Plaintext offline PIN verification
2. Signature verification only
3. Enciphered online PIN verification
4. No cardholder verification needed

```
* * * * * * * * * * * * * * * * * * * * * * *

*      Personalised information survey      *

* * * * * * * * * * * * * * * * * * * * * * *

Cardholder name is SRICHAIYONGPANICH/N.MR
Card type is SOLO
Card sixteen digits are 6767****0232****
This card is effective from 11 / October / 2008
This card will expire on 31 / March / 2010
```

10.3 Abbey (Santander) card survey

```
*** EMV card survey report 3

Card is reset, the ATR value returned is 3B 6E 00 00 00 31 C0 65 54 B6 01 00 84 71 D6 8C
61 31

Card manufacturer is Abbey United Kingdom


* * * * * * * * * * * * * * * * * * * * * *

*    Card payment applications survey     *

* * * * * * * * * * * * * * * * * * * * * *

Payment application folder exists in this card

There is only one payment application on this card

The list of all payment applications:

- Application label is MASTERCARD
- Application priority is 01


* * * * * * * * * * * * * * * * * * * * *

*   Card authentication methods survey    *

* * * * * * * * * * * * * * * * * * * * *


Card authentication methods checking:
```

- Static data authentication is supported
- Dynamic data authentication is NOT supported
- Combined dynamic data authentication and application cryptogram generation is NOT supported

The data element for static data authentication found on this card:

- Issuer Public Key Certificate is
91 F9 F1 F0 D3 13 53 73 93 23 03 03 93 03 13 83 93 17 04 B5 F2 50 30

* *

* Cardholder verification methods survey *

* *

Verification methods checking:

- Plaintext offline verification Plaintext is supported. This method is performed if terminal supports this method
- Enciphered online PIN verification is supported. This method is performed if terminal supports this method
- Plaintext offline PIN and signature is NOT supported.
- Enciphered offline PIN verification is NOT supported.
- Enciphered offline PIN and signature is NOT supported.
- Signature verification only is supported. This method is performed if terminal supports this method
- No cardholder verification needed is supported. This method is performed if terminal supports this method

The priority order of CVM in this card is:

1. Enciphered online PIN verification
2. Plaintext offline PIN verification
3. Signature verification only
4. No cardholder verification needed

* *

* Personalised information survey *

* *

Cardholder name is MAMMERI/AMINA

Card type is MasterCard
Card sixteen digits are 5454****7818****
This card is effective from 01 / February / 2007
This card will expire on 28 / February / 2010

10.4 Barclays card survey

*** EMV card survey report 4

Card is reset, the ATR value returned is 3B 6E 00 00 00 31 C0 71 C6 65 01 B0 01 03 37 83 90 00

Card manufacturer is Barclays United Kingdom

* *

* Card payment applications survey *

* *

Payment application folder exists in this card

There is only one payment application on this card

The list of all payment applications:

- Application label is VISA DEBIT
- Application priority is 01

* *

* Card authentication methods survey *

* *

Card authentication methods checking:

- Static data authentication is supported
- Dynamic data authentication is NOT supported
- Combined dynamic data authentication and application cryptogram generation is NOT supported

The data element for static data authentication found on this card:

- Issuer Public Key Certificate is 70 F9 F1 F0 D3 13 23 53 33 43 03 03 93 03 13 13 73

* *

* Cardholder verification methods survey *

* *

Verification methods checking:

- Plaintext offline verification Plaintext is supported. This method is performed if terminal supports this method
- Enciphered online PIN verification is supported. This method is performed if terminal supports this method
- Plaintext offline PIN and signature is NOT supported.
- Enciphered offline PIN verification is NOT supported.
- Enciphered offline PIN and signature is NOT supported.
- Signature verification only is supported. This method is performed if terminal supports this method
- No cardholder verification needed is supported. This method is performed if terminal supports this method

The priority order of CVM in this card is:

1. Enciphered online PIN verification
2. Plaintext offline PIN verification
3. Signature verification only
4. Enciphered online PIN verification
5. No cardholder verification needed

* *

*   Personalised information survey   *

* *

Cardholder name is TYE/LINDA
Card type is VISA DEBIT
Card sixteen digits are 4659****1281****
This card is effective from 31 / December / 2008
This card will expire on 01 / April / 2012

10.5 Thai Bank of Ayudhya card survey

*** EMV card survey report 5

Card is reset, the ATR value returned is 3B 6E 00 00 00 31 C0 71 86 65 01 78 01 27 34 83 90 00

Cannot trace back card manufacturer

```
* * * * * * * * * * * * * * * * * * * * * * *

*    Card payment applications survey     *

* * * * * * * * * * * * * * * * * * * * * * *
```

Payment application folder exists in this card

There is only one payment application on this card

The list of all payment applications:

- Application label is VISA CREDIT
- Application priority is 01

```
* * * * * * * * * * * * * * * * * * * * * *

*   Card authentication methods survey    *

* * * * * * * * * * * * * * * * * * * * * *
```

Card authentication methods checking:

- Static data authentication is supported
- Dynamic data authentication is NOT supported
- Combined dynamic data authentication and application cryptogram generation is NOT supported

The data element for static data authentication found on this card:

- Certification Authority Public Key Index is 07

- Issuer Public Key Certificate is
7D 11 08 20 10 00 00 59 28 76 54 F7 04 C5 F2 50 30 60 80 15 F2 40 31 10 82 15 A0 84 55
20 51 00 00 90 72 75 F3 40 10 19 2F 07 02 FF 00 8E 0E 00 00 00 00 00 00 00 00 1E 03 02
03 1F 00 9F 0D 05 F0 58 8C 88 00 9F 0E 05 00 00 00 00 00 9F 0F 05 F0 78 8C 98 00 5F 28
02 07 64 70 81 96 8F 01 07 90 81 90 7E 4E FC 2E 2F 7F 15 CD F9 50 15 4E FD 4E F9

- Issuer Public Key Remainder is
65 4F 70 4C 5F 25 03 06 08 01 5F 24 03 11 08 21 5A 08 45 52 05 10 00 09 07 27 5F 34 01
01 9F 07 02 FF 00 8E 0E 00 00 00 00 00 00 00 00 1E 03 02 03 1F 00 9F 0D 05 F0 58 8C 88
00 9F 0E 05 00 00 00 00 00 9F 0F 05 F0 78 8C 98 00 5F 28 02 07 64 70 81 96 8F 01 07 90
81 90 7E 4E FC 2E 2F 7F 15 CD F9 50 15 4E FD 4E F3 CC 2F 68 23 61 61 FC 22 93 A8 BD 45
77 F0 98 95 A0 B8 9F 46 F7 3B C4 18 E8 5D 97 BF D2 A8

* *

* Cardholder verification methods survey *

* *

Verification methods checking:

- Plaintext offline verification Plaintext is NOT supported.
- Enciphered online PIN verification is supported. This method is performed if terminal supports this method
- Plaintext offline PIN and signature is NOT supported.
- Enciphered offline PIN verification is NOT supported.
- Enciphered offline PIN and signature is NOT supported.
- Signature verification only is supported. This method is performed if terminal supports this method
- No cardholder verification needed is supported. This method will always be performed

The priority order of CVM in this card is:

1. Signature verification only
2. Enciphered online PIN verification
3. No cardholder verification needed

* *

*   Personalised information survey   *

* *

Cardholder name is MR.N/SRICHAIYONG
Card type is VISA
Card sixteen digits are 4559****0009****
This card is effective from 01 / August / 2007
This card will expire on 21 / August / 2012

10.6 Vietnam Vietcombank card survey

*** EMV card survey report 6

Card is reset, the ATR value returned is 3B 6D 00 00 80 31 80 65 B0 84 01 00 C8 83 00 90 00

Cannot trace back card manufacturer

```
* * * * * * * * * * * * * * * * * * * * * *
*    Card payment applications survey *
* * * * * * * * * * * * * * * * * * * * * *
There is no payment application folder existing on this card
There is no payment application on this card


* * * * * * * * * * * * * * * * * * * * * * *
*   Card authentication methods survey *
* * * * * * * * * * * * * * * * * * * * * * *
Cannot perform card authentication


* * * * * * * * * * * * * * * * * * * * * * * *
* Cardholder verification methods survey *
* * * * * * * * * * * * * * * * * * * * * * * *
Cannot perform cardholder verification


* * * * * * * * * * * * * * * * * * * * * * * *
*       Personalised information survey      *
* * * * * * * * * * * * * * * * * * * * * * * *
Personalised information cannot be retrieved
```

10.7 Survey analysis

This section provides an overall comment about the results obtained above. Based on the features of each card in real life, the correctness of the system can be verified.

Firstly, it is interesting to see all UK EMV cards only have one payment application at the moment, although the card can accommodate many payment applications. Interestingly, the payment application label is the card type itself. For example, a Santander MasterCard has payment application label names MASTERCARD.

Secondly, it can be concluded from all four UK Chip and PIN reports that all UK cards do support both card authentication and cardholder verification. However, for card authentication, at the moment only static data authentication is supported. None of UK bank cards support dynamic data authentication. For cardholder verification, none of UK bank cards supports enciphered offline PIN verification. However, all UK cards support plaintext offline PIN verification, enciphered online PIN verification and signature verification. It can be concluded that UK bank cards do provide both online and offline facilities for verification procedures.

Of the two international cards, there are quite many differences in the report of cardholder verification methods. For the Thai bank card, the report shows that it only support enciphered online PIN verification and signature verification. No PIN verification is supported at all as the report indicated. Since Thailand does not implement PIN facility yet, and the surveyed Thai card is a signature one, the report information is hence correct. Finally, the Vietcombank report does not show any information apart from ATR sequence. This is understandable since Vietcombank card is not an EMV card. An attempt to locate payment application was not successful, hence no survey could be obtained.

11. Conclusion and future work

This chapter summaries the project and proposes possible work to implement in the future. Initially, the chapter gives the project writer a chance to self-evaluate the project development. Next, all difficulties encountered at each project development stage, as well as experience gained after completing this project are discussed in subsequent sections. The most important aspect after the project is finished is what the project writer has critically learned by doing this project. This is discussed in the next section. Finally, the chapter concludes by a list of plans for future investigation.

11.1 Project self-evaluation

This section gives a self-review of the project under the project writer’s perspective when it is concluded. Firstly, it is worth reminding electronic payment has dramatically changed the way humans spend money in public places. Instead of carrying a wallet full of bank notes and coins, nowadays people only have to carry a piece of plastic which contains all the needed information to spend in stores. Not only does electronic payment increase the speed of transactions since shop sellers do not have to give small changes to customers, but it also increases the security of transactions since all money is transferred directly behind the scene from bank to bank. However, a loss of bank card can result in a loss of all money stored in an account. Thus the need of a more secure protection mechanism for those plastic cards is increasingly demand. EMV has provided a great standard for secure electronic payment system. However, as the trend of business, a demand for better protection in the future can be foreseen. Late 2009s saw the first introduction of contactless EMV which even speeds up the transaction by terminating the need to insert bank card into readers. Information can be transported in the air, thus providing ease of access, yet at the same time demanding new security protection. This project was executed to prepare the project writer with sufficient knowledge to do further research in electronic payment area.

Firstly, because of a feature of an information security project, three crucial security objectives are always guaranteed for the project through-out project development. They are -

- Confidentiality: a guarantee that sensitive information on EMV Card including personal detailed information does not get into wrong hands.

  Confidentiality can be exploited by reading the card report generated by the software program, and use of information such as cardholder name, sixteen digits, expiry date, and effective date for online payment. However, the software makes sure that this sensitive information is hidden by asterisk mark, and will not be reversed easily.
- Integrity: a guarantee that all data retrieved from Card are manipulated and stored correctly. Software system will prevent the card from being damaged in critical situation such as user deliberately enters three wrong PIN to block the card, in such case.
- Availability: a guarantee that software system should always work and be able to access information from Card when needed.

Secondly, the project has demonstrated fully and correctly all six project objectives mentioned in chapter 3.3. Upon completion of the project, a majority of EMV knowledge has been fully understood by the project writer. These include how card authentication and cardholder verification work, as well as how to implement offline PIN verification. These details are basis for any electronic payment systems, furthermore they provide a good foundation to implement an actual merchant terminal in the future. Finally, the project writer has a good experience on experimenting with real life bank cards and sees how the banks organise their information. The list of possible future work is scheduled to make sure of the possibility of project expansion in the future.

11.2 Project difficulties and solutions

This section explained what difficulties have been encountered when this project was developing at each development stage. Each stage is followed by a solution approach. For a full description of each development stage, refer to chapter 3.4.

11.2.1 Requirements analysis stage

- Problem

The hardest challenge at this early stage is to decide what programming language is to be used to write software system. The decision must take into the account the existing hardware, and the ease of access to the system.

- Solution

C++ and Java were the two possibilities since they are both taught in undergraduate course. Java was chosen because of the vast majority of supported library and the ease of access to PC/SC framework and Smart card resource manager provided under Windows operating system.

11.2.2 Research stage

- Problem

The major development issue with EMV was the lack of available resources. At the time this project was concluded (12-March-2010), there were only three EMV-related books on sale on Amazon UK. Two of them (O’Mahony and Haddad’s) contained very little useful information about EMV for this project

- O'Mahony, D., et al, *Electronic Payment Systems for E-commerce*, Artech House, 2001.
- Haddad, A., A New Way to Pay: Creating Competitive Advantage Through the EMV Smart Card Standard, Gower Publishing, 2005.
- Radu, C., *Implementing Electronic Card Payment Systems*, Artech House, 2002.

- Solution

Most theories conducted in this project are collected directly from four EMV specification books, ISO 7816-4 standard, MasterCard documentation, along with various online articles and blogs. However, the information is sparse and not united, as they come from many different online sources, thus making it harder for this development stage.

11.2.3 Modelling stage

- Problem

The hardest obstacle in this stage is to decide what functionalities to include in the system, and how to design them.

- Solution

Class Responsibility Collaborator cards (CRC cards) produced handwritten from scratch provides a useful approach to design all functionalities and the relationship amongst them.

11.2.4 Coding stage

- Problem

The biggest challenge at this stage was to make the smart card reader and smart card to communicate with each other, and write the first 'hello word' program.

- Solution

Many sources were used to aid this early stage, after two entities can communicate with each other, the rest of the procedure can be built on.

11.2.5 Testing stage

- Problem

It was difficult to collect a variety of UK Chip and PIN cards to perform testing and it is understandable that most testing failures relate to the fact different banks organise their cards differently. A list of all test cases is documented in chapter 9.

- Solution

Minor tweaks are performed for each bank. An example of such tweak: All early HSBC cards which expire early in 2010 are recognised as 'First Direct UK VISA Debit' based on ATR smart card list, while all new HSBC cards issued after 2009 are properly recognised as 'HSBC UK Debit Card'. This issue can be fixed by adding this ATR string to an exception list.

11.2.6 Documenting stage

- Problem

The hardest obstacle to produce a good formal report is how to organise the ideas in a fluent readable way, and how to make the report look formal.

- Solution

As a diary log is maintained since the first day the project was begun, it helps to keep track of where the ideas come from and organise them in a scientific way. Word style, theme and layout erroneous are fixed upon suggestions by project supervisor.

11.3 Experience gained

A variety of experiences and skills have been learned by project designer upon completing the project. Initially, in order to propose with the EMV topic, project writer must do some research, not only because the topic is utterly new for undergraduate student, but the Chip and PIN aspect are not new in the United Kingdom. This process provides project proposers skills to perform independent research under guidelines from project supervisor. Furthermore, as the nature of a theoretical analysis project, many information under hexadecimal byte and bit from need to be analysed, thus preparing project writer with some information analysis skill, which is not only useful for EMV analysis, but also helpful for other analysis areas too. Secondly, since one of the achievements of the project is the practical software, which independently demands software development skills to be used, hence the waterfall model has been studied and applied to guarantee the best approach for software development. And, as the project utilises some hardware equipment, which makes it stand out from a normal pure software project. Software development process must take into account the existing of hardware features and make sure they communicate nicely with each other as a united system. Thirdly, what makes the project interesting is the chance to use the system with real life EMV Chip and PIN cards in the United Kingdom. This investigation process gives project designer a specific in-depth experience at how EMV is implemented in real life. Finally, in order to produce this project report, a numerous skill of Microsoft Word and report writing skill has been learned, thus in the end, a fluent and formal report can be produced.

11.4 Future work

This section outlined some unfinished work which could not be achieved in time because of time constraints and limited available resources.

- A VISA or MasterCard public key would help to perform certificate verification, however, these public keys are only issued to banks, and are almost kept secret from individuals.
- Further study into EMV contactless cards as they are increasing in industry demand. Barclays and Abbey have recently issued EMV contactless cards for their customers. However, to implement contactless feature, a contactless reader is required.
- Further work to create a full secure working terminal, which operates as secure as real merchant terminal in stores. This operation requires a further in-depth look into terminal risk management function performed by terminal.

Therefore, all in this project has shown the profound nature of EMV cards and their structure on the retail environment and indicated areas where future work would benefit all parties for the common good - retailers, consumers and financiers.

Appendix A: Software manual

This section provides a user-friendly manual for the users who are unfamiliar with technical details to execute this system.

There are four basic steps to run this project

- **Step 1**: Hardware preparation

  Connect PC/SC SmartCard Device to Personal Computer via USB or Serial Port. For all Windows environments, Device will be set up automatically by 'SmartCard Resource Manager' included in the Operating System. No driver installations are needed. A small Light Detector on Device will flash to signal Device is ready to use

  Insert EMV SmartCard into Reader Device slot, notice the contact surface between ICC and Reader

- **Step 2**: Software preparation

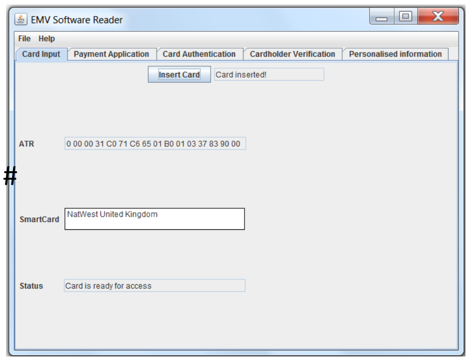
  Install Java Runtime Environment: http://java.sun.com/

  Install JACCAL: http://jaccal.sourceforge.net/

- **Step 3**: Software executing

  Execute EMVSmartCardReader Java class. A graphic user interface will show up with all instructions.

  For Card Input tab, presses the 'Insert Card' button to begin a new working session. The status bar should reads 'Card is ready for access'



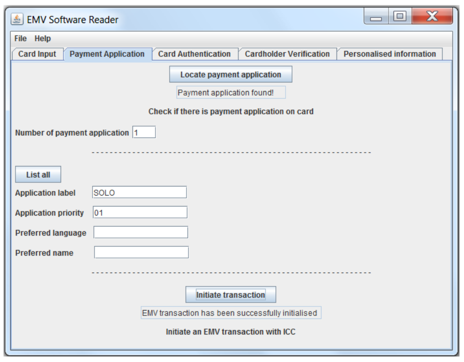
Card inserted!

ATR 0 00 00 31 C0 71 C6 65 01 B0 01 03 37 83 90 00

#

SmartCard NatWest United Kingdom

Status Card is ready for access



Payment application found!

Check if there is payment application on card

Number of payment application 1

Application label SOLO

Application priority 01

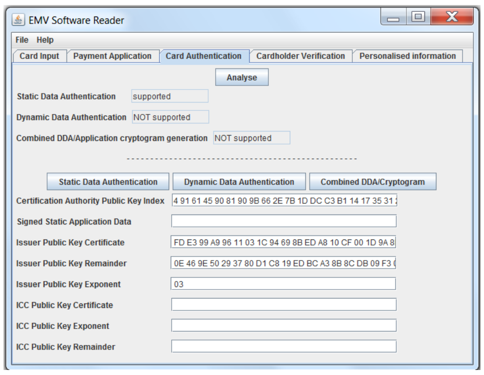
Preferred language

Preferred name

EMV transaction has been successfully initialised

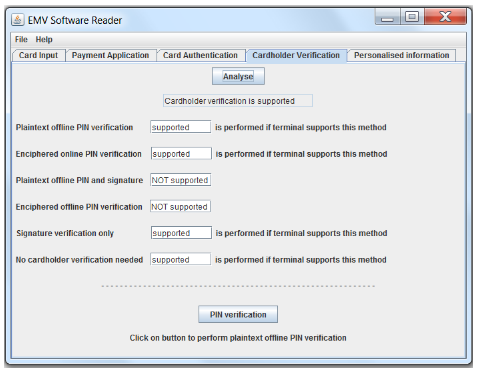
Initiate an EMV transaction with ICC

For Payment Application tab, the user can check if there is a payment application on card by pressing the 'Locate payment application' button. Furthermore, by pressing 'List all' button, all relevant information about payment applications are listed. Finally, the 'Initiate transaction' button will create an EMV transaction between software reader and ICC. The status will be updated accordingly.



Static Data Authentication: supported
Dynamic Data Authentication: NOT supported
Combined DDA/Application cryptogram generation: NOT supported

Certification Authority Public Key Index: 4 91 61 45 90 81 90 9B 66 2E 7B 1D DC C3 B1 14 17 35 31
Signed Static Application Data:
Issuer Public Key Certificate: FD E3 99 A9 96 11 03 1C 94 69 8B ED A8 10 CF 00 1D 9A 8
Issuer Public Key Remainder: 0E 46 9E 50 29 37 80 D1 C8 19 ED BC A3 8B 8C DB 09 F3
Issuer Public Key Exponent: 03
ICC Public Key Certificate:
ICC Public Key Exponent:
ICC Public Key Remainder:

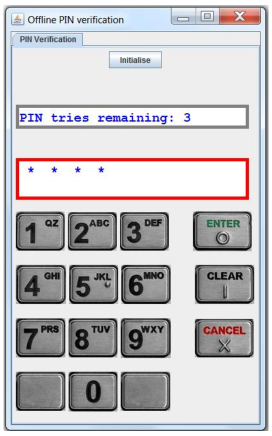


Cardholder verification is supported

Plaintext offline PIN verification: supported is performed if terminal supports this method
Enciphered online PIN verification: supported is performed if terminal supports this method
Plaintext offline PIN and signature: NOT supported
Enciphered offline PIN verification: NOT supported
Signature verification only: supported is performed if terminal supports this method
No cardholder verification needed: supported is performed if terminal supports this method

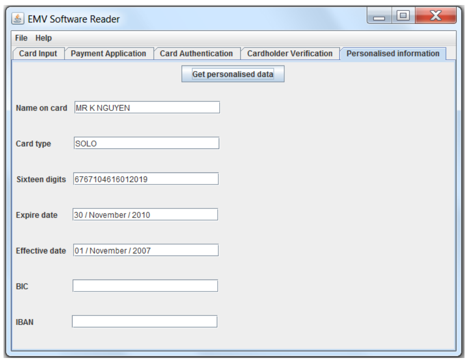
Click on button to perform plaintext offline PIN verification

The next two tabs – Card authentication and Cardholder verification allows the user to survey information of the two protection mechanisms. By pressing the 'Analyse' button, the statuses of those methods will change to either 'supported' or 'NOT supported'. Under Cardholder verification tab, there is another 'PIN verification' button which launches a separate window to perform real offline PIN verification procedure



PIN tries remaining: 3

* * * *



Name on card: MR K NGUYEN
Card type: SOLO
Sixteen digits: 6767104616012019
Expire date: 30 / November / 2010
Effective date: 01 / November / 2007
BIC:
IBAN:

The user can enter PIN with both provided PIN pad and keyboard. The software was programmed to capture inputs from both devices. If the user mistypes his PIN, it can be reset by pressing the 'CLEAR' button. For security purposes, entered PIN will be displayed in asterisk mark, although the real PIN is saved by the software. By pressing the 'ENTER' button, the programme will performed PIN verification and result is informed to the user in the status bar. Finally, by pressing the 'CANCEL' button, current transaction will be finished.

The final tab – Personalised information provides information which was hard-coded during personalised stage of card manufacturing. And finally, the user can choose to export all surveyed information onto a text file with 'Report generation' option provided under File menu.

- **Step 4**: Software closing
  Each new smart card requires a new session to be created. However, it is not possible for the software itself to detect the existence or disappearance of smart card in reader device. User must manually tell software that a new inserted ICC needs to be recognised.

Appendix B: Full source code listing

EMVSoftwareReader.java

```
import com.jaccal.*;
import com.jaccal.command.*;
import com.jaccal.util.*;

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class EMVSoftwareReader extends JFrame implements ActionListener
{
        JMenuItem item2 = new JMenuItem("About");
        JMenuItem item4 = new JMenuItem("Save report");

        static String AID = "";
        static SessionFactory f;
        static Session [] se;
        static ApduCmd capdu;
        static CardResponse rapdu;

        // main EMV reader will set those values, then GUI will read them
        and display
        static String atrValue = "";
        static String cardManufacturer = "";
        static String cardLanguage = "";
        static String cardType = ""; // application label
        static String accountNumber = "";
        static String nameOnCard = "";
        static String sixteenDigit = "";
        static String expireYear = "";
        static String expireMonth = "";
        static String expireDate = "";
        static String beginYear = "";
        static String beginMonth = "";
        static String beginDate = "";
        static String AFL = "";
        static String BIC = "";
        static String IBAN = "";


        /// SDA data element
        static String CertificationAuthorityPublicKeyIndex = "";
        static String IssuerPublicKeyCertificate = "";
        static String SignedStaticApplicationData = "";
        static String IssuerPublicKeyRemainder = "";
        static String IssuerPublicKeyExponent = "";


        /// DDA data element
        static String ICCPublicKeyCertificate = "";
        static String ICCPublicKeyExponent = "";
        static String ICCPublicKeyRemainder = "";


        /// CVM
        static String Plaintext_offline_PIN_verification = "NOT supported";
        static String Enciphered_online_PIN_verification = "NOT supported";
        static String Plaintext_offline_PIN_and_signature = "NOT
        supported";
        static String Enciphered_offline_PIN_verification = "NOT
        supported";
        static String Enciphered_offline_PIN_and_signature = "NOT
        supported";
        static String Signature_verification_only = "NOT supported";
        static String No_cardholder_verification_needed = "NOT supported";
        //CVM condition
        static String cPlaintext_offline_PIN_verification = "";
        static String cEnciphered_online_PIN_verification = "";
        static String cPlaintext_offline_PIN_and_signature = "";
        static String cEnciphered_offline_PIN_verification = "";
        static String cEnciphered_offline_PIN_and_signature = "";
        static String cSignature_verification_only = "";
        static String cNo_cardholder_verification_needed = "";
        // flag to determine the order of CVM
        static String first = "";
        static String second = "";
        static String third = "";
        static String fourth = "";
        static String fifth = "";
        static String sixth = "";
        static String seventh = "";


        /// application
        static String applicationLabel = "";
        static String applicationName = "";
        static String applicationPriority = "";
        static String applicationLanguage = "";
        static int numberOfApplications = 0;
        static boolean transaction = false;

        /// boolean flag
```

```
        static boolean SDA = false, DDA = false, cDDA = false, CVM = false;

        // constructor
        public EMVSoftwareReader()
        {
                JTabbedPane table = new JTabbedPane();
                table.addTab("Card Input", new InputForm());
                table.addTab("Payment Application", new applicationForm());
                table.addTab("Card Authentication", new camForm());
                table.addTab("Cardholder Verification", new cvmForm());
                table.addTab("Personalised information", new
                personalForm());

           JMenuBar menuBar = new JMenuBar();
           JMenu menu = new JMenu("File");
           JMenu menu1 = new JMenu("Help");
           menuBar.add(menu);
           menuBar.add(menu1);
           JMenuItem item1 = new JMenuItem("Exit");
           menu.add(item4);
           menu.add(item1);
           menu1.add(item2);

           ClickListener1 cl1 = new ClickListener1();
           item1.addActionListener(cl1);

           item2.addActionListener(this);
           item4.addActionListener(this);

                this.setJMenuBar(menuBar);
                this.add(table);

                this.setTitle("EMV Software Reader");
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                this.setSize(700, 650);
                this.setLocationRelativeTo(null);
                this.setVisible(true);
        }

        public void actionPerformed(ActionEvent e)
        {
                if (e.getSource() == item2)
                        JOptionPane.showMessageDialog(this, "Programmed
by Khuong Nguyen\nSupervised by Prof. Chris Mitchell", "EMV Software
Reader", 1);
                else
                if (e.getSource() == item4)
                {
                        generateReport();
                        JOptionPane.showMessageDialog(null, "Report has 
been generated!", "Report generation", 2);
                }
        }

        class ClickListener1 implements ActionListener
        {
                public void actionPerformed(ActionEvent e)
                {
                        System.exit(0);
                }
        }

        /*
         * Card Authentication Methods form
         */
        class camForm extends JPanel implements ActionListener
        {
                JPanel p1 = new JPanel();
                JPanel p2 = new JPanel();
                JPanel p3 = new JPanel();
                JPanel p4 = new JPanel();
                JPanel p5 = new JPanel();
                JPanel p6 = new JPanel();
                JPanel p7 = new JPanel();
                JPanel p8 = new JPanel();
                JPanel p9 = new JPanel();
                JPanel p10 = new JPanel();
                JPanel p11 = new JPanel();
                JPanel p12 = new JPanel();
                JPanel p13 = new JPanel();
                JPanel p14 = new JPanel();
                JPanel p15 = new JPanel();
                JPanel p16 = new JPanel();

                JPanel container = new JPanel();

                JButton bAnalyse = new JButton("Analyse");
                JButton bSDA = new JButton("Static Data Authentication");
                JButton bDDA = new JButton("Dynamic Data 
Authentication");
                JButton bCDA = new JButton("Combined DDA/Cryptogram");

                JLabel l1 = new JLabel("Static Data Authentication      ");
                JLabel l2 = new JLabel("Dynamic Data Authentication ");
                JLabel l3 = new JLabel("Combined DDA/Application 
cryptogram generation ");
```

```
		JLabel l4 = new JLabel("- - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - -");

		// SDA
		JLabel l5 = new JLabel("Certification Authority Public Key
Index  ");
		JLabel l6 = new JLabel("Issuer Public Key Certificate
");
		JLabel l7 = new JLabel("Signed Static Application Data
");
		JLabel l8 = new JLabel("Issuer Public Key Remainder
");
		JLabel l9 = new JLabel("Issuer Public Key Exponent
");
		JLabel l10 = new JLabel("ICC Public Key Certificate
");
		JLabel l11 = new JLabel("ICC Public Key Exponent
");
		JLabel l12 = new JLabel("ICC Public Key Remainder
");


		JTextField tx1 = new JTextField(10);	//
		JTextField tx2 = new JTextField(10);	//
		JTextField tx3 = new JTextField(10);	//
		JTextField tx4 = new JTextField(30);	//
		JTextField tx5 = new JTextField(30);	//
		JTextField tx6 = new JTextField(30);	//
		JTextField tx7 = new JTextField(30);	//
		JTextField tx8 = new JTextField(30);	//
		JTextField tx9 = new JTextField(30);	//
		JTextField tx10 = new JTextField(30);	//
		JTextField tx11 = new JTextField(30);	//
		JTextField tx12 = new JTextField(30);	//

		public camForm()
		{
			this.setLayout(new GridLayout(14, 1));
			this.add(p1); this.add(p2); this.add(p3);
this.add(p4); this.add(p5); this.add(p6);
			this.add(p8); this.add(p10); this.add(p9);
this.add(p11); this.add(p12); this.add(p13);
			this.add(p14); this.add(p15);

			p1.add(bAnalyse);
			p1.setLayout(new FlowLayout(FlowLayout.CENTER));
			bAnalyse.addActionListener(this);
```

```
        p2.add(l1); p2.add(tx1);
        tx1.setEditable(false);
        p2.setLayout(new FlowLayout(FlowLayout.LEFT));

        p3.add(l2); p3.add(tx2);
        tx2.setEditable(false);
        p3.setLayout(new FlowLayout(FlowLayout.LEFT));

        p4.add(l3); p4.add(tx3);
        tx3.setEditable(false);
        p4.setLayout(new FlowLayout(FlowLayout.LEFT));

        p5.add(l4);
        p5.setLayout(new FlowLayout(FlowLayout.CENTER));

        p6.add(bSDA);
        bSDA.addActionListener(this);
        p6.add(bDDA);
        bDDA.addActionListener(this);
        p6.add(bCDA);
        bCDA.addActionListener(this);
        p6.setLayout(new FlowLayout(FlowLayout.CENTER));

        /// for SDA
        p8.add(l5); p8.add(tx4);
        p8.setLayout(new FlowLayout(FlowLayout.LEFT));
        p9.add(l6); p9.add(tx5);
        p9.setLayout(new FlowLayout(FlowLayout.LEFT));
        p10.add(l7); p10.add(tx6);
        p10.setLayout(new FlowLayout(FlowLayout.LEFT));
        p11.add(l8); p11.add(tx7);
        p11.setLayout(new FlowLayout(FlowLayout.LEFT));
        p12.add(l9); p12.add(tx8);
        p12.setLayout(new FlowLayout(FlowLayout.LEFT));
        p13.add(l10); p13.add(tx9);
        p13.setLayout(new FlowLayout(FlowLayout.LEFT));
        p14.add(l11); p14.add(tx10);
        p14.setLayout(new FlowLayout(FlowLayout.LEFT));
        p15.add(l12); p15.add(tx11);
        p15.setLayout(new FlowLayout(FlowLayout.LEFT));
}

public void actionPerformed(ActionEvent e)
{
        if (e.getSource() == bAnalyse)
        {
                if (SDA == true)
                        tx1.setText("supported");
```

```
                    else
                            tx1.setText("NOT supported");
                    if (DDA == true)
                            tx2.setText("supported");
                    else
                            tx2.setText("NOT supported");
                    if (cDDA == true)
                            tx3.setText("supported");
                    else
                            tx3.setText("NOT supported");
            }
            else
            if (e.getSource() == bSDA)
            {
                    getSDAdata();

    tx4.setText(display(CertificationAuthorityPublicKeyIndex));

    tx5.setText(display(IssuerPublicKeyCertificate));
                    //tx6.setText(SignedStaticApplicationData);

    tx7.setText(display(IssuerPublicKeyRemainder));

    tx8.setText(display(IssuerPublicKeyExponent));
            }
            else
            if (e.getSource() == bDDA || e.getSource() ==
bCDA)
            {
                    getDDAdata();

    tx4.setText(display(CertificationAuthorityPublicKeyIndex));

    tx5.setText(display(IssuerPublicKeyCertificate));
                    //tx6.setText(SignedStaticApplicationData);

    tx7.setText(display(IssuerPublicKeyRemainder));

    tx8.setText(display(IssuerPublicKeyExponent));
            }
        }
    }

    // retrieve needed element for SDA authentication
    public void getSDAdata()
    {
        int index, length;
```

```
		// get CertificationAuthorityPublicKeyIndex, tag 8F
		index = AFL.indexOf("8F");
		if (index != -1)
		{
			//index = AFL.indexOf("8F", index + 1);
			length = hexToDec(AFL.substring(index + 2, index +
4 ));
			System.out.println(length);
			//System.out.println(AFL.substring(index + 2, index +
4 ));
			// multiple 2 since it is 2 bytes per hexa
			CertificationAuthorityPublicKeyIndex =
AFL.substring(index + 4, index + 4 + length * 2);
		}

		// get Issuer Public Key Certificate, tag 90
		index = AFL.indexOf("90");
		if (index != -1)
		{
			length = hexToDec(AFL.substring(index + 2, index +
4 ));
			// multiple 2 since it is 2 bytes per hexa
			IssuerPublicKeyCertificate = AFL.substring(index + 4,
index + 4 + length * 2);
		}

		//Signed Static Application Data, tag 93
		index = AFL.indexOf("93");
		if (index != -1)
		{
			length = hexToDec(AFL.substring(index + 2, index +
4 ));
			// multiple 2 since it is 2 bytes per hexa
			SignedStaticApplicationData = AFL.substring(index +
4, index + 4 + length * 2);
		}

		// Issuer Public Key Remainder, tag 92
		index = AFL.indexOf("92");
		if (index != -1)
		{
			length = hexToDec(AFL.substring(index + 2, index +
4 ));
			// multiple 2 since it is 2 bytes per hexa
			IssuerPublicKeyRemainder = AFL.substring(index + 4,
index + 4 + length * 2);
		}
```

```
                // Issuer Public Key Exponent, tag '9F32'
                index = AFL.indexOf("9F32");
                if (index != -1)
                {
                        length = hexToDec(AFL.substring(index + 4, index +
6 ));
                        // multiple 2 since it is 2 bytes per hexa
                        IssuerPublicKeyExponent = AFL.substring(index + 6,
index + 6 + length * 2);
                }

        }

        public void getDDAdata()
        {
                int index, length;

                // get CertificationAuthorityPublicKeyIndex, tag 8F
                index = AFL.indexOf("8F");
                if (index != -1)
                {
                        length = hexToDec(AFL.substring(index + 2, index +
4 ));
                        // multiple 2 since it is 2 bytes per hexa
                        CertificationAuthorityPublicKeyIndex =
AFL.substring(index + 4, index + 4 + length * 2);
                }

                // get Issuer Public Key Certificate, tag 90
                index = AFL.indexOf("90");
                if (index != -1)
                {
                        length = hexToDec(AFL.substring(index + 2, index +
4 ));
                        // multiple 2 since it is 2 bytes per hexa
                        IssuerPublicKeyCertificate = AFL.substring(index + 4,
index + 4 + length * 2);
                }

                // Issuer Public Key Remainder, tag 92
                index = AFL.indexOf("92");
                if (index != -1)
                {
                        length = hexToDec(AFL.substring(index + 2, index +
4 ));
                        // multiple 2 since it is 2 bytes per hexa
                        IssuerPublicKeyRemainder = AFL.substring(index + 4,
index + 4 + length * 2);
```

```
                }

                // Issuer Public Key Exponent, tag '9F32'
                index = AFL.indexOf("9F32");
                if (index != -1)
                {
                        length = hexToDec(AFL.substring(index + 4, index +
6 ));
                        // multiple 2 since it is 2 bytes per hexa
                        IssuerPublicKeyExponent = AFL.substring(index + 6,
index + 6 + length * 2);
                }

                // ICC Public Key Certificate, tag '9F46'
                index = AFL.indexOf("9F46");
                if (index != -1)
                {
                        length = hexToDec(AFL.substring(index + 4, index +
6 ));
                        // multiple 2 since it is 2 bytes per hexa
                        ICCPublicKeyCertificate = AFL.substring(index + 6,
index + 6 + length * 2);
                }

                // ICC Public Key Exponent, tag '9F47'
                index = AFL.indexOf("9F47");
                if (index != -1)
                {
                        length = hexToDec(AFL.substring(index + 4, index +
6 ));
                        // multiple 2 since it is 2 bytes per hexa
                        ICCPublicKeyExponent = AFL.substring(index + 6,
index + 6 + length * 2);
                }

                //ICC Public Key Remainder, tag '9F48'
                index = AFL.indexOf("9F48");
                if (index != -1)
                {
                        length = hexToDec(AFL.substring(index + 4, index +
6 ));
                        // multiple 2 since it is 2 bytes per hexa
                        ICCPublicKeyRemainder = AFL.substring(index + 6,
index + 6 + length * 2);
                }

        }
```

```
/*
 * Cardholder Verification Methods form
 */
class cvmForm extends JPanel implements ActionListener
{
        JPanel p1 = new JPanel();
        JPanel p2 = new JPanel();
        JPanel p3 = new JPanel();
        JPanel p4 = new JPanel();
        JPanel p5 = new JPanel();
        JPanel p6 = new JPanel();
        JPanel p7 = new JPanel();
        JPanel p8 = new JPanel();
        JPanel p9 = new JPanel();
        JPanel p10 = new JPanel();
        JPanel p11 = new JPanel();
        JPanel p12 = new JPanel();
        JPanel p13 = new JPanel();
        JPanel p14 = new JPanel();
        JPanel p15 = new JPanel();
        JPanel p16 = new JPanel();
        JPanel p17 = new JPanel();
        JPanel p18 = new JPanel();
        JPanel p19 = new JPanel();
        JPanel p20 = new JPanel();


        JButton bAnalyse = new JButton("Analyse");
        JButton bPIN = new JButton("PIN verification");

        JLabel l1 = new JLabel("Status");
        JLabel l2 = new JLabel("Plaintext offline PIN verification
");
        JLabel l3 = new JLabel("Enciphered online PIN verification
");
        JLabel l4 = new JLabel("Plaintext offline PIN and signature
");
        JLabel l5 = new JLabel("Enciphered offline PIN verification
");
        JLabel l6 = new JLabel("Enciphered offline PIN and signature
");
        JLabel l7 = new JLabel("Signature verification only
");
        JLabel l8 = new JLabel("No cardholder verification needed
");

        JLabel c2 = new JLabel("");
```

```
		JLabel c3 = new JLabel("");
		JLabel c4 = new JLabel("");
		JLabel c5 = new JLabel("");
		JLabel c6 = new JLabel("");
		JLabel c7 = new JLabel("");
		JLabel c8 = new JLabel("");

		JLabel order1 = new JLabel("");
		JLabel order2 = new JLabel("");
		JLabel order3 = new JLabel("");
		JLabel order4 = new JLabel("");
		JLabel order5 = new JLabel("");
		JLabel order6 = new JLabel("");
		JLabel order7 = new JLabel("");


		JTextField tx1 = new JTextField(20);		//
		JTextField tx2 = new JTextField(8);//
		JTextField tx3 = new JTextField(8);//
		JTextField tx4 = new JTextField(8);//
		JTextField tx5 = new JTextField(8);//
		JTextField tx6 = new JTextField(8);//
		JTextField tx7 = new JTextField(8);//
		JTextField tx8 = new JTextField(8);//


		JLabel text = new JLabel("Click on button to perform
plaintext offline PIN verification");
		JLabel text1 = new JLabel("The priority order of CVM in this
card");
		JLabel line = new JLabel("- - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");

		public cvmForm()
		{
			this.setLayout(new GridLayout(19, 1));
			this.add(p1); this.add(p2); this.add(p3);
this.add(p4); this.add(p5); this.add(p6);   this.add(p7); this.add(p8);
this.add(p9);
			this.add(p13);

			p1.add(bAnalyse);
			p1.setLayout(new FlowLayout(FlowLayout.CENTER));
			bAnalyse.addActionListener(this);

			p2.add(tx1);
			tx1.setEditable(false);
			p2.setLayout(new FlowLayout(FlowLayout.CENTER));
			p3.add(l2); p3.add(tx2); p3.add(c2);
			p3.setLayout(new FlowLayout(FlowLayout.LEFT));

			p4.add(l3); p4.add(tx3); p4.add(c3);
			p4.setLayout(new FlowLayout(FlowLayout.LEFT));

			p5.add(l4); p5.add(tx4); p5.add(c4);
			p5.setLayout(new FlowLayout(FlowLayout.LEFT));

			p6.add(l5); p6.add(tx5); p6.add(c5);
			p6.setLayout(new FlowLayout(FlowLayout.LEFT));

			p7.add(l6); p7.add(tx6); p7.add(c6);
			p7.setLayout(new FlowLayout(FlowLayout.LEFT));

			p8.add(l7); p8.add(tx7); p8.add(c7);
			p8.setLayout(new FlowLayout(FlowLayout.LEFT));

			p9.add(l8); p9.add(tx8); p9.add(c8);
			p9.setLayout(new FlowLayout(FlowLayout.LEFT));
			////////////////// Priority order
			p13.add(text1);
			p13.setLayout(new FlowLayout(FlowLayout.CENTER));

			//if (!first.isEmpty())
			{
				this.add(p14);
				p14.add(order1);
				p14.setLayout(new
FlowLayout(FlowLayout.LEFT));
			}
			//if (!second.isEmpty())
			{
				this.add(p15);
				p15.add(order2);
				p15.setLayout(new
FlowLayout(FlowLayout.LEFT));
			}
			//if (!third.isEmpty())
			{
				this.add(p16);
				p16.add(order3);
				p16.setLayout(new
FlowLayout(FlowLayout.LEFT));
			}
			//if (!fourth.isEmpty())
			{
				this.add(p17);
				p17.add(order4);
				p17.setLayout(new
FlowLayout(FlowLayout.LEFT));
			}
			//if (!fifth.isEmpty())
			{
				this.add(p18);
				p18.add(order5);
				p18.setLayout(new
FlowLayout(FlowLayout.LEFT));
			}
			//if (!sixth.isEmpty())
			{
				this.add(p19);
				p19.add(order6);
				p19.setLayout(new
FlowLayout(FlowLayout.LEFT));
			}
			//if (!seventh.isEmpty())
			{
				this.add(p20);
				p20.add(order7);
				p20.setLayout(new
FlowLayout(FlowLayout.LEFT));
			}
			///////////////////

			///////////////// PIN verification form
			//this.add(p12);
			this.add(p10); this.add(p11);

			p10.add(bPIN);
			bPIN.addActionListener(this);
			p10.setLayout(new FlowLayout(FlowLayout.CENTER));

			p11.add(text);
			p11.setLayout(new FlowLayout(FlowLayout.CENTER));

			p12.add(line);
			p12.setLayout(new FlowLayout(FlowLayout.CENTER));

		}

		public void actionPerformed(ActionEvent e)
		{
			if (e.getSource() == bAnalyse)
			{
```

```
                                if (CVM == true)
                                        tx1.setText("Cardholder verification is
supported");
                                else
                                // no need to perform any more CVM analysis
since ICC does not support it
                                {
                                        tx1.setText("Cardholder verification is
NOT supported");
                                        return;
                                }

                                // analyse CVM information
                                CVManalyse();

                                // set cvm
                                tx2.setText(Plaintext_offline_PIN_verification);

        tx3.setText(Enciphered_online_PIN_verification);

        tx4.setText(Plaintext_offline_PIN_and_signature);

        tx5.setText(Enciphered_offline_PIN_verification);

        tx6.setText(Enciphered_offline_PIN_and_signature);
                                tx7.setText(Signature_verification_only);

        tx8.setText(No_cardholder_verification_needed);

                                // set condition rule
                                c2.setText(cPlaintext_offline_PIN_verification);

        c3.setText(cEnciphered_online_PIN_verification);

        c4.setText(cPlaintext_offline_PIN_and_signature);

        c5.setText(cEnciphered_offline_PIN_verification);

        c6.setText(cEnciphered_offline_PIN_and_signature);
                                c7.setText(cSignature_verification_only);

        c8.setText(cNo_cardholder_verification_needed);

                                // set priority order
                                if (!first.isEmpty()) order1.setText("1. " +
first);
                                if (!second.isEmpty()) order2.setText("2. " +
second);
```

```
                                        if (!third.isEmpty()) order3.setText("3. " +
third);
                                        if (!fourth.isEmpty()) order4.setText("4. " +
fourth);
                                        if (!fifth.isEmpty()) order5.setText("5. " +
fifth);
                                        if (!sixth.isEmpty()) order6.setText("6. " +
sixth);
                                        if (!seventh.isEmpty()) order7.setText("7. " +
seventh);
                              }
                              else
                              if (e.getSource() == bPIN)
                              {
                                        new EMVPINVerification();
                              }
                    }
          }

          // analyse CVM information
          public void CVManalyse()
          {
                    String CVR = "", rule = "", condition = "";
                    int index, length;

                    // get CertificationAuthorityPublicKeyIndex, tag 8E
                    index = AFL.indexOf("8E");
                    if (index != -1)
                    {
                              length = hexToDec(AFL.substring(index + 2, index +
4 ));
                              // multiple 2 since it is 2 bytes per hexa
                              CVR = AFL.substring(index + 4, index + 4 + length *
2);


                              // ignore the first 8 bytes of X and Y
                              CVR = CVR.substring(16);
                    }

                    //42 01 41 03 1E 03 02 03 1F 03
                    index = 0;
                    while (index < CVR.length())
                    {
                              rule = CVR.substring(index, index + 2);
                              condition = CVR.substring(index + 2, index + 4);

                              // convert to 8 bit binary & trim the 6 bit right most
                              rule = hexToBin(rule).substring(2);
                    if (condition.equals("00"))
                          condition = "will always be performed";
                    else
                    if (condition.equals("01"))
                          condition = "is performed if unattended cash";
                    else
                    if (condition.equals("02"))
                          condition = "is performed if not unattended
cash, not manual cash and not purchase with cashback";
                    else
                    if (condition.equals("03"))
                          condition = "is performed if terminal supports
this method";
                    else
                    if (condition.equals("04"))
                          condition = "is performed if customer pays
with manual cash";
                    else
                    if (condition.equals("05"))
                          condition = "is performed if transaction with
cashback";
                    else
                          condition = "";

                    if (rule.equals("000001"))
                    {
                          Plaintext_offline_PIN_verification =
"supported";
                          cPlaintext_offline_PIN_verification = condition;
                          if (first.isEmpty()) first = "Plaintext offline PIN
verification";
                          else
                          if (second.isEmpty()) second = "Plaintext
offline PIN verification";
                          else
                          if (third.isEmpty()) third = "Plaintext offline
PIN verification";
                          else
                          if (fourth.isEmpty()) fourth = "Plaintext offline
PIN verification";
                          else
                          if (fifth.isEmpty()) fifth = "Plaintext offline PIN
verification";
                          else
                          if (sixth.isEmpty()) sixth = "Plaintext offline
PIN verification";
                          else
```

```
                                if (seventh.isEmpty()) seventh = "Plaintext
offline PIN verification";
                        }
                        else
                        if (rule.equals("000010"))
                        {
                                Enciphered_online_PIN_verification =
"supported";
                                cEnciphered_online_PIN_verification =
condition;
                                if (first.isEmpty()) first = "Enciphered online
PIN verification";
                                else
                                if (second.isEmpty()) second = "Enciphered
online PIN verification";
                                else
                                if (third.isEmpty()) third = "Enciphered online
PIN verification";
                                else
                                if (fourth.isEmpty()) fourth = "Enciphered
online PIN verification";
                                else
                                if (fifth.isEmpty()) fifth = "Enciphered online
PIN verification";
                                else
                                if (sixth.isEmpty()) sixth = "Enciphered online
PIN verification";
                                else
                                if (seventh.isEmpty()) seventh = "Enciphered
online PIN verification";
                        }
                        else
                        if (rule.equals("000011"))
                        {
                                Plaintext_offline_PIN_and_signature =
"supported";
                                cPlaintext_offline_PIN_and_signature =
condition;
                                if (first.isEmpty()) first = "Plaintext offline PIN
and signature";
                                else
                                if (second.isEmpty()) second = "Plaintext
offline PIN and signature";
                                else
                                if (third.isEmpty()) third = "Plaintext offline
PIN and signature";
                                else
```

```
                    if (fourth.isEmpty()) fourth = "Plaintext offline
PIN and signature";
                    else
                    if (fifth.isEmpty()) fifth = "Plaintext offline PIN
and signature";
                    else
                    if (sixth.isEmpty()) sixth = "Plaintext offline
PIN and signature";
                    else
                    if (seventh.isEmpty()) seventh = "Plaintext
offline PIN and signature";
               }
               else
               if (rule.equals("000100"))
               {
                    Enciphered_offline_PIN_verification =
"supported";
                    cEnciphered_offline_PIN_verification =
condition;
                    if (first.isEmpty()) first = "Enciphered offline
PIN verification";
                    else
                    if (second.isEmpty()) second = "Enciphered
offline PIN verification";
                    else
                    if (third.isEmpty()) third = "Enciphered offline
PIN verification";
                    else
                    if (fourth.isEmpty()) fourth = "Enciphered
offline PIN verification";
                    else
                    if (fifth.isEmpty()) fifth = "Enciphered offline
PIN verification";
                    else
                    if (sixth.isEmpty()) sixth = "Enciphered offline
PIN verification";
                    else
                    if (seventh.isEmpty()) seventh = "Enciphered
offline PIN verification";
               }
               else
               if (rule.equals("000101"))
               {
                    Enciphered_offline_PIN_and_signature =
"supported";
                    cEnciphered_offline_PIN_and_signature =
condition;
```

```
                                if (first.isEmpty()) first = "Enciphered offline
PIN and signature";
                                else
                                if (second.isEmpty()) second = "Enciphered
offline PIN and signature";
                                else
                                if (third.isEmpty()) third = "Enciphered offline
PIN and signature";
                                else
                                if (fourth.isEmpty()) fourth = "Enciphered
offline PIN and signature";
                                else
                                if (fifth.isEmpty()) fifth = "Enciphered offline
PIN and signature";
                                else
                                if (sixth.isEmpty()) sixth = "Enciphered offline
PIN and signature";
                                else
                                if (seventh.isEmpty()) seventh = "Enciphered
offline PIN and signature";
                        }
                        else
                        if (rule.equals("011110"))
                        {
                                Signature_verification_only = "supported";
                                cSignature_verification_only = condition;
                                if (first.isEmpty()) first = "Signature
verification only";
                                else
                                if (second.isEmpty()) second = "Signature
verification only";
                                else
                                if (third.isEmpty()) third = "Signature
verification only";
                                else
                                if (fourth.isEmpty()) fourth = "Signature
verification only";
                                else
                                if (fifth.isEmpty()) fifth = "Signature
verification only";
                                else
                                if (sixth.isEmpty()) sixth = "Signature
verification only";
                                else
                                if (seventh.isEmpty()) seventh = "Signature
verification only";
                        }
                        else
                        if (rule.equals("011111"))
                        {
                                No_cardholder_verification_needed =
"supported";
                                cNo_cardholder_verification_needed =
condition;
                                if (first.isEmpty()) first = "No cardholder
verification needed";
                                else
                                if (second.isEmpty()) second = "No cardholder
verification needed";
                                else
                                if (third.isEmpty()) third = "No cardholder
verification needed";
                                else
                                if (fourth.isEmpty()) fourth = "No cardholder
verification needed";
                                else
                                if (fifth.isEmpty()) fifth = "No cardholder
verification needed";
                                else
                                if (sixth.isEmpty()) sixth = "No cardholder
verification needed";
                                else
                                if (seventh.isEmpty()) seventh = "No
cardholder verification needed";
                        }

                        index += 4;
                }

        }

        /*
         * Personal information form
         */
        class personalForm extends JPanel implements ActionListener
        {
                JPanel p1 = new JPanel();
                JPanel p2 = new JPanel();
                JPanel p3 = new JPanel();
                JPanel p4 = new JPanel();
                JPanel p5 = new JPanel();
                JPanel p6 = new JPanel();
                JPanel p7 = new JPanel();
                JPanel p8 = new JPanel();

                JButton bInit = new JButton("Get personalised data");
```

```
		JLabel l1 = new JLabel("Name on card  ");
		JLabel l2 = new JLabel("Sixteen digits  ");
		JLabel l3 = new JLabel("Expire date      ");
		JLabel l4 = new JLabel("Effective date  ");
		JLabel l5 = new JLabel("Card type         ");
		JLabel l6 = new JLabel("BIC                   ");
		JLabel l7 = new JLabel("IBAN                ");

		JTextField tx1 = new JTextField(20);		// name on card
		JTextField tx2 = new JTextField(20);		// 16 digits
		JTextField tx3 = new JTextField(20);		// expire month
		JTextField tx4 = new JTextField(20);		// expire year
		JTextField tx5 = new JTextField(20);		// card type
		JTextField tx6 = new JTextField(20);		// bic
		JTextField tx7 = new JTextField(20);		// iban

		public personalForm()
		{
			this.setLayout(new GridLayout(8, 2));
			this.add(p1); this.add(p2); this.add(p7);
	this.add(p3); this.add(p4); this.add(p5); this.add(p6); this.add(p8);

			p1.add(bInit);
			p1.setLayout(new FlowLayout(FlowLayout.CENTER));
			bInit.addActionListener(this);

			p2.add(l1); p2.add(tx1);
			p2.setLayout(new FlowLayout(FlowLayout.LEFT));

			p3.add(l2); p3.add(tx2);
			p3.setLayout(new FlowLayout(FlowLayout.LEFT));

			p4.add(l3); p4.add(tx3);
			p4.setLayout(new FlowLayout(FlowLayout.LEFT));

			p5.add(l4); p5.add(tx4);
			p5.setLayout(new FlowLayout(FlowLayout.LEFT));

			p7.add(l5); p7.add(tx5);
			p7.setLayout(new FlowLayout(FlowLayout.LEFT));

			p6.add(l6); p6.add(tx6);
			p6.setLayout(new FlowLayout(FlowLayout.LEFT));

			p8.add(l7); p8.add(tx7);
			p8.setLayout(new FlowLayout(FlowLayout.LEFT));
		}
		public void actionPerformed(ActionEvent e)
		{
			// collect personal information
			if (e.getSource() == bInit)
			{
				getPersonalInformation(AFL);

				tx1.setText(nameOnCard);
				tx2.setText(sixteenDigit);
				tx3.setText(expireDate + " / " + expireMonth
+ " / " + expireYear);
				tx4.setText(beginDate + " / " + beginMonth +
" / " + beginYear);
				tx5.setText(cardType);
				tx6.setText(BIC);
				tx7.setText(IBAN);
			}
		}
	}


	class applicationForm extends JPanel implements ActionListener
	{
		JPanel p1 = new JPanel();
		JPanel p2 = new JPanel();
		JPanel p3 = new JPanel();
		JPanel p4 = new JPanel();
		JPanel p5 = new JPanel();
		JPanel p6 = new JPanel();
		JPanel p7 = new JPanel();
		JPanel p8 = new JPanel();
		JPanel p9 = new JPanel();
		JPanel p10 = new JPanel();
		JPanel p11 = new JPanel();
		JPanel p12 = new JPanel();
		JPanel p13 = new JPanel();
		JPanel p14 = new JPanel();

		JButton bStart = new JButton("Locate payment application");
	// verify payment application existence
		JButton bList = new JButton("List all");	// List all payment
applications on card
		JButton bTransaction = new JButton("Initiate transaction");

		JLabel appNum = new JLabel("Number of payment
application");
```

```
		JLabel line = new JLabel("- - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
		JLabel line1 = new JLabel("- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
		JLabel text1 = new JLabel("Check if there is payment
application on card");
		JLabel text2 = new JLabel("Initiate an EMV transaction with
ICC");

		JLabel l3 = new JLabel("Application label       ");
		JLabel l4 = new JLabel("Application priority   ");
		JLabel l5 = new JLabel("Preferred language  ");
		JLabel l6 = new JLabel("Preferred name        ");

		JTextField tx1 = new JTextField(15);	// status
		JTextField tx2 = new JTextField(3);	// for number of
applications
		JTextField tx3 = new JTextField(13);	// application label
		JTextField tx4 = new JTextField(13);	// priority
		JTextField tx5 = new JTextField(13);	// language
		JTextField tx6 = new JTextField(13);	// name
		JTextField tx7 = new JTextField(25);	// transaction
status

		public applicationForm()
		{
			this.setLayout(new GridLayout(14, 2));
			this.add(p1); this.add(p3); this.add(p14);
this.add(p4); this.add(p6); this.add(p5); this.add(p7); this.add(p8);
			this.add(p9); this.add(p10); this.add(p2);
this.add(p11); this.add(p12); this.add(p13);

			p1.add(bStart);
			p1.setLayout(new FlowLayout(FlowLayout.CENTER));
			bStart.addActionListener(this);

			p3.add(tx1);
			tx1.setEditable(false);
			p3.setLayout(new FlowLayout(FlowLayout.CENTER));

			p6.add(line);
			p6.setLayout(new FlowLayout(FlowLayout.CENTER));

			p4.add(appNum); p4.add(tx2);
			p4.setLayout(new FlowLayout(FlowLayout.LEFT));

			p5.add(bList);
			bList.addActionListener(this);
		p5.setLayout(new FlowLayout(FlowLayout.LEFT));


		p7.add(l3); p7.add(tx3);
		p7.setLayout(new FlowLayout(FlowLayout.LEFT));

		p8.add(l4); p8.add(tx4);
		p8.setLayout(new FlowLayout(FlowLayout.LEFT));

		p9.add(l5); p9.add(tx5);
		p9.setLayout(new FlowLayout(FlowLayout.LEFT));

		p10.add(l6); p10.add(tx6);
		p10.setLayout(new FlowLayout(FlowLayout.LEFT));

		p2.add(line1);
		p2.setLayout(new FlowLayout(FlowLayout.CENTER));

		p11.add(bTransaction);
		bTransaction.addActionListener(this);
		p11.setLayout(new FlowLayout(FlowLayout.CENTER));

		p12.add(tx7);
		tx7.setEditable(false);
		p12.setLayout(new FlowLayout(FlowLayout.CENTER));

		p13.add(text2);
		p13.setLayout(new FlowLayout(FlowLayout.CENTER));

		p14.add(text1);
		p14.setLayout(new FlowLayout(FlowLayout.CENTER));
	}

	public void actionPerformed(ActionEvent e)
	{
		try{
			// collect information and get ready for PIN
Verification
			if (e.getSource() == bStart)
			{
				capdu = new ApduCmd("00 A4 04 00
0E 31 50 41 59 2E 53 59 53 2E 44 44 46 30 31");
				rapdu = se[0].execute(capdu);
				//System.out.println(rapdu);		//
result of selecting 1PAY.SYS.DDF01

				if (rapdu.getStatusWord().isSuccess())
```

```
						tx1.setText("Payment
application found!");
					else
					{
						tx1.setText("Payment
application NOT found!");
						return;
					}

					capdu = new ApduCmd("00 B2 01 0C
00");
					rapdu = se[0].execute(capdu);
					// System.out.println(rapdu);
					String size =
NumUtil.hex2String(rapdu.getStatusWord().getSw2());
					//System.out.println(size);

					//////////////////////////////////////////
					// concat appropriate byte length
obtained above to get PSE data
					capdu = new ApduCmd("00B2010C" +
size);
					rapdu = se[0].execute(capdu);
					System.out.println(rapdu);

					// getAID(rapdu.toString());
					processApplication(rapdu.toString());

					//System.out.println("[Step 4] Now that
we know the AID, select the application");
					capdu = new ApduCmd("00 A4 04 00
07" + AID);

					rapdu = se[0].execute(capdu);

					capdu = new ApduCmd("80 A8 00 00
02 83 00");
					rapdu = se[0].execute(capdu);

					// check if transaction has been
initiated successfully
					if (rapdu.statusWord.isSuccess())
						transaction = true;
					else
					{
						transaction = false;
						return;
					}
```

```
        CAM_CVM_check(rapdu.toString().substring(10, 12));    // extract 2
byte AIP to check CAM & CVM
                                        //System.out.println(rapdu);

                                        // get & process AFL
                                        getAFL(rapdu.toString());

                                        tx2.setText("" +
numberOfApplications);
                                }
                                else
                                if (e.getSource() == bList)
                                {
                                        tx3.setText(applicationLabel);
                                        tx4.setText(applicationPriority);
                                        tx5.setText(applicationLanguage);
                                        tx6.setText(applicationName);
                                }
                                else
                                if (e.getSource() == bTransaction)
                                {
                                        if (transaction == true)
                                                tx7.setText("EMV transaction
has been successfully initialised");
                                        else
                                                tx7.setText("Cannot initiate EMV
transaction!");
                                }

                        } catch (CardException e1)
                        {
                                System.out.println("Error when accessing
PSE!");
                        }

                }
        }

        // this function lists all existed application in ICC
        public void processApplication(String st)
        {
                int index, length;
                st = sanitise(st);

                // only takes the part from 4F to end
                st = st.substring(st.indexOf("4F"));
```

```
		// scan all payment applications
		while (true)
		{
		// get AID value (in hexa), tag 4F
		index = st.indexOf("4F");
		if (index != -1)
		{
			length = hexToDec(st.substring(index + 2, index + 4
));
			// multiple 2 since it is 2 bytes per hexa
			AID = st.substring(index + 4, index + 4 + length *
2);

			// remove the AID from st
			st = st.substring(index + 4 + length * 2);
		}
		// since we know that there is no more application left
		else break;

		// get application label (in ASCII), tag 50
		index = st.indexOf("50");
		if (index != -1)
		{
			length = hexToDec(st.substring(index + 2, index + 4
));
			// multiple 2 since it is 2 bytes per hexa
			applicationLabel = st.substring(index + 4, index + 4
+ length * 2);

			// convert this hexa string to ASCII will give a
meaningful name
			applicationLabel = hexToASCII(applicationLabel);

			// remove the label from st
			st = st.substring(index + 4 + length * 2);
		}

		// get application priority, tag 87
		index = st.indexOf("87");
		if (index != -1)
		{
			length = hexToDec(st.substring(index + 2, index + 4
));
			// multiple 2 since it is 2 bytes per hexa
			applicationPriority = st.substring(index + 4, index + 4
+ length * 2);
```

```
                // remove the label from st
                st = st.substring(index + 4 + length * 2);
            }

            // get application language preference, tag 5F 2D
            index = st.indexOf("5F2D");
            if (index != -1)
            {
                length = hexToDec(st.substring(index + 4, index + 6
                ));
                // multiple 2 since it is 2 bytes per hexa
                applicationLanguage = st.substring(index + 6, index
                + 6 + length * 2);

                // convert this hexa string to ASCII will give a
                meaningful name
                applicationLanguage =
                hexToASCII(applicationLanguage);

                // remove the label from st
                st = st.substring(index + 6 + length * 2);
            }

            // get application preferred name, tag 9F 12
            index = st.indexOf("9F12");
            if (index != -1)
            {
                length = hexToDec(st.substring(index + 4, index + 6
                ));
                // multiple 2 since it is 2 bytes per hexa
                applicationName = st.substring(index + 6, index + 6
                + length * 2);

                // convert this hexa string to ASCII will give a
meaningful name
                applicationName = hexToASCII(applicationName);

                // remove the label from st
                st = st.substring(index + 6 + length * 2);
            }
            numberOfApplications++;
            }
    }


    public void getAFL(String st)
    {
            st = sanitise(st);
```

```
        st = st.substring(3, st.length() - 9);

        // get whole length string AFLs, signals by tag 80
        //int length = hexToDec(st.substring(st.indexOf("80") + 2,
        st.indexOf("80") + 4));

        // ignore the next 2 byte of AIP
        // we only interest in SFI
        // first position of SFI appearance
        int index = st.indexOf("80") + 8;

        int interval = -8;
        while (true)
        {
                interval += 8;
                // no more AFL to read
                if (index + interval + 2 >= st.length()) break;

                String SFI = st.substring(index + interval, index +
                interval + 2);
                SFI = hexToBin(SFI);

                // replace 100 at tail
                SFI = SFI.substring(0, SFI.length() - 3) + "100";

                // convert SFI to decimal
                int dec = Integer.valueOf(SFI, 2);

                // convert SFI to hexa from
                SFI = Integer.toHexString(dec);

                // restore zero if needed
                if (SFI.length() == 1) SFI = "0" + SFI;

                // apply SFI to read record
                processAFL(SFI);
        }
}


public void processAFL(String st)
{
        try {
                capdu = new ApduCmd("00B201" + st + "00");
                rapdu = se[0].execute(capdu);
                //System.out.println(rapdu);
                String size =
                NumUtil.hex2String(rapdu.getStatusWord().getSw2())
                ;

                capdu = new ApduCmd("00B201" + st + size);
                rapdu = se[0].execute(capdu);
                //System.out.println(rapdu);

                String SFI = sanitise(rapdu.toString());

                // append all results of READ RECORD into a big AFL
                AFL += SFI.substring(3, SFI.length() - 9);
                //System.out.println("This is AFL " + AFL);
        } catch (CardException e)
        {
                System.out.println("Error when accessing READ
                RECORD!");
        }
}



/*
 * Provide input form, reset ATR, prepare card
 */
class InputForm extends JPanel implements ActionListener
{
        JPanel p1 = new JPanel();
        JPanel p2 = new JPanel();
        JPanel p3 = new JPanel();
        JPanel p4 = new JPanel();

        JButton bInit = new JButton("Insert Card");

        JLabel l1 = new JLabel("ATR            ");
        JLabel l2 = new JLabel("SmartCard ");
        JLabel l3 = new JLabel("Status        ");

        JTextField tx1 = new JTextField(15);        // for init
        JTextField tx2 = new JTextField(25);        // for ATR
        JTextField tx3 = new JTextField(25);        // for Status

        JTextArea atrText = new JTextArea(2, 25);

        public InputForm()
        {
                //this.setLayout(new FlowLayout(FlowLayout.LEFT));
                this.setLayout(new GridLayout(4, 2));
                this.add(p1); this.add(p2); this.add(p3);
                this.add(p4);

                p1.add(bInit); p1.add(tx1);
                tx1.setText("Waiting for card ...");
                tx1.setEditable(false);
                p1.setLayout(new FlowLayout(FlowLayout.CENTER));
                bInit.addActionListener(this);

                p2.add(l1); p2.add(tx2);
                tx2.setEditable(false);
                p2.setLayout(new FlowLayout(FlowLayout.LEFT));

                p3.add(l2); p3.add(atrText);
                atrText.setBorder(new LineBorder(Color.BLACK, 1));
                atrText.setEditable(false);
                p3.setLayout(new FlowLayout(FlowLayout.LEFT));

                p4.add(l3); p4.add(tx3);
                tx3.setEditable(false);
                p4.setLayout(new FlowLayout(FlowLayout.LEFT));

            }

            public void actionPerformed(ActionEvent e)
            {
                // collect information and get ready for PIN
                Verification
                if (e.getSource() == bInit)
                {
                    tx1.setText("Card inserted!");
                    tx3.setText("Card is ready for access");
                    EMVStart(); // reset EMV & acquire ATR
                    tx2.setText(atrValue.substring(6));
                    if (!cardManufacturer.isEmpty())
                        atrText.setText(cardManufacturer);
                    else
                        atrText.setText("Cannot identify
smartcard manufacturer!");
                }
            }

        }

        public void EMVStart()
        {
            try{
            f = SessionFactory.getInstance();
```

```
                se = f.createSessions();

                for(int i = 0; i < se.length; i++)
                {
                        Atr atr = se[i].open();         // open new session with
EMV card
                        atrValue = atr.toString();
                        System.out.println(atr);

                        // get card Manufacturer
                        cardBrand(atr.toString());
                }
                } catch (CardException e)
                {
                        System.out.println("Error when process ATR!");
                }
        }


        // feed 2 byte AIP hexa into this function to check SDA, DDA, CDA &
CVM support
        public void CAM_CVM_check(String st)
        {
                int decimal = hexToDec(st); // convert hex to Dec
                st = Integer.toBinaryString(decimal);       // convert Dec to
Binary

                // append 0 at the beginning to restore 8 bit as
toBinaryString cut all zero at front
                while (st.length() < 8)
                        st = "0" + st;

                if (st.charAt(1) == '1') SDA = true;
                if (st.charAt(2) == '1') DDA = true;
                if (st.charAt(3) == '1') CVM = true;
                if (st.charAt(6) == '1') cDDA = true;

        }

        public void cardBrand(String atrCard)
        {
                // cut [ATR] part at the beginning of string
                atrCard = atrCard.substring(6, atrCard.length()-2);

                // check NatWest, HSBC, ...
                if (atrCard.equals("3B 6E 00 00 00 31 C0 71 C6 65 01 B0 01
03 37 83 90 00"))
                {
```

```
			cardManufacturer = " NatWest United Kingdom";
			return;
		}

		try	{
			FileReader dataIn = new
FileReader("C://Data//smartcard_list.txt");
			BufferedReader f = new BufferedReader(dataIn);

			String line;
			while (true)
			{
				while (true)
				{
					line = f.readLine();
					if (!line.isEmpty() && line.charAt(0) >=
'0' && line.charAt(0) <= '9')	break;
					if (line.equals("# do not delete"))
break;
				}
				if (line.equals("# do not delete")) break;

				if (atrCard.equals(line))
					while (true)
					{
						line = f.readLine();
						line = line.trim();
						if (line.isEmpty()) break;
						System.out.println(line);
						cardManufacturer += line +
"\n";
					}
			}

		} catch (IOException e)
		{
			System.out.println("IO Error with
smartcard_list.txt!");
		}

	}

	// retrieving personal information
	public void getPersonalInformation(String st)
	{
		int index;

		// effective date
```

```
                index = st.indexOf("5F25");
                if (index != -1)
                {
                        beginYear = st.substring(index + 6, index + 8);
                        beginMonth = st.substring(index + 8, index + 10);
                        beginDate = st.substring(index + 10, index + 12);
                }

                // expire date
                index = st.indexOf("5F24");
                if (index != -1)
                {
                        expireYear = st.substring(index + 6, index + 8);
                        expireMonth = st.substring(index + 8, index + 10);
                        expireDate = st.substring(index + 10, index + 12);
                }

                // 16 digits
                index = st.indexOf("5A");
                if (index != -1)
                {
                        sixteenDigit = st.substring(index + 4, index + 20);
                }

                // get Card Holder name, tag 5F 20
                if (st.indexOf("5F20") != -1)
                {
                        int nameLength =
hexToDec(st.substring(st.indexOf("5F20") + 4, st.indexOf("5F20") + 6 ));
                        nameOnCard = st.substring(st.indexOf("5F20") + 6,
st.indexOf("5F20") + 6 + nameLength * 2);

                        // covert the hexa string nameOnCard will give a
meaningful ASCII name
                        nameOnCard = hexToASCII(nameOnCard);
                }

                // get BIC, tag 5F 54
                if (st.indexOf("5F54") != -1)
                {
                        int nameLength =
hexToDec(st.substring(st.indexOf("5F54") + 4, st.indexOf("5F54") + 6 ));
                        BIC = st.substring(st.indexOf("5F54") + 6,
st.indexOf("5F54") + 6 + nameLength * 2);
                }

                // get IBAN, tag 5F 53
                if (st.indexOf("5F53") != -1)
```

```
		{
			int nameLength = 
hexToDec(st.substring(st.indexOf("5F53") + 4, st.indexOf("5F53") + 6 ));
			IBAN = st.substring(st.indexOf("5F53") + 6, 
st.indexOf("5F53") + 6 + nameLength * 2);
		}

		// analyse card type
		getCardType();
		getExpireMonth();
		getBeginMonth();
	}

	// conver expire month to letters
	public void getExpireMonth()
	{
		int eMonth = Integer.parseInt(expireMonth);
		switch (eMonth)
		{
			case 1: expireMonth = "January"; break;
			case 2: expireMonth = "February"; break;
			case 3: expireMonth = "March"; break;
			case 4: expireMonth = "April"; break;
			case 5: expireMonth = "May"; break;
			case 6: expireMonth = "June"; break;
			case 7: expireMonth = "July"; break;
			case 8: expireMonth = "August"; break;
			case 9: expireMonth = "September"; break;
			case 10: expireMonth = "October"; break;
			case 11: expireMonth = "November"; break;
			case 12: expireMonth = "December"; break;
		}

		expireYear = "20" + expireYear;
	}

	// conver effective month to letters
	public void getBeginMonth()
	{
		int eMonth = Integer.parseInt(beginMonth);
		switch (eMonth)
		{
			case 1: beginMonth = "January"; break;
			case 2: beginMonth = "February"; break;
			case 3: beginMonth = "March"; break;
			case 4: beginMonth = "April"; break;
			case 5: beginMonth = "May"; break;
			case 6: beginMonth = "June"; break;
```

```
                case 7: beginMonth = "July"; break;
                case 8: beginMonth = "August"; break;
                case 9: beginMonth = "September"; break;
                case 10: beginMonth = "October"; break;
                case 11: beginMonth = "November"; break;
                case 12: beginMonth = "December"; break;
        }

        beginYear = "20" + beginYear;
}

// analyse card type based on 16 digits
public void getCardType()
{
        if (sixteenDigit.substring(0, 6).equals("417500") ||
                        sixteenDigit.substring(0, 4).equals("4917") ||
                        sixteenDigit.substring(0, 4).equals("4913") ||
                        sixteenDigit.substring(0, 4).equals("4508") ||
                        sixteenDigit.substring(0, 4).equals("4844"))
                        cardType = "VISA Electron";
                else
                if (sixteenDigit.substring(0, 1).equals("4"))
                        cardType = "VISA";
                else
                if (sixteenDigit.substring(0, 2).equals("51") ||
                        sixteenDigit.substring(0, 2).equals("52") ||
                        sixteenDigit.substring(0, 2).equals("53") ||
                        sixteenDigit.substring(0, 2).equals("54") ||
                        sixteenDigit.substring(0, 2).equals("55"))
                        cardType = "MasterCard";
                else
                if (sixteenDigit.substring(0, 2).equals("34") ||
                sixteenDigit.substring(0, 2).equals("37"))
                        cardType = "American Express";
                else
                if (sixteenDigit.substring(0, 3).equals("300") ||
                        sixteenDigit.substring(0, 3).equals("301") ||
                        sixteenDigit.substring(0, 3).equals("302") ||
                        sixteenDigit.substring(0, 3).equals("303") ||
                        sixteenDigit.substring(0, 3).equals("304") ||
                        sixteenDigit.substring(0, 3).equals("305"))
                        cardType = "Diners Club Blanche";
                else
                if (sixteenDigit.substring(0, 4).equals("2014") ||
                sixteenDigit.substring(0, 4).equals("2149"))
                        cardType = "Diners Club enRoute";
                else
                if (sixteenDigit.substring(0, 2).equals("36"))
                        cardType = "Diners Club International";
                else

                if (sixteenDigit.substring(0, 2).equals("54") ||
                sixteenDigit.substring(0, 2).equals("55"))

                        cardType = "Diners Club US & Canada";
                else
                if (sixteenDigit.substring(0, 4).equals("6011") ||
                        sixteenDigit.substring(0, 3).equals("622") ||
                        sixteenDigit.substring(0, 2).equals("64") ||
                        sixteenDigit.substring(0, 2).equals("65"))
                        cardType = "Discover Card";
                else
                if (sixteenDigit.substring(0, 4).equals("3528") ||
                        sixteenDigit.substring(0, 4).equals("3589"))
                        cardType = "JCB";
                else
                if (sixteenDigit.substring(0, 4).equals("6304") ||
                        sixteenDigit.substring(0, 4).equals("6706") ||
                        sixteenDigit.substring(0, 4).equals("6771") ||
                        sixteenDigit.substring(0, 4).equals("6709"))
                        cardType = "Laser Card";
                else
                if (sixteenDigit.substring(0, 4).equals("5018") ||
                        sixteenDigit.substring(0, 4).equals("5020") ||
                        sixteenDigit.substring(0, 4).equals("5038") ||
                        sixteenDigit.substring(0, 4).equals("6304") ||
                        sixteenDigit.substring(0, 4).equals("6759") ||
                        sixteenDigit.substring(0, 4).equals("6761") ||
                        sixteenDigit.substring(0, 4).equals("6763"))
                        cardType = "Maestro";
                else
                if (sixteenDigit.substring(0, 4).equals("6334") ||
                        sixteenDigit.substring(0, 4).equals("6767"))
                        cardType = "SOLO";
                else
                if (sixteenDigit.substring(0, 4).equals("4903") ||
                        sixteenDigit.substring(0, 4).equals("4905") ||
                        sixteenDigit.substring(0, 4).equals("4911") ||
                        sixteenDigit.substring(0, 4).equals("4936") ||
                        sixteenDigit.substring(0, 4).equals("6333") ||
                        sixteenDigit.substring(0, 4).equals("6759") ||
                        sixteenDigit.substring(0, 6).equals("564182")
                        ||
                        sixteenDigit.substring(0, 6).equals("633110"))
                        cardType = "SWITCH";
                else
                if (sixteenDigit.substring(0, 6).equals("502293"))
```

```
                        cardType = "One World Bancorp";
                else
                if (sixteenDigit.substring(0, 6).equals("606263"))
                        cardType = "MONEYTECH";
}


public void generateReport()
{
        try{
                PrintWriter f = new
                PrintWriter("C://Data//report.txt");
                f.println("*** EMV card survey report");
                f.println();

                f.println("Card is reseted, the ATR value returned is "
                + atrValue);

                if (!cardManufacturer.isEmpty())
                        f.println("Card manufacturer is " +
                cardManufacturer);
                else
                        f.println("Cannot trace back card
                manufacturer");


                f.println();
                f.println();
                ////////////////////////////////////////////////////////

                f.println("* * * * * * * * * * * * * * * * * * * * *
*");
                f.println("*    Card payment applications survey
*");
                f.println("* * * * * * * * * * * * * * * * * * * * *
*");
                f.println();
                f.println("Payment application folder exists in this
card");
                if (numberOfApplications == 1)
                        f.println("There is only one payment
application on this card");
                else
                        f.println("There are " + numberOfApplications
+ " payment applications on this card");

                f.println();
                f.println("The list of all payment applications:");
```

```
f.println();

/// application
f.println("- Application label is " + applicationLabel);
f.println("- Application priority is " +
applicationPriority);
if (!applicationName.isEmpty())
        f.println("- Application preferred name is " +
applicationName);
if (!applicationLanguage.isEmpty())
        f.println("- Application preferred language is "
+ applicationLanguage);

f.println();
f.println();
//////////////////////////////////////////////////////

f.println("* * * * * * * * * * * * * * * * * * * * *
*");
f.println("*   Card authentication methods survey
*");
f.println("* * * * * * * * * * * * * * * * * * * * *
*");
f.println();
f.println("Card authentication methods checking: ");
if (SDA)
        f.println("- Static data authentication is
supported");
else
        f.println("- Static data authentication is NOT
supported");

if (DDA)
        f.println("- Dynamic data authentication is
supported");
else
        f.println("- Dynamic data authentication is NOT
supported");

if (cDDA)
        f.println("- Combined dynamic data
authentication and application cryptogram generation
is supported");
else
        f.println("- Combined dynamic data
authentication and application cryptogram generation
is NOT supported");
```

```
			f.println();
			f.println("The data element for static data
			authentication found on this card: ");
			if (!CertificationAuthorityPublicKeyIndex.isEmpty())
				f.println("- Certification Authority Public Key
			Index is " + CertificationAuthorityPublicKeyIndex);
			if (!IssuerPublicKeyCertificate.isEmpty())
				f.println("- Issuer Public Key Certificate is " +
			IssuerPublicKeyCertificate);
			if (!SignedStaticApplicationData.isEmpty())
				f.println("- Signed Static Application Data is "
			+ SignedStaticApplicationData);
			if (!IssuerPublicKeyRemainder.isEmpty())
				f.println("- Issuer Public Key Remainder is " +
			IssuerPublicKeyRemainder);
			if (!IssuerPublicKeyExponent.isEmpty())
				f.println("- Issuer Public Key Exponent is " +
			IssuerPublicKeyExponent);

			f.println();
			f.println();
			////////////////////////////////////////////////////////

			f.println("* * * * * * * * * * * * * * * * * * * * * *
			*");
			f.println("* Cardholder verification methods survey
			*");
			f.println("* * * * * * * * * * * * * * * * * * * * * *
			*");
			f.println();
			f.println("Verification methods checking: ");
			if (CVM)
			{
				f.println("- Plaintext offline verification
Plaintext is " + Plaintext_offline_PIN_verification + ". This method " +
cPlaintext_offline_PIN_verification);
				f.println("- Enciphered online PIN verification is
" + Enciphered_online_PIN_verification + ". This method " +
cEnciphered_online_PIN_verification);
				f.println("- Plaintext offline PIN and signature
is " + Plaintext_offline_PIN_and_signature + ". This method " +
cPlaintext_offline_PIN_and_signature);
				f.println("- Enciphered offline PIN verification is
" + Enciphered_offline_PIN_verification + ". This method " +
cEnciphered_offline_PIN_verification);
				f.println("- Enciphered offline PIN and
signature is " + Enciphered_offline_PIN_and_signature + ". This method "
+ cEnciphered_offline_PIN_and_signature);
```

```
                    f.println("- Signature verification only is " +
Signature_verification_only + ". This method " +
cSignature_verification_only);
                    f.println("- No cardholder verification needed is
                    " + No_cardholder_verification_needed + ".
                    This method " +
                    cNo_cardholder_verification_needed);
                    f.println();
                    f.println("The priority order of CVM in this card
                    is:");
                    if (!first.isEmpty()) f.println("1. " + first);
                    if (!second.isEmpty()) f.println("2. " +
                    second);
                    if (!third.isEmpty()) f.println("3. " + third);
                    if (!fourth.isEmpty()) f.println("4. " + fourth);
                    if (!fifth.isEmpty()) f.println("5. " + fifth);
                    if (!sixth.isEmpty()) f.println("6. " + sixth);
                    if (!seventh.isEmpty()) f.println("7. " +
                    seventh);
            }
            else
                    f.println("This card does not support any
                    cardholder verification method");

            f.println();
            f.println();
            /////////////////////////////////////////////////////////


            f.println("* * * * * * * * * * * * * * * * * * * * *
            *");
            f.println("*       Personal information survey       *");
            f.println("* * * * * * * * * * * * * * * * * * * * *
            *");
            f.println();
            f.println("Cardholder name is " + nameOnCard);
            f.println("Card type is " + cardType);
            f.println("Card sixteen digits are " +
            sixteenDigit.substring(0, 12) + "****");
            f.println("This card is effective from " + beginDate + "
            / " + beginMonth + " / " + beginYear);
            f.println("This card will expire on " + expireDate + " /
            " + expireMonth + " / " + expireYear);
            if (!BIC.isEmpty())
                    f.println("The bank\\'s BIC is " + BIC);
            if (!IBAN.isEmpty())
                    f.println("The bank\\'s IBAN is " + IBAN);
```

```
                f.close();

        } catch (IOException e)
        {
                System.out.println("Error when generating report!");
        }
}

////////////////////////////////////////////////////////
// These are supported functions to process hexadecimal

// convert Hex to ASCII
public static String hexToASCII(String hexa)
{
        int[] text = new int [hexa.length() / 2];
        int j = 0;
        String ascii = "";
        StringBuilder s = new StringBuilder(hexa);

        // remove all space within hexa String
        for (int i=0; i < s.length(); i++)
                if (s.charAt(i) == ' ')
                        s.deleteCharAt(i);

        // assign new non-space string back to hexa
        hexa = s.toString();

        for (int i = 0; i < hexa.length(); i += 2)
                text[j++] = Integer.parseInt(hexa.substring(i, i + 2),
                16);

        for (int i=0; i<text.length; i++)
        {
                ascii += (char)text[i];
                // print ascii text to screen too!
                //System.out.print((char)text[i] + " ");
        }

        return ascii;
}


// convert Hex to Decimal (number)
public int hexToDec(String hex)
{
        return Integer.parseInt(hex, 16);
}
```

```
	// convert Hex to Binary (number), restore zero at front too
	public String hexToBin(String hex)
	{
		String s = Integer.toBinaryString(hexToDec(hex));

		// restore zero
		while (s.length() < 8)
			s = "0" + s;

		return s;
	}

	public String sanitise(String st)
	{
		StringBuilder s = new StringBuilder(st);

		for (int i=0; i < s.length(); i++)
			if (s.charAt(i) == ' ')
				s.deleteCharAt(i);

		return s.toString();
	}

	// restore space for display purpose
	public String display(String st)
	{
		StringBuilder s = new StringBuilder(st);

		for (int i=s.length(); i >= 0 ; i--)
			if (i % 2 == 0)
				s.insert(i, " ");

		return s.toString();
	}


	public boolean checkPDOL(String st)
	{
		return (st.indexOf("9F 38") != -1);
	}

	public static void main(String[] args)
	{
		// GUI
		new EMVSoftwareReader();

	}
}
```

```
EMVPINVerification.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;

import com.jaccal.Atr;
import com.jaccal.CardException;
import com.jaccal.CardResponse;
import com.jaccal.Session;
import com.jaccal.SessionFactory;
import com.jaccal.command.ApduCmd;
import com.jaccal.util.NumUtil;

public class EMVPINVerification extends JFrame
{
   static String AID = "";
   static SessionFactory f;
        static Session[] se;
        static ApduCmd capdu;
        static CardResponse rapdu;

        public EMVPINVerification()
        {
                JTabbedPane table = new JTabbedPane();
                table.addTab("PIN Verification", new PINForm());

                this.add(table);
                this.setTitle("Offline PIN verification");
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                this.setSize(400, 650);
                this.setLocationRelativeTo(null);
                this.setVisible(true);
        }

        class PINForm extends JPanel implements ActionListener,
FocusListener, KeyListener
        {
                JPanel pa00 = new JPanel();
                JPanel pa0 = new JPanel();
                JPanel pa1 = new JPanel();
                JPanel pa2 = new JPanel();
                JPanel pa3 = new JPanel();
                JPanel pa4 = new JPanel();
                JPanel pa5 = new JPanel();

                JButton bInit = new JButton("Initialise");
```

```
		JButton b1 = new JButton(new
ImageIcon("C://Data//EMVImage//1.png"));
		JButton b2 = new JButton(new
ImageIcon("C://Data//EMVImage//2.png"));
		JButton b3 = new JButton(new
ImageIcon("C://Data//EMVImage//3.png"));
		JButton b4 = new JButton(new
ImageIcon("C://Data//EMVImage//4.png"));
		JButton b5 = new JButton(new
ImageIcon("C://Data//EMVImage//5.png"));
		JButton b6 = new JButton(new
ImageIcon("C://Data//EMVImage//6.png"));
		JButton b7 = new JButton(new
ImageIcon("C://Data//EMVImage//7.png"));
		JButton b8 = new JButton(new
ImageIcon("C://Data//EMVImage//8.png"));
		JButton b9 = new JButton(new
ImageIcon("C://Data//EMVImage//9.png"));
		JButton b0 = new JButton(new
ImageIcon("C://Data//EMVImage//0.png"));
		JButton bE1 = new JButton(new
ImageIcon("C://Data//EMVImage//empty.png"));
		JButton bE2 = new JButton(new
ImageIcon("C://Data//EMVImage//empty.png"));
		JButton bEnter = new JButton(new
ImageIcon("C://Data//EMVImage//enter.png"));
		JButton bClear = new JButton(new
ImageIcon("C://Data//EMVImage//clear.png"));
		JButton bCancel = new JButton(new
ImageIcon("C://Data//EMVImage//cancel.png"));

		JTextArea status = new JTextArea(1, 29);
		JTextArea screen = new JTextArea(2, 29);

		JLabel space1 = new JLabel("        ");
		JLabel space2 = new JLabel("        ");
		JLabel space3 = new JLabel("        ");
		JLabel space4 = new JLabel("        ");

		String star = "";  // anonymous * on screen
		String pin = "";   // plaintext PIN
		int pinTry = 0;

		boolean focus = false;

		public PINForm()
		{
```

```
b1.setBorder(null); b2.setBorder(null);
b3.setBorder(null);
b4.setBorder(null); b5.setBorder(null);
b6.setBorder(null);
b7.setBorder(null); b8.setBorder(null);
b9.setBorder(null);
b0.setBorder(null); bE1.setBorder(null);
bE2.setBorder(null);
b0.addActionListener(this);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
b5.addActionListener(this);
b6.addActionListener(this);
b7.addActionListener(this);
b8.addActionListener(this);
b9.addActionListener(this);
bEnter.setBorder(null); bClear.setBorder(null);
bCancel.setBorder(null);
bEnter.addActionListener(this);
bClear.addActionListener(this);
bCancel.addActionListener(this);

//this.setLayout(new FlowLayout(FlowLayout.LEFT));
this.setLayout(new GridLayout(7, 1));
this.add(pa1); this.add(pa00); this.add(pa0);
this.add(pa2); this.add(pa3); this.add(pa4);
this.add(pa5);

pa1.add(bInit);
pa1.setLayout(new FlowLayout(FlowLayout.CENTER));
bInit.addActionListener(this);

pa00.add(status);
status.setBorder(new LineBorder(Color.GRAY, 5));
status.setFont(new Font("monospaced", Font.BOLD,
20));
status.setForeground(Color.BLUE);
status.setEditable(false);
pa00.setLayout(new FlowLayout(FlowLayout.LEFT));

pa0.add(screen);

screen.addFocusListener(this);
screen.addKeyListener(this);

screen.setBorder(new LineBorder(Color.RED, 5));
```

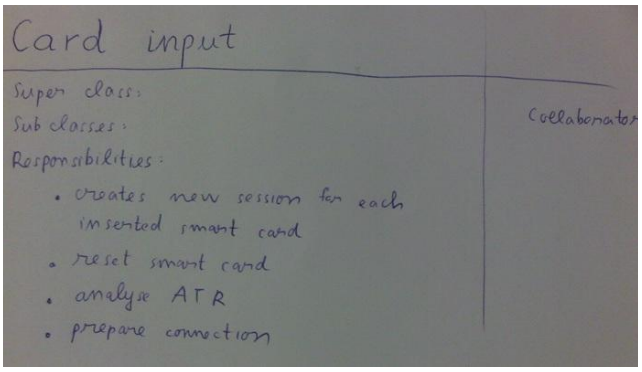
```
        screen.setFont(new Font("monospaced", Font.BOLD,
        20));
        screen.setForeground(Color.BLUE);
        screen.setEditable(false);
        pa0.setLayout(new FlowLayout(FlowLayout.LEFT));

        pa2.add(b1); pa2.add(b2); pa2.add(b3);
        pa2.add(space1); pa2.add(bEnter);
        pa2.setLayout(new FlowLayout(FlowLayout.LEFT));
        pa3.add(b4); pa3.add(b5); pa3.add(b6);
        pa3.add(space2); pa3.add(bClear);
        pa3.setLayout(new FlowLayout(FlowLayout.LEFT));
        pa4.add(b7); pa4.add(b8); pa4.add(b9);
        pa4.add(space3); pa4.add(bCancel);
        pa4.setLayout(new FlowLayout(FlowLayout.LEFT));
        pa5.add(bE1); pa5.add(b0); pa5.add(bE2);
        pa5.setLayout(new FlowLayout(FlowLayout.LEFT));

}

public void keyTyped(KeyEvent e)
{
        char key = e.getKeyChar();

        if (key == '0')
        {
                star += " * ";
                screen.setText(star);
                pin += "0";
                screen.repaint();
        }
        else
        if (key == '1')
        {
                star += " * ";
                screen.setText(star);
                pin += "1";
                screen.repaint();
        }
        else
        if (key == '2')
        {
                star += " * ";
                screen.setText(star);
                pin += "2";
                screen.repaint();
        }
        else
```

```
if (key == '3')
{
        star += " * ";
        screen.setText(star);
        pin += "3";
        screen.repaint();
}
else
if (key == '4')
{
        star += " * ";
        screen.setText(star);
        pin += "4";
        screen.repaint();
}
else
if (key == '5')
{
        star += " * ";
        screen.setText(star);
        pin += "5";
        screen.repaint();
}
else
if (key == '6')
{
        star += " * ";
        screen.setText(star);
        pin += "6";
        screen.repaint();
}
else
if (key == '7')
{
        star += " * ";
        screen.setText(star);
        pin += "7";
        screen.repaint();
}
else
if (key == '8')
{
        star += " * ";
        screen.setText(star);
        pin += "8";
        screen.repaint();
}
else
```

```
        if (key == '9')
        {
                star += " * ";
                screen.setText(star);
                pin += "9";
                screen.repaint();
        }

}

public void keyPressed(KeyEvent e)
{

}

public void keyReleased(KeyEvent e)
{

}

public void focusGained(FocusEvent e)
{
        screen.repaint();
        //screen.revalidate();
        focus = true;
        screen.setBorder(new LineBorder(Color.RED, 5));
}


public void focusLost(FocusEvent e)
{
        screen.repaint();
        //screen.revalidate();
        focus = false;
        screen.setBorder(new LineBorder(Color.GRAY, 5));
}

public void actionPerformed(ActionEvent e)
{
        // collect information and get ready for PIN
        Verification
        if (e.getSource() == bInit)
        {
                PINProcess(); // initialise ICC
                pinTry = getPINTryCounter();
                if (pinTry == 1)
                {
                        status.setForeground(Color.RED);
                status.setText("Only ONE PIN try
                left!");
        }
        else
        {
                status.setForeground(Color.BLUE);
                status.setText("PIN tries remaining: " +
                pinTry);
        }
}
else
// PIN input
if (e.getSource() == b1 || e.getSource() == b2 ||
e.getSource() == b3 || e.getSource() == b4 ||
e.getSource() == b5 ||
e.getSource() == b6 || e.getSource() == b7 ||
e.getSource() == b8 || e.getSource() == b9 ||
e.getSource() == b0 )
{
        star += " * ";
        screen.setText(star);
        if ( e.getSource() == b1 ) pin += "1"; else
        if ( e.getSource() == b2 ) pin += "2"; else
        if ( e.getSource() == b3 ) pin += "3"; else
        if ( e.getSource() == b4 ) pin += "4"; else
        if ( e.getSource() == b5 ) pin += "5"; else
        if ( e.getSource() == b6 ) pin += "6"; else
        if ( e.getSource() == b7 ) pin += "7"; else
        if ( e.getSource() == b8 ) pin += "8"; else
        if ( e.getSource() == b9 ) pin += "9"; else
        if ( e.getSource() == b0 ) pin += "0";
}
else
if (e.getSource() == bEnter)  // send entered PIN to
ICC
{
        if (pin.length() < 4 || pin.length() > 12)    //
        wrong PIN length, reset PIN too
        {
                status.setText("Invalid PIN length");
                screen.setText("");
                pin = "";
                star = "";
        }
        else    // perform PIN verification
        {
                if (verifyPIN(pin))
        }

                                        screen.setText("PIN entered
                                        Successful!");
                                else
                                        screen.setText("PIN does not
                                        match!");

                                pinTry = getPINTryCounter();
                                if (pinTry == 1)
                                {
                                status.setForeground(Color.RED);
                                status.setText("Only ONE PIN try
                                left!");
                                }
                                else
                                {
                                status.setForeground(Color.BLUE);
                                status.setText("PIN tries remaining: " +
                                pinTry);
                                }
                                pin = "";
                                star = "";
                        }
                }
                else
                if (e.getSource() == bClear)  // clear screen & data
                {
                        screen.setText("");
                        star = "";
                        pin = "";
                }
                else
                if (e.getSource() == bCancel)  // terminate process
                {
                        status.setText("Card is ejected");
                        screen.setText("");
                        star = "";
                        pin = "";
                        try {
                                se[0].close();
                                f.resetSessionFactory();
                        } catch (CardException e1) { }
                        return;
                }
        }
}

public static int hexToDec(String hex)
{
        return Integer.parseInt(hex, 16);
}

public static void getAID(String st)
{
        StringBuilder s = new StringBuilder(st);

        for (int i=0; i < s.length(); i++)
                if (s.charAt(i) == ' ')
                        s.deleteCharAt(i);

        st = s.toString();
        int AIDLength = hexToDec(st.substring(st.indexOf("4F") + 2,
        st.indexOf("4F") + 4 ));

        AID = st.substring(st.indexOf("4F") + 4, st.indexOf("4F") +
        4 + AIDLength * 2);

}

// extract the PIN Try Counter number
public int getPINTryCounter()
{
        StringBuilder s = new StringBuilder();
        try{
                System.out.println("GETTING PIN TRY COUNTER");
                capdu = new ApduCmd("80 CA 9F 17 00");
                rapdu = se[0].execute(capdu);
                System.out.println(rapdu);

                System.out.println("Try again with new length");
                capdu = new ApduCmd("80 CA 9F 17 04");
                rapdu = se[0].execute(capdu);
                System.out.println(rapdu);
                /////////////

                s = new StringBuilder(rapdu.toString());

                for (int i=0; i < s.length(); i++)
                        if (s.charAt(i) == ' ')
                                s.deleteCharAt(i);

        } catch (CardException e)
        {
                System.out.println("Error when getting PIN try
                counter!");
        }
        String st = s.toString();
        return Integer.parseInt(st.substring(st.indexOf("9F17") + 6,
        st.indexOf("9F17") + 8));
}

// send PIN to ICC
public boolean verifyPIN(String pin)
{
        try{
        System.out.println("APPLYING VERIFY COMMAND");
        //00 20 00 80 08

        String command = "00 20 00 80 08 24" + pin;
        while (command.length() < 26) //14 + 12 = 26
                command += "F";
        System.out.println(command);

        capdu = new ApduCmd(command);
        rapdu = se[0].execute(capdu);
        System.out.println(capdu);
        if (capdu.toString().substring(5, 10).equals("90 00"))
                return true;
        } catch (CardException e)
        {
                System.out.println("Error when sending PIN to ICC");
        }
        return false;
}

// prepare ICC for PIN verification
public static void PINProcess()
{
        try {
                f = SessionFactory.getInstance();
                se = f.createSessions();
                Atr atr = se[0].open();

                capdu = new ApduCmd("00 A4 04 00 0E 31 50 41 59
                2E 53 59 53 2E 44 44 46 30 31");
                rapdu = se[0].execute(capdu);

                capdu = new ApduCmd("00 B2 01 0C 00");
                rapdu = se[0].execute(capdu);

                String size =
                NumUtil.hex2String(rapdu.getStatusWord().getSw2())
                ;

                capdu = new ApduCmd("00 B2 01 0C" + size);
```

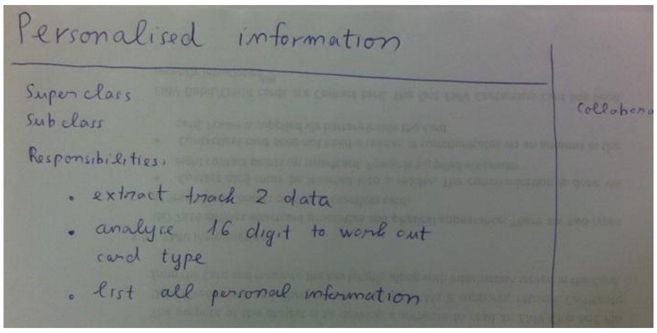
```
                    rapdu = se[0].execute(capdu);

                    getAID(rapdu.toString());

                    capdu = new ApduCmd("00 A4 04 00 07" + AID);
                    rapdu = se[0].execute(capdu);

                    capdu = new ApduCmd("80 A8 00 00 02 83 00");
                    rapdu = se[0].execute(capdu);

            } catch (CardException e)
            {
                    System.out.println("Error when preparing PIN
                    verification!");
            }
    }

    public static void main(String[] args)
    {
            new EMVPINVerification();
    }

}
```

Appendix C: CRC cards


Card input

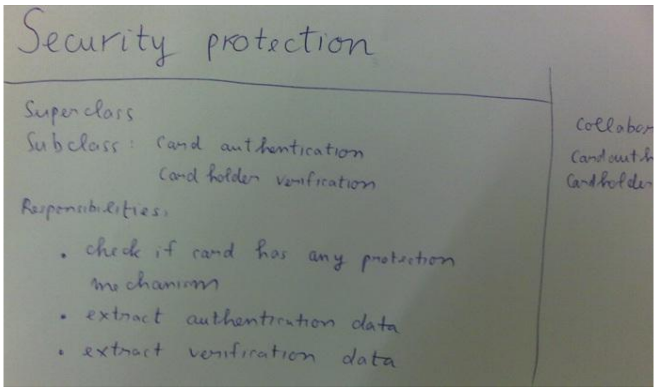
Super class:
Sub classes:
Responsibilities:
- creates new session for each inserted smart card
- reset smart card
- analyse ATR
- prepare connection

Collaborator


Personalised information

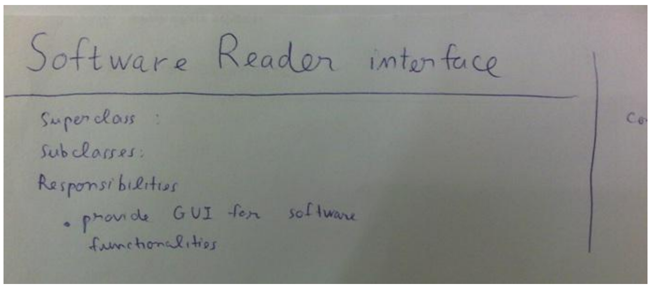
Super class
Sub class
Responsibilities:
- extract track 2 data
- analyse 16 digit to work out card type
- list all personal information

Collabora


Security protection

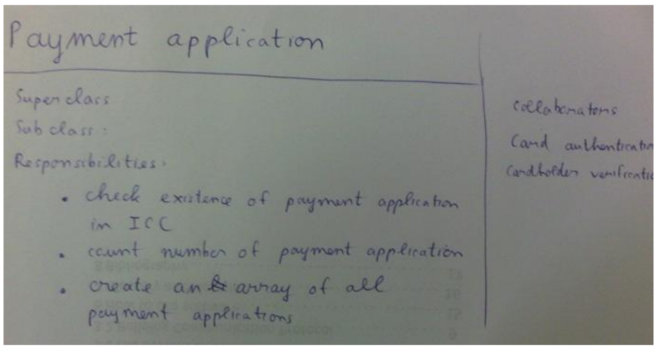
Super class
Subclass: Card authentication
Card holder verification
Responsibilities:
- check if card has any protection mechanism
- extract authentication data
- extract verification data

Collabor
Card auth
Cardholder


Software Reader interface

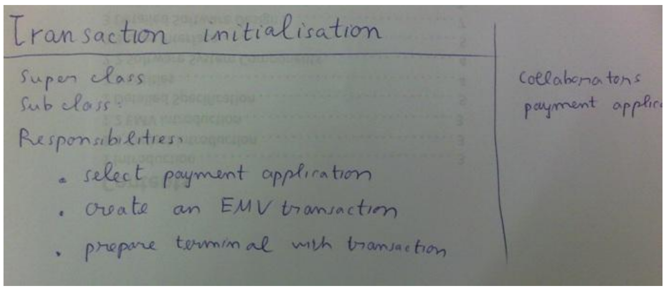
Super class :
Subclasses:
Responsibilities
- provide GUI for software functionalities

Co


Payment application

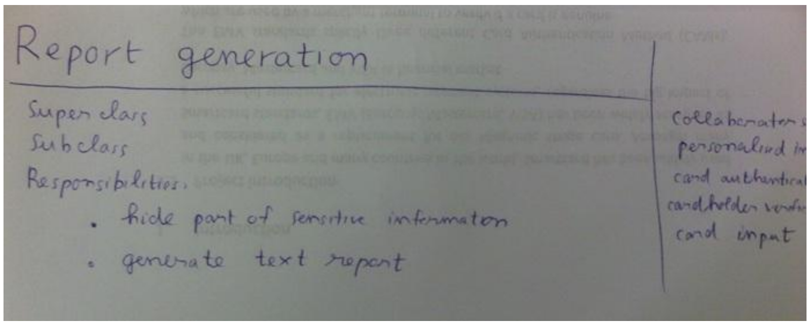
Super class
Sub class :
Responsibilities:
- check existence of payment application in ICC
- count number of payment application
- create an ~~&~~ array of all payment applications

Collaborators
Card authenticatio
Cardholder verificati


Transaction initialisation

Super class
Sub class:
Responsibilities:
- select payment application
- create an EMV transaction
- prepare terminal with transaction

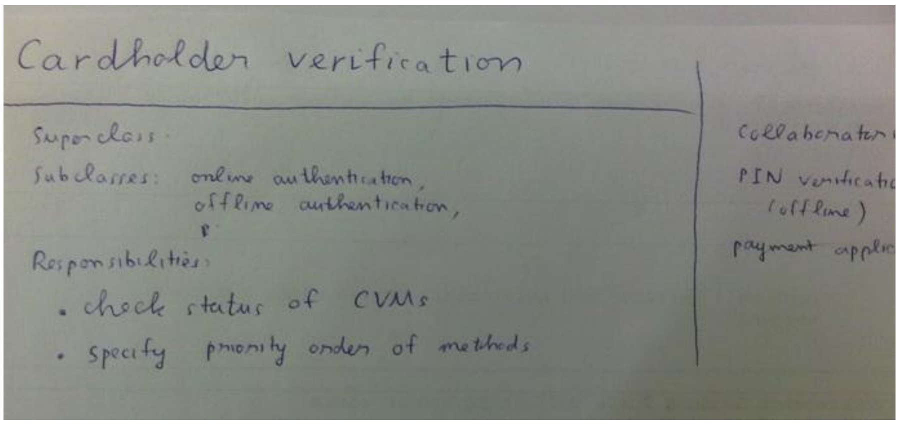
Collaborators
payment applic


Report generation

Super class
Subclass
Responsibilities:
- hide part of sensitive information
- generate text report

Collaborators
personalised in
card authentica
cardholder verifi
card input

163



Cardholder verification

Superclass:

Subclasses: online authentication,
offline authentication,

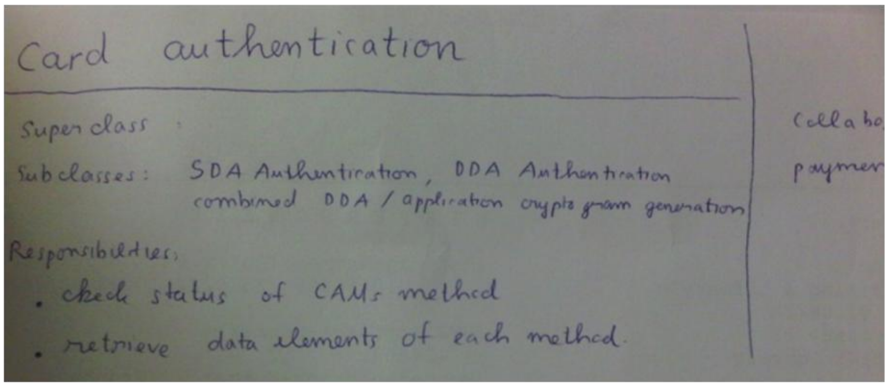
Responsibilities:

- check status of CVMs
- specify priority order of methods

Collaborator

PIN verificatio
(offline)

payment applic



Card authentication

Superclass :

Subclasses: SDA Authentication, DDA Authentication
combined DDA / application cryptogram generation

Responsibilities:

- check status of CAMs method
- retrieve data elements of each method.

Collabo

paymen

References and Bibliography

[1] EMVCo, EMV 2008 Integrated Circuit Card Specifications for Payment Systems, *Book 1 – Application Independent ICC to Terminal Interface Requirements*, Version 4.2, June 2008, http://www.emvco.com/specifications.aspx?id=155

[2] EMVCo, EMV 2008 Integrated Circuit Card Specifications for Payment Systems, *Book 2 – Security and Key Management*, Version 4.2, June 2008, http://www.emvco.com/specifications.aspx?id=155

[3] EMVCo, EMV 2008 Integrated Circuit Card Specifications for Payment Systems, *Book 3 – Application Specification*, Version 4.2, June 2008, http://www.emvco.com/specifications.aspx?id=155

[4] EMVCo, EMV 2008 Integrated Circuit Card Specifications for Payment Systems, *Book 4 – Cardholder, Attendant, and Acquirer Interface Requirements*, Version 4.2, June 2008, http://www.emvco.com/specifications.aspx?id=155

[5] Mitchell, C., *IY2760/CS3760 Case Study 1: EMV*, Version 1, ISG Group, Royal Holloway, http://www.isg.rhul.ac.uk/~cjm/IY2760/Case_study_1_EMV_0910_v1.pdf

[6] Radu, C., *Implementing Electronic Card Payment Systems*, Artech House, 2002

[7] Chen, Z., *A Programmer's Guide to Java Smart Cards (Java Series)*, Prentice Hall, 2000

[8] Buetler, I., Smart Card APDU Analysis, viewed 25-November-2009, https://www.blackhat.com/presentations/bh-usa-08/Buetler/BH_US_08_Buetler_SmartCard_APDU_Analysis_V1_0_2.pdf

[9] Rousseau, L., smartcard_list.txt, viewed 10-September-2009, http://ludovic.rousseau.free.fr/softwares/pcsc-tools/smartcard_list.txt

[10] Sheong, S., *Getting information from an EMV Chip card with Java*, viewed 01-October-2009, http://blog.saush.com/2006/09/08/getting-information-from-an-emv-chip-card/

[11] Microsoft, *Smartcard Resource Manager*, http://msdn.microsoft.com/en-us/library/aa380148(VS.85).aspx

[12] Mayes, K., *Smart Cards and RFIDs in the Modern World*, ISG Group, Royal Holloway, http://www.isg.rhul.ac.uk/~cjm/IY2760/Smart%20card%20intro%20051109%20handouts.pdf

[13] EMVCo, *EMV Card Personalization Specification*, Version 1.1, July 2007, http://www.emvco.com/specifications.aspx?id=20

[14] MasterCard Worldwide, Card Personalization Validation Guide, January 2009, http://www.paypass.com/pdf/public_documents/CPV_Manual_Jan_2009.pdf

[15] Logic group, *Introduction to EMV Chip & PIN*, http://www.the-logic-group.com/Downloads/Intro-to-EMV.pdf

[16] Photolia, *ATM Keypad picture*, viewed 14-January-2010, http://static-p3.fotolia.com/jpg/00/00/04/50/400_F_45001_9lewF3qI6OLmDoylvj1TQrA6lZVSOq.jpg

[17] Wikipedia, *Bank card number*, viewed 21-January-2010, http://en.wikipedia.org/wiki/Credit_card_numbers