FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Payment Terminal Emulator

Liudmila Kisialiova

Mestrado em Engenharia de Software

Supervisor: António Miguel Pontes Pimenta Monteiro

June 24, 2020

Payment Terminal Emulator

Liudmila Kisialiova

Mestrado em Engenharia de Software

June 24, 2020

Abstract

Nowadays, cash payments are becoming less popular [28]. The majority prefers to use a payment card everywhere instead of carrying large quantities of coins and banknotes, which can be insecure. However, few understand the complicated process that stands behind the habitual inserting that card into the PoS (Point-of-Sale) terminal, entering the PIN, and taking the receipt.

Inside the PoS terminal take place the steps of Card Application, Reading Card Settings, Cardholder Verification, Risk Management, Action Analysis, Online or Offline Authentication, Pin Verification and other steps. To reduce skimming fraud and to unite all payment processes across the world for different terminal and card types, the EMV specifications were written for Europe, and later for USA and other regions [19].

The initiative for EMV was taken by three companies – Europay, Mastercard an Visa – in the 1990s with the view to replace magnetic stripe cards with smart cards [7]. The first specification was released in 1996. Now the EMV standards are maintained by EMVCo company, which includes six companies – American Express, Discover, JCB, Mastercard, UnionPay, and Visa [20].

FinTech organizations (later in this paper also called Payment Systems) develop full payment solutions for the acquirer banks and merchants, so that they can introduce PoS terminal payments in their businesses.

In order to facilitate development and testing of the terminal software and reduce the cost of buying expensive hardware for development environment, the FinTech companies aim to develop terminal emulator applications. The payment terminal emulator is an application to emulate the full payment transaction process. The emulator helps to test integration of new acquirer's interfaces into the payment system based on EMV protocol.

Many FinTech companies already have their own terminal emulators, however, existing solutions are proprietary and its implementation is not described widely in research papers. One of the Portuguese FinTech companies needs a standalone application to cover most popular terminal payment flows to get advantage over rivals in the competitive FinTech market. The 3C Emulator application for smart card transactions is under development and its implementation solutions are described in this paper. The emulator application should satisfy the EMV ICC specifications [13, 14, 15, 16].

To achieve the objective, a standalone software application was developed. 3C Emulator includes emulation of the main payment transactions, *e.g.* Sale, Reverse, Pre-authorisation, Top-up and other transactions. The transaction then is processed in the company's servers, which, in their term, contact the real merchants networks to process the transaction online. The application process the steps in accordance with EMV specifications.

The application was written in Java language, and uses a SQLite database [47] to store temporary application data, *e.g.* list of default cards. The graphical interface is written with Java AWT technology. Since 3C Emulator is a standalone application, no server is needed. However, the application communicates with the Payment System and merchants' servers to process the payments. The communication uses HTTPS and TLS protocols. For message encryption the RSA algorithm is used.

The application was tested with unit testing (using JUnit library [26]) and integration testing. It was used by a few developers in 3C TechLab to verify the application workflow. Unfortunately, due to the COVID-19 situation in 2020 and the lack of support time in the company, the full integration of the solution into the certification process put on hold without a scheduled time frame.

In this work we consider communication protocol between a smart card and a PoS terminal based on EMV international standard, together with its known vulnerabilities. Also we provide the development and test details, together with the application architecture. Finally, we consider the work limitations and possible future work.

Keywords: EMV, PoS terminal, Smart Card, Emulator, Bank payment

Resumo

Atualmente, os pagamentos em dinheiro estão a tornar-se menos populares [28]. A maioria das pessoas prefere usar o cartão de débito ou crédito em qualquer lugar, em vez de trazer grandes quantidades de notas e moedas, o que pode ser inseguro. No entanto, poucas pessoas conhecem a complexidade que se encontra por detrás da inserção do cartão no terminal PoS (ponto de venda), introdução do PIN e recolha do recibo.

Dentro do terminal PoS, existem as etapas de Card Application, Reading Card Settings, Cardholder Verification, Risk Management, Action Analysis, Online or Offline Authentication, Pin Verification, entre outras. Para reduzir a fraude e unir todos os processos de pagamento em todo o mundo para diferentes tipos de terminais e cartões, criaram-se as especificações EMV para a Europa e, posteriormente, para os EUA e outras áreas do mundo [19].

A iniciativa de EMV foi tomada por três empresas - Europay, Mastercard e Visa - na década de 1990, com o objetivo de substituir os cartões de banda magnética por cartões inteligentes [7]. A primeira especificação foi lançada em 1996. Agora, os padrões EMV são mantidos pela empresa EMVCo, que inclui as seis empresas - American Express, Discover, JCB, Mastercard, UnionPay e Visa [20].

As organizações conecidas por FinTech (posteriormente neste documento, também chamadas de Payment Systems) desenvolvem soluções completas de pagamento para os bancos e comerciantes adquirentes, para que possam introduzir os pagamentos via terminal PoS nos seus negócios.

Para facilitar o desenvolvimento e o teste do software dos terminais e reduzir o custo da compra de hardware caro para o ambiente de desenvolvimento, as empresas FinTech têm como objetivo desenvolver apicações que atuem como emuladores de terminal. O emulator de terminal de pagamento é uma aplicação para reproduzir o processo completo de uma transação de pagamento. O emulador ajuda a testar a integração da nova interface do adquirente do sistema de pagamento com base no protocolo EMV.

Muitas empresas FinTech já possuem os seus próprios emuladores de terminal. No entanto, as soluções existentes são proprietárias e a sua implementação não é descrita amplamente na literatura. Este trabalho teve origem na necessidade de uma das empresas portuguesas de FinTech precisa de uma aplicação independente para cobrir os fluxos mais populares de pagamentos de terminais e obter vantagem sobre a concorrência no competitivo mercado FinTech. O emulator de terminal de pagamento para transações com cartão inteligente foi desenvolvido sendo as suas soluções de implementação descritas neste documento. A aplicação deste emulador deve atender às especificações EMV ICC [13, 14, 15, 16].

De forma a atingir o objetivo, foi desenvolvida uma aplicação de software independente. A aplicação 3C Emulator inclui a emulação das transações de pagamento principais, tais como Sale, Reverse, Pre-authorisation, Top-up e outras transações. Cada transação é processada nos servidores da empresa, que, por seu turno, entram em contato com as redes reais dos comerciantes, para processarem as transações online. A aplicação processa todas as etapas necessárias de acordo com as especificações EMV.

A aplicação foi escrita em linguagem Java e usa a base de dados SQLite [47] para armazenar dados temporários da aplicação, como por exemplo a lista de cartões padrões. A interface gráfica foi escrita com a tecnologia Java AWT. Como o 3C Emulator é uma aplicação independente, nenhum servidor é necessário; porém, a aplicação comunica com os servidores do Payment System e dos comerciantes para processar os pagamentos. A comunicação usa os protocolos HTTPS e TLS. O algoritmo RSA foi usado para encriptar mensagens.

A aplicação foi testada com unit tests (usando a biblioteca JUnit [26]) e teste de integração (foi usado por alguns desenvolvedores na 3C TechLab para verificar o fluxo de trabalho da aplicação). Infelizmente, devido à situação do COVID-19 em 2020 e à falta do tempo de suporte na empresa, a total integração da solução ao processo de certificação está suspensa indefinidamente.

Nesta pesquisa, consideramos o protocolo de comunicação entre um cartão inteligente e um terminal PoS baseado no padrão internacional EMV, juntamente com as suas vulnerabilidades conhecidas. Também fornecemos os detalhes de desenvolvimento e teste, juntamente com a arquitetura da aplicação e do sistema geral. Por último, consideramos as limitações do trabalho e possíveis melhorias futuras.

Keywords: EMV, PoS terminal, Smart Card, Emulator, Bank payment

Acknowledgements

First of all, I would like to thank António Miguel Monteiro (Professor in Security in Software Engineering in Engineering Faculty University of Porto and the supervisor of this dissertation work) for his helpful comments, suggestions and academic guidance.

Secondly, I would like to thank development teams in 3C TechLab for useful ideas and discussion about essentials in the EMV specifications. And especially thanks to Ricardo Anacleto (Project Manager of the Terminal Development team), who is the dissertation sub-supervisor.

I would like to say special thanks to Gustavo Camacho and his family for giving me the last motivational wave to finish the second year of my master degree. Your family always gives me the best example of motivation, persistence, confidence in own wishes and just being a good person.

Moreover, I would like to thank my ex-husband Ivan Kisialiou and ex-family-in-law for supporting me in my wish to try studying in another country. You always believed in me and gave valuable advice. Although the studying abroad broke our family, I feel great gratitude for all previous years we lived together and being supported remotely during my studying.

And I am deeply grateful to my family (parents, my brother Alexey and my sister-in-law) for always supporting me in the most difficult moments. Thank you for hours and hours of phone calls. The feeling, that you are still close to me, dramatically helped me in this emigration journey. From the bottom of my heart, thank you for supporting me unconditionally.

To all these people, staff at FEUP and particularly those involved in my masters degree: my sincerest thanks.

Liudmila Kisialiova

"Pain is inevitable, suffering is optional."

Haruki Murakami

Contents

List of Figures

List of Tables

Abbreviations

AAC	Application Authentication Cryptogram
ADF	Application Definition File
AEF	Application Elementary File
AID	Application Identifier
APDU	Application Protocol Data Unit
ARQC	Authorisation Request Cryptogram
ATM	Automated teller machines
ATR	Answer To Reset
AWT	Abstract Window Toolkit
BNKSM	Bank Simulator service
C-ADPU	Command APDU
CDA	Combined DDA/Application Cryptogram Generation
CDOL	Card Risk Management Data Object List
CLA	Class Byte of the Command Message
CLK	Clock
CNP	Card Not Presented
CPU	Central Processing Unit
CTP	Collection Tree Protocol
CVV	Card Verification Value
DB	Database
DCC	Dynamic Currency Conversion
DDA	Dynamic Data Authentication
DDOL	Dynamic Data Authentication Data Object List
DOL	Data Object List
DOM	Document Object Model
EEPROM	Electrically Erasable Programmable Read Only Memory
EMV	EuropayMastercard Visa
IntegraFE	Itegra Front End
FinTech	Financial Technology
Gnd	Ground
H	Maximum Voltage Value
HSM	Hardware Security Module
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output
ICC	Integrated Circuit Card
IFSI	Information Field Size Integer
INS	Instruction Byte of Command Message
Java AWT	Java Abstract Window Toolkit

L	Minimum Voltage Value
Lc	Exact Length of Data Sent by the TAL in a Case 3 or 4 Command
Le	Maximum Length of Data Expected by the TAL in Response to a Case 2 or 4 Command
MF	Master File
MID	Merchant Identifier
MTI	Message Type Indicator
NAS	New Authorisation Service
NFC	Near-field Communication
NFC SWP	Near Field Communication Single-Wire Protocol
NPU	Network Processing Unit
P1	Parameter 1
P2	Parameter 2
P3	Parameter 3
PAN	Primary Account Number
PC	Personal Computer
PDOL	Processing Options Data Object List
PLA	Poly Lactic Acid
PoS	Point-of-Sale
PVC	Poly Vinyl Chloride
QA	Quality Assurance
R-ADPU	Response APDU
RAM	Random Access Memory
RFU	Reserved for Future Use
ROM	Read Only Memory
RRN	Retrieval Reference Number
RSA	Rivest–Shamir–Adleman
RST	Reset
SDA	Static Data Authentication
SFI	Short File Identifier
SSAD	Signed Static Application Data
SSL	Secure Sockets Layer
STAN	System Trace Audit Number
SW1	Status Byte One
SW2	Status Byte Two
T = 0	Character-oriented asynchronous half duplex transmission protocol
T = 1	Block-oriented asynchronous half duplex transmission protocol
TC	Transaction Certificate
TDOL	Transaction Certificate Data Object List
TID	Terminal Identifier
TLS	Transport Layer Security
TLV	Tag Length Value
TPDU	Transport Protocol Data Unit
UAT	User Acceptance Testing
VPP	Programming Voltage
WI	Waiting Time Integer
XML	Extensible Markup Language

Chapter 1

Introduction

Point of Sale terminal (PoS terminal) is an electronic device used to process payment with credit or debits cards. The newest terminal supports different payment protocols, contactless or not, with or without physical presence of the card. In the majority of cases, payment terminals are used in retail locations, such as shops, hotels or restaurants. And the usage of non-cash payment helps merchants to facilitate accounting process, reduce money change inconvenience, and attract the clients who prefer non-cash payments.

1.1 Context

The main operation of the payment terminal is transferring of the funds from the customer bank account to the seller's one by network, along with other operations, such as checking the funds amount of the card, changing PIN code, printing a receipt, collecting payment reports and so on. Since the transferring of the funds directly deals with real money, the terminals usually become the attack goal of fraud and hacking. The PoS terminals combine hardware and software, and it is extremely important that produced device and developed software solution to be as much secure and safe as possible to prevent money crime.

There are some security specifications and guidance, that explain how to develop terminal hardware and software. These documents aim not only guide developers, but also establish security minimum for terminals to be safe, unite the terminal all over the word to use the same communication protocols, and so on. One of the most used in Payment Systems specification belongs to EMVCo company [20]. It became that much popular, that in some literature the smart (ICC) cards are called EMV cards, the terminals that support these cards - EMV terminals. This dissertation work only considers EMV specifications.

The development of full terminal solution, that is ready to be introduced into retail business, is complicated, multistage and time-consuming process. Small business, like shops, hotels, restaurants, are not capable to develop the solution from the scratch. Even sound banks with strong IT department are not able to produce terminals with their own software installed. Therefore, in the FinTech market the Payment Systems exist. In Portugal, 3C TechLab [49] is one of the examples of such companies. The main target of the company is to develop ready-to-use solution: PoS terminals, Apple Pay, Google Pay, PayPal, online payments with 3-D Secure technology and other products.

> "3C TechLab is a newly created FinTech organization initiated by 3C Payment to act as an innovation incubator for both payment and non-payment solutions. Based in the vibrant city of Porto and with strategic investment from 3C Payment, 3C TechLab provides technical nearshore services, complimentary value add products and increases the development capacity and speed to market for 3C Payment products. With specialist skills in both Apple and Android applications together with expert knowledge in EFTPOS EMV software, 3C TechLab is able to ensure quality assurance whilst maintaining an agile start up mentality within its workforce." [49]

3C TechLab together with 3C Payment [39] provide the solutions to great number of banks and retail companies all over the world. The banks and retail companies have their own payment networks, so the terminals should be integrated properly into existing clients' networks. And usually this is not a trivial process due to the different format of messages and network connection, the ways of processing currency exchange, the variety of card the terminal should support, and *etc.*. The final solution should be thoroughly checked against satisfaction of the client requirements and tested well in accordance with EMV specification.

1.2 Problem

As was mentioned above, one of the goals of Payment Systems, like 3C TechLab, is to provide the clients with terminal solutions, that are stable, secure and comprehensive in term of possible operations.

The best way for the company to reduce the cost of development of such a solution is invest in testing and software certification procedures. All company's software is thoroughly verified before shipping to the clients. After the software is developed and installed into terminal device, the certification team checks following requirements:

1. The terminal supports all card type that the clients requested (*e.g.* Visa, Mastercard, Diners, American Express);
2. The terminal process the complex payment flow with all mandatory and optional payment steps (*e.g.* Data Authentication, Cardholder Verification, Online or Offline Issuer Authentication);
3. The software supports different models of terminals that the client requested. This include the cost of purchase and shipping of terminals for development and testing.

Besides the certification of new developed software, 3C TechLab has to ensure in stability of existing terminal solutions. It is extremely difficult to avoid introducing of error while making changes in existing terminal software.

Finally, so far the certification process are not able to cover all possible flows and sometimes the certification team resorts to manual testing. This work requires much time and, moreover, can be error-prone.

3C TechLab needs a standalone software of terminal emulator to reduce the cost of development and to improve certification process. The software emulates the payment flow faster and with more accuracy, it can be applied to the number of the clients, support different types of cards and terminals.

1.3 Motivation and Objectives

The main benefit of introduction the payment terminal emulator into the 3C TechLab development process is the reduction of the cost of the development and certification. The standalone application can be used in different phases of development independently by the variety of developers and QA engineers. The improvement of the quality of terminal software can lead the company to get higher position in the highly competitive FinTech market.

There are existing terminal emulator solutions, but most of them are proprietary and can not be applicable in all payment flow that the 3C TechLab software provides. Therefore, the company has an objective to create their own emulator software.

1.4 Work Structure

This document begins by presenting a literature review and comprehensive research on EMV specification in Chapter 2. Then, in Chapter 3 a detailed list of requirements and expected system behavior are presented. In Chapters 4 and 5 we uncover the architectural and development details of the implemented solution, where the work unfolds. Chapter 6 provides the application verification result using requirements list as an acceptance test. Finally, Chapter 7 presents the conclusions of the developed work as well as the identification of possible future work.

Chapter 2

Payment Terminal

This chapter presents a review on the State-of-the-Art of the Payment Terminal Emulator. We explain the process of the production of smart cards, business communication between anchor institutions and main payment flows, focusing on the variety of the smart cards, terminal models, several security mechanisms, including their advantages and disadvantages. Finally, the chapter provides known security vulnerabilities in EMV terminal usage.

2.1 Introduction

Nowadays, card payments become more popular [28]. However, few understand the complicated process that stands behind the habitual inserting that card into the payment terminal.

If one opens his pocket and looks at the card they have, probably they can find cards with magnetic stripe only, or blank cards without any signs, but which are the electronic keys with the NFC chip inside, or the cards that have the silver or gold chips. Moreover, the card can be stored in virtual way in smart-phones, that process the transactions in contactless way. We provide the review of possible payment means and its production.

Then, several institutions are involved in the process of the fund transfer from one bank account to another. For instance, there are (1)(2) two banks, (3) the company that produce smart cards, (4) the company that provides payment scheme, (5) the payment system company and *etc.* If the PoS terminal is used in retail location, (6) merchant is also involved. The payment algorithm with all possible anchors is described 2.1.

Finally, EMV specification, that is represented in a set of books, is used to guide development of PoS terminals and smart cards. The specification has become widely known among payment systems, it spread from Europe continent to the whole world. And nowadays it became that much popular, that in some literature the smart (ICC) cards are called EMV cards, the terminals that support these cards - EMV terminals. Following EMV specification, the payment terminal emulator was developed.

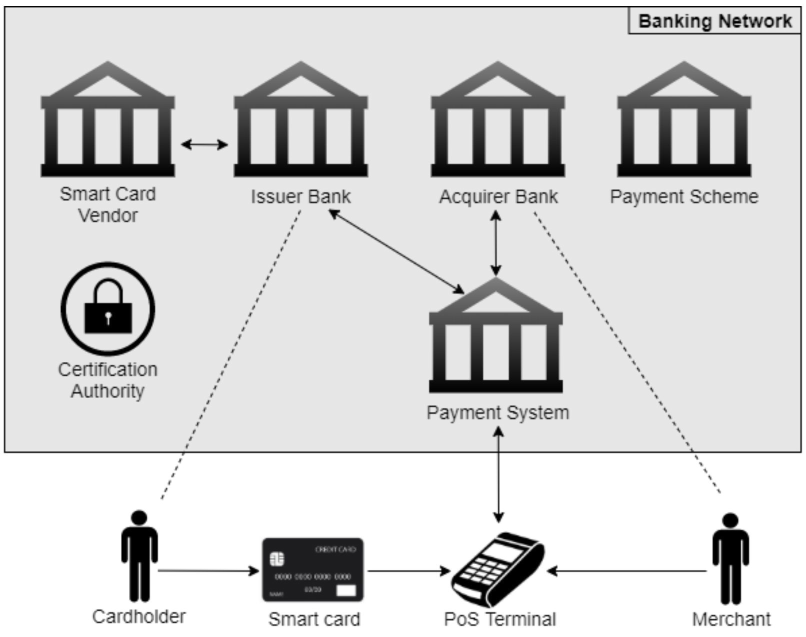


Figure 2.1: Payment algorithm with PoS terminal in respect of the EMV specification

2.2 Payment Process

2.2.1 From Merchant to Bank

Speaking about payment, one can consider different payment schemes and means:

- Paper instruments, *e.g.* cash (banknotes and coins) and cheques;
- Payment cards, *e.g.* debit, credit, pre-paid and e-purses (electronic money);
- Mobile devices working in remote schemes and contactless schemes and representing some smart card.
- CNP (Card Not Present) payment data for online transactions.

In this research we focus on payment mechanisms based on payment cards. The payment card can have different communication features, such as:

- Magnetic stripe (supports only online transactions);
- ICC (Integrated Circuit Card) chip (supports both online and offline transactions, for instance, payments in planes during flight or in trains in tunnels);
- NFC antenna (provides additional interface to ICC chip).

Figure 2.1 illustrates the actors that participate in executing a secure purchase transaction and clarifies the business algorithm of the PoS terminal transaction using a smart card.

Smart Card Vendor – is a manufacturer of smart cards. When a client requests a new payment card, the Issuer Bank sends a request to the Smart Card Vendor to produce both card hardware and install software, namely setting card data, card key pair, issuer data, issuer public keys, expiration date, and so on.

Issuer Bank – is the bank that provides the client with a card (the client obtains the status of a cardholder), contracts with the clients, collects client's personal data to satisfy security purposes, and so on [30].

Acquirer Bank – is the bank that contracts with the merchants, collect personal and business data, provides the merchants with a terminal.

Payment Scheme (*e.g.* Visa, Mastercard, American Express, *etc.*) – is a network guaranteeing the transfer of money. Moreover, it sets specific rules over manufacturing process, collecting private data of clients, and so on [44]. During the payment process, the Payment Scheme does not play any role, however, it sets regulations for ICC data elements, the terminal messages format, and other static rules.

Payment System (*e.g.* 3C Payment, Amazon Pay, Atom, PayPal Braintree, PayPal Venmo, PayPal Xoom, SIX Payment, Qiwi, WebMoney, Yandex.Money, *etc.*) – is a complete payment solution provided by FinTech company. In PoS payments scheme, the Acquirer Bank contracts the Payment System to provide terminals ready to install in merchants sell points. The majority of Payment Systems uses external companies, that produce terminal hardware, for example Worldline, Verifone, Ingenico, PAX Technology, CyberNet, *etc.* [42]. Then the PoS terminal is configured by the Payment System provider in accordance with the Acquirer Bank environment.

Merchant – the person or a group of people willing to integrate terminal payment in their business. The most popular example is when merchants are represented by medium size selling businesses, that trade in goods produced by other people.

Cardholder – the client of the bank, who the bank produces and supplies a payment card to.

PoS Terminal – is a device which interacts with payment cards to make electronic funds transfers. While installing software, the Payment System configures the address of a Certification Authority, so that the terminal can upload a signed issuer public keys from it. Since upload is done automatically online, there is no need to physically visit merchant points to re-install terminal software.

Smart Card – a payment mean. In this dissertation work we consider ICC cards, however it can be an NFC analog of the card, for example, smartphone with card data attached.

The common terminal-card communication includes the following steps [48] (more details can be found in subsection *B. EMV Transaction Flow*):

1. An attended inputs transaction amount, specifies currency if needed, and set other transaction settings;

2. When the card is applied, Data Authentication is processed (online or offline) to check the card genuineness;

3. The cardholder verifies identity with PIN (online or offline PIN verification), signature or other verification method;
4. The processing of the transaction can be online (the terminal sends requests to Acquirer and if needed to Issuer) or offline;
5. If the transaction is approved, the terminal shows an “Approved” message. Otherwise, the terminal displays an error message;
6. The card should be deactivated.

Although the introduction of a PoS terminal into the merchant business has a distinct advantage, not all merchants are eager to start acceptance of non-cash payments [33]. The reasons are:

1. The more card types to support – the more expensive the terminal. For small businesses, the cost of supporting rare card types can exceed the possible income.
2. Network cost. Once the merchant decides to extend their payment methods with PoS terminals, they should provide a stable and reliable Internet connection, so that terminals can process transactions online or partially online.
3. Installation effort. Sometimes the benefit of introducing terminals for small businesses is too small comparing to the effort of its configuration. Namely, reaching agreement with an Acquirer bank, sharing business details with third parties, additional security risks, and other actions, can be not worthwhile.

However, for the medium size businesses the benefits of non-cash payments are difficult to underestimate.

2.2.2 Smart Cards and ICC

The smart card – is a plastic card with chip integrated. Also a smart card is called an Integrated Circuit Card (ICC). A smart card:

1. has a unique identifier (smart token);
2. has a microprocessor, that provides business logic to support electronic transaction;
3. has a strong security mechanism;
4. has no internal power supply;
5. is extremely difficult to forge or duplicate;
6. can be run only using external energy resource (card reader with power source);

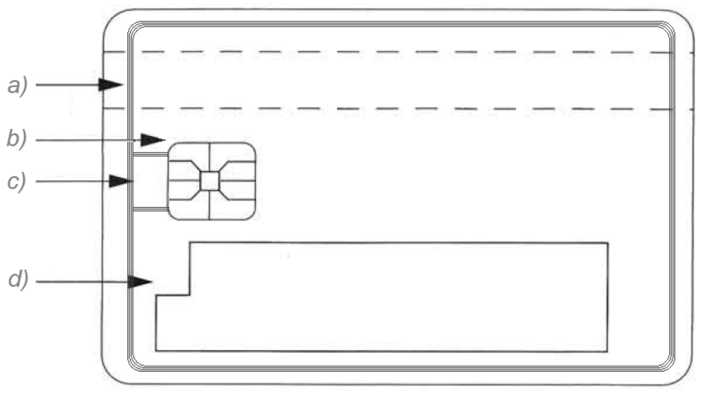


Figure 2.2: Physical appearance of the payment card: a) magnetic stripe (back of the card); b) contacts of chip; c) NFC antenna; d) embossing area

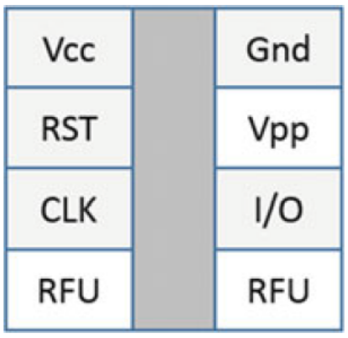
7. has three types of semiconductor memory: Read Only Memory (ROM), Random Access Memory (RAM) and Electrically Erasable Programmable Read Only Memory (EEPROM).

A smart card can be used as not only a payment mean, but also as a mobile telecommunications instrument (SIM cards), identity cards or passports, transport cards (e-tickets), IT access control (e-key), timing control cards (SPORTident cards), and other means. Basically, the smart card can provide such functions as:

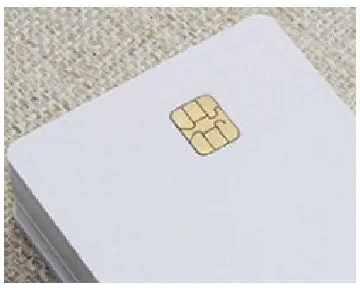
- Identification (smart card identifies person, for example ID card)
- Authentication (for instance, smart card allows access only to particular person and acts as a key)
- Storing data with possibility to modify it (for instance, timekeeping system)
- Authorisation (the usage of smart card by a defined person to access the system or process a transaction).

Most popular payment cards, that nowadays issuer banks produce, have several interfaces to operate in a payment (Figure 2.2):

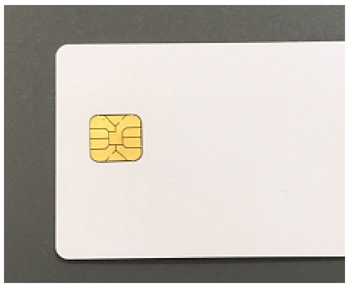
- Embossed card number and CVV (card verification value) are used for online payments;
- Signature of cardholder on the back side of the card is used for the authentication. For risk management sometimes the issuer or the acquirer requires signature verification in the moment of a transaction.
- Magnetic Stripe is used in swipe communication. The usage of magnetic stripe is considered less secure in comparison with ICC and NFC communications [34]. Some cases of vulnerabilities are described in subsection *D. Fraud examples*.



(a) Scheme of ICC chip contacts



(b) Six-contact chip



(c) Eight-contact chip

Figure 2.3: Contacts on ICC chip

- NFC microchip and several loops of wire installed inside the plastic part of the card. Some card readers support contactless communication. The communication and message exchange is almost the same as for the ICC communication. NFC interface replaces physical contact between card and card reader, but remote reader interacts with the same chip memory.
- ICC chip is used by inserting the card into card reader and establishing physical connection with its contacts. The chip is hidden behind the contacts and has a surface less than 50% of the contacts surface.

Chip manufactures and banks agreed to keep the size of chip surface relatively small, due to several reasons:

- cheaper to produce, because consumption of material is smaller;
- physically more solid, and the chip suffers less from bending pressure;
- card reader has to provide less power to activate the chip;
- more secure, since small dimension complicates re-engineering of the chip

Since the plastic base of the card has no style restriction, manufactures provide unusual features: Translucent, Perspective, Coloured Edge, Tactile, Nacre, Fragrance, Metal Effect, Durocore, Poly Vinyl Chloride (PVC) recycled, Poly Lactic Acid (PLA)—a PVC-free card, Shapes, CDplus

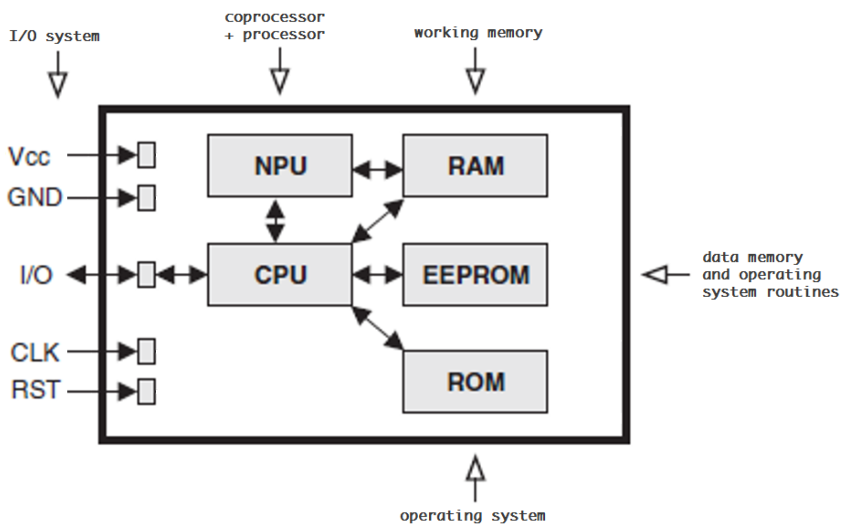


Figure 2.4: Smart card architecture

Connectcard [34]. However, the chip structure shall strictly follow international requirements. ICC can have a maximum o eight contacts (Figure 2.3) and all surface of the chip is gold or silver coloured. The minimum number of chip contacts is five:

- Vcc is supply voltage
- RST is a reset line
- CLK provides clock pulse
- Gnd is level of minimum voltage
- I/O line is used for communication and to pass messages between ICC and reader

Additional contacts are:

- Vpp is programming voltage. The contact is used in old-style EEPROM memories (electrically erasable programmable read-only memory) with possibility to re-program the chip.
- two RFU (Reserved for Future Use) contacts can be used in special cases. Nowadays, most cards and readers already support the usage of this contacts, for example, high-speed USB interface, and NFC SWP (Near Field Communication Single-Wire Protocol), and others.

As illustrated in Figure 2.4, the chip consists of the following components:

1. I/O system – chip interface comprising contacts on the surface of the card.
2. Central Processing Unit (CPU) and Network Processing Unit (NPU). Due to presence of CPU the card was named smart [34].

3. Three types of memory:
   (a) ROM (Read Only Memory) – this semiconductor memory is typically used for storing the operating system programs, such as the program needed to start the smart card when its power is turned on. ROM is also known as operating system memory.
   (b) RAM (Random Access Memory) – the fastest type of memory, used only for temporary storage. The information in RAM is accessed only in a certain (relatively small) amount of time. RAM is also known as working memory.
   (c) EEPROM (Electrically Erasable Programmable Read Only Memory) – electrically erasable and modifiable memory. It is used for storing programs and data that periodically needs modification. Due to security reasons, modern smart cards usually do not have EEPROM, but can replace it by other kind of non-volatile flash memory.

Card and card reader communicate with character-oriented (called “T = 0”) or block-oriented (called “T = 1”) asynchronous half duplex transmission protocol. The protocol “T = 0” is mostly used in telecommunications (SIM cards), and supports sending one byte at a time. However, the protocol “T = 1” sends data in blocks of characters. This protocol is used in payment cards.

2.2.3 PoS Terminals

First of all, there are the variety of electronic devices that used to process payments [34], for instance:

- Automated teller machines (ATMs)
- Branch terminals
- Cardholder-activated terminals
- Electronic cash registers
- Personal computers
- and other devices

However, in this dissertation work, we focus on PoS terminal payments.

A Point-of-Sale terminal is an electronic device used to process card payments at a merchant points. The PoS terminal functionality includes [34]:

1. Reading data from credit or debit card
2. Checking an account balance
3. Transfer the funds to another account
4. Process payment by bank or by merchant point

5. Recording the transaction and printing a receipt

Book 4 of EMV ICC specification [16] describes all terminal types. A terminal that follows the specification can have attended or unattended environment. In case of an attended terminal, the transaction is processed with the presence of both a cardholder and an attended (merchant agent or acquirer agent). And in contrast, unattended environment does not have "face-to-face" presence.

Some terminals support Voice Referrals. In case of an attended terminal the issuer can wish to verify the cardholder, then the terminal displays a "Call your bank" message, so that the cardholder has to confirm his transaction intention.

The terminals can use different type of communication:

- Offline-only. In this case there is no Internet connection, and there is no random transaction selection to process it online, since all transactions are processed offline. Once the terminal is connected to the Internet, all transactions are sent with the batch of data. The upload of transactions is called reconciliation.
- Online-only. In online-only terminal no Risk Management analysis and random transaction selection are used, since all transactions are verified and processed online. Such terminals require an Internet connection, and all Risk Management procedures are performed on the acquirer side.
- Offline with online capability. This type of communication is the most popular nowadays and join benefits from both online-only and offline-only. This type of terminals supports Risk Management and sends transactions data to Acquirer in bunches of packages to reduce processing time.

If the terminal has online capability, then several transaction steps can be processed online or offline:

- Data Authentication
- PIN Verification
- Transaction processing

Not all terminals have all hardware in one physical box. Card reader, magnetic stripe reader, ICC reader, and keyboard can be connected to the PoS terminal remotely. In this case it is important to provide stable communication, proper plug in and plug off actions to maintain integrity of the operation.

As it was described earlier, smart cards can have more then one communication interface (ICC chip, NFC antenna, magnetic stripe). The terminals should support different input capabilities in order to process transactions via the interface that the selected card has.

PoS terminals have the following input capabilities [16]: inserting ICC chip, swiping magnetic stripe, placing NFC antenna close, and manual entry of card data. In case of failed read of ICC and presence of magnetic stripe, the terminal can display a message with advice to proceed the operation via magnetic stripe.

The terminal hardware can include:

- keypad (numbers (0 - 9), alphabetic (A - Z) and special ('*', '.', '#') keys, commands ('Cancel', 'Enter', 'Clear'), functions ('F1', 'F2', 'Backspace', 'Escape') and others)
- PIN pad ('0' - '9', 'Cancel' and 'Enter' keys, key '5' in the center of the pad should have tactile identifier, usually represented with raised dot)
- Display (minimum 2 lines of 16 characters)
- Memory with erase-protection. Some data is static (terminal manufacturer data, acquirer and merchant data, terminal capability data), some is dynamic (system trace audit number STAN counter, date counter). Moreover, all sensitive data should be protected properly (card PANs (Primary Account Numbers) applied in the terminal, black lists, *etc.*)
- Independent clock mechanism – to verify certificate expiration dates.
- Printer
- Card readers (ICC reader, magnetic stripe reader, NFC reader)

The terminal should support at least one language – the local language of the terminal usage region. However, the characters, that the display should be able to show, are from the Latin alphabet plus common special characters (in accordance with ISO/IEC 8859 encoding standard). In the case of supporting several languages, the terminal reads the language preference, that is set by the issuer, from the smart card, and display messages to the user using this language. If the preferred language is not supported by the terminal, options are shown to the user for selecting another language from the list.

A terminal has a list of mandatory messages, each one with a hexadecimal identifier. In Table 2.1, one can find the list of 19 standard messages that can be displayed in the terminals.

In the higher level, there is a list of possible business transactions that the acquirer interface processes [37]. The terminal is configured in accordance to the transaction types that the acquirer expects. The list of all transaction types is described in ISO technical documentation [25], and most popular transaction types are presented in Table 2.2. This standard defines a message format and a communication flow so that different systems can exchange these transaction requests and responses.

Following ISO8583 standard, the message type indicator (MTI) is a four-digit numeric field which indicates the overall function of the message.

The first digit of the MTI indicates the ISO 8583 version in which the message is encoded:

- 0xxx – ISO8583:1987
- 1xxx – ISO8583:1993

Table 2.1: Standard terminal messages

Identifier	**Message**
’01’	*(AMOUNT)*
’02’	*(AMOUNT) OK?*
’03’	*APPROVED*
’04’	*CALL YOUR BANK*
’05’	*CANCEL OR ENTER*
’06’	*CARD ERROR*
’07’	*DECLINED*
’08’	*ENTER AMOUNT*
’09’	*ENTER PIN*
’0A’	*INCORRECT PIN*
’0B’	*INSERT CARD*
’0C’	*NOT ACCEPTED*
’0D’	*PIN OK*
’0E’	*PLEASE WAIT*
’0F’	*PROCESSING ERROR*
’10’	*REMOVE CARD*
’11’	*USE CHIP READER*
’12’	*USE MAG STRIPE*
’13’	*TRY AGAIN*

- 2xxx – ISO8583:2003
- 3xxx, 4xxx, 5xxx, 6xxx, 7xxx – Reserved by ISO
- 8xxx – National use
- 9xxx – Private use

The second digit specifies the overall purpose of the message:

- x1xx – Authorization message to check availability of funds. Some Acquirer banks process payment only based on authorisation message.

Table 2.2: Most popular transaction types from [25]

Code of Request	**Code of Response**	**Transaction type**
X100	X101	Authorization
X100	X101	Pre-authorization
X100	X101	Refund authorization
X100	X101	Top-up authorization
X200	X210	Sale
X220	X221	Completion
X420	X430	Reversal
X520	X521	Final reconciliation
X800	X810	Diagnostic message

- x2xx – Financial messages to process payment transaction and initiate the transferring of funds.
- x3xx – File actions message to collect statistics and perform audits.
- x4xx – Reversal messages, to return the funds.
  - Reversal (x4x0 or x4x1): Reverses previous authorization.
  - Chargeback (x4x2 or x4x3): Charges back a previously cleared financial message.
- x5xx – Reconciliation message to transmit settlement message.
- x6xx – Administrative message to transmit administrative advice, for example failure (rejection, application error and so on) message.
- x7xx – Fee collection messages
- x8xx – Network management message to maintain network connection, for example, logon, echo test, logoff, diagnostic message, secure key exchange and other messages.
- x0xx, x9xx – Reserved by ISO

The third digit specifies the message target and its method of sending:

- xx0x – Request from Acquirer to Issuer, can be approved or rejected
- xx1x – Issuer response to xx0x message
- xx2x – Advice that an action has taken place
- xx3x – Advice response to xx2x message
- xx4x – Notification that an event has taken place
- xx5x – Notification acknowledgement is response to xx4x message
- xx6x, xx7x – Instruction of ISO 8583:2003 and its acknowledgement
- xx8x, xx9x – Reserved for ISO use

Finally, the fourth digit specifies the location of the message source:

- xxx0 – Acquirer
- xxx1 – Acquirer repeat
- xxx2 – Issuer
- xxx3 – Issuer repeat

- xxx4 – Other
- xxx5 – Other repeat
- xxx6, xxx7, xxx8, xxx9 – Reserved for ISO use

The most frequent messages that Acquirer banks sends and expect in response are 0100 / 0110, 0200 / 0210, 0420, 0421, 0520 and 0800.

2.3 EMV Protocol

The majority of terminals used in Europe satisfies the EMV specifications, that became the payment systems guide. The EMV requirements are represented in a set of specification documents [18]. The EMV standards cover such complex protocols as 3D Secure, Payment Tokenisation, QR Codes, Terminal payment with different communication types, and many others. In this work the focus is on the EMV ICC specifications for Payment Systems.

Once the card is inserted and the contacts are locked by the terminal, the terminal initiates the session. As shown in Figure 2.5, firstly the terminal powers VCC, then sets I/O into reception mode, then provides CLK (periodical high and low voltage values with frequency in the range 1 MHz to 5 MHz). The reception mode of I/O can be set after CLK initiation, but not later then 200 cycles.

Below we describe the activation process in more detail. To activate ICC contacts, the terminal shall generate a supply voltage (Vcc) to the contact VCC of $5V \pm 0.4V$ DC and current drawn in the range 0 to 55 mA. Different ICC card classes support different maximum voltage, but never higher then the value $5V \pm 0.4V$ DC.

The maximum voltage value (H stands for High or '1' stands for "true") and the minimum voltage value (L stands for Low or '0' stands for "false") on the other contacts (Ground, Clock, Reset, I/O) depends on maximum Vcc voltage supported and generated on VCC contact. For example, the card of class C (accepted from January 2014) uses Vcc values 1.66 V to 1.94 V with current 35 mA. Clock and Reset, I/O has a value of H 0.8*Vcc to Vcc, Clock and Reset and I/O have a value of L 0 to 0.15*Vcc. GRD has a value in range of 0 to 0.4V. For I/O voltage value L is 0 to 0.2V (depends on the reception or transmission mode selected), and voltage value H is 0.7*Vcc to Vcc.

After contacts activation, the terminal processes Cold Reset, which is used to receive card configuration. The terminal sets RST value H and, after not less than 400 clocks, receives the ATR (Answer to Reset) message on I/O contact. The maximum delay is 40,000 clocks.

Cold Reset is used to start communication, and in contrast, Warm Reset can be performed without disconnecting power supply. To perform Warm Reset, the terminal sets value L to RST, then after 40.000-45.000 cycles – to value H. And then, as in the case of Cold Reset, the terminal waits for ATR message that contains all card configurations. If Cold Reset leads to a rejected ATR message, the terminal tries to retrieve configurations with Warm Reset. If ATR message has wrong format or is not verified by the terminal, the session should be deactivated.

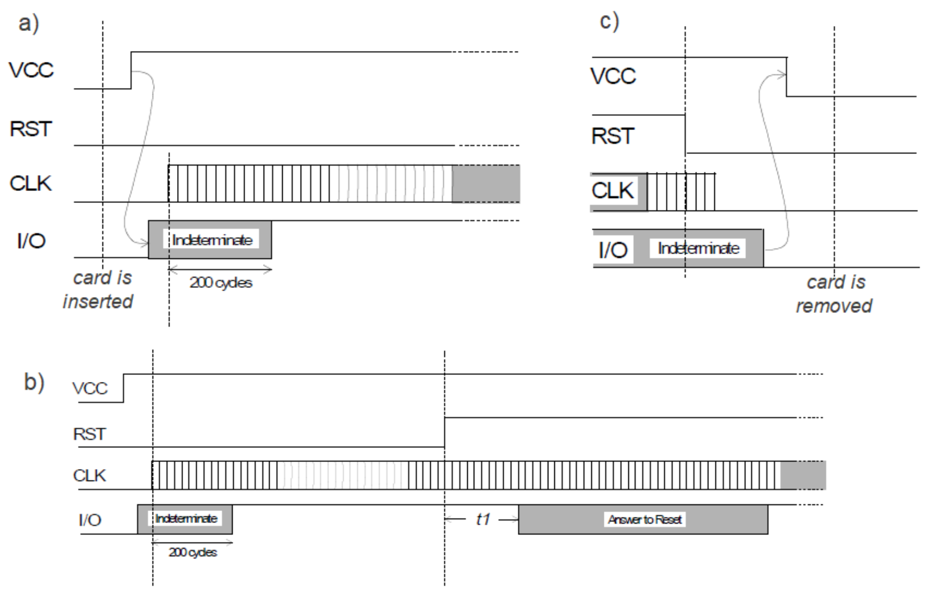


Figure 2.5: Terminal supply sequences: a) Contact activation sequence; b) Cold Reset sequence; c) contact deactivation sequence [13]

To deactivate the ICC card (Figure 2.5 c), the terminal sends the following sequence: sets value L to RST, sets value L to CLK, sets value L to I/O, and, finally, sets value L to VCC.

With ATR message the ICC card provides the terminal with configurations in which the card can communicate. For example, the logic convention of characters in messages (inverse or direct), supported asynchronous transmission protocol (character-oriented “T = 0” or block-oriented “T = 1”), value of programming voltage required by ICC and maximum voltage, the work waiting time (WI), maximum field size (IFSI), error detection code and other values.

When communication is established, the terminal and the card will exchange messages (data elements, files, commands) via I/O line. The terminal controls in which mode the line works: reception or transmission. During the communication, the ICC and the terminal exchange issue information (from ICC files), acquirer or merchant information (from terminal data), transaction details by the protocol agreed upon by both the card and the terminal.

2.3.1 EMV Data Elements, Files and Commands

ICC card is developed to store information. All information in the memory is divided into logical data elements. The minimum set of data elements is described in EMV specification [15], however, it can be extended for proprietary purposes.

Data elements

All data elements are represented in TLV format (Tag, Length, Value), for example, to pass language preference with a length of 2 characters in the tag “5F2D”, the card stores the data sequence “5F2D02EN”.

There is a mapping of the values and tags in Annex A in [15]. The definitions of the tags satisfy ISO/IEC 8825 and ISO/IEC 7816. Bellow there is an example of some EMV data elements:

- Amount data ('81', '9F02' – Authorised amount, '9F04', '9F03' – Secondary amount, '9F3A' – Amount in reference currency)
- Currency data ('9F42', '9F3B', '9F43' – Application currency, '9F44', '9F3D' – Currency minor, '5F2A', '5F36', '9F3C' – Transaction currency)
- Security data ('9F26' – Application cryptogram, '9F27' – Cryptogram date, '8F', '9F22' – Certification Authority public key, '9F49' – DDOL, '9F2D', '9F2E', '9F2F' – PIN encipherment public key certificate, '9F46', '9F47', '9F48' – ICC public key certificate, '90', '9F32', '92' – Issuer public key certificate, '9F4B' – Signed Dynamic Application Data, '93', '9F4A' – Signed Static Application Data, '97' – TDOL, '98' – hash of transaction certificate, '9F37' – Unpredictable Number)
- Dates ('5F24', '5F25' – Application expiration and effective dates, '9A', '9F21' – Transaction date and time)
- Card data ('5A' – Application PAN, '5F34' – PAN sequence number, '5F20', '9F0B' – Cardholder name, '5F57' – Account type, '5F53' - International Bank Account Number IBAN, '5F28', '5F55', '5F56' – Country code, '9F11', '91', '9F0D', '9F0E', '9F0F', '9F10' – Issuer data, '5F2D' – Language preference, '9F13' – Transaction counter, '9F17' – PIN counter, '99' – PIN data)
- Identificators ('9F01', '9F06' – Acquirer Identifier, '5F54' – Bank Identification code BIC, '9F45' – Authentication code, '9F4C' – ICC dynamic number, '9F1E' – Device serial number, '42' – Issuer identification number, '9F16' – Merchant Identifier, '9F1C' – Terminal Identification)
- File names ('94' – AFL, '4F' - ADF name, '84' – DF name, '9D' – DDF name, 'BF0C', 'A5', '6F' – FCI data, '88' – SFI)
- Script commands ('86', '9F18', '71', '72' – Issuer script data, '9F38' – PDOL)
- Merchant data ('9F15' - Category code, '9F4E' – Location and name)
- Version numbers ('9F08', '9F09' – Application version data)
- Risk Management data (9F07' – Application usage control, '8C' – CDOL1, '8D' – CDOL2, '8E', '9F34' – CVM lists and results, '9F4D', '9F4F' – Logs, '9F1D' – Terminal Risk Management data)
- Capabilities ('9F40' – Terminal capabilities, '82' – Card capabilities, '9F39' – Terminal entry mode, '9F33' – Template capabilities)

- Other data elements.

Besides application data elements, smart cards can store extended data, for instance files and commands. According to the EMV specification, this data is represented in Application Protocol Data Unit (APDU) and Transport Protocol Data Unit (TPDU), respectively.

Files

The files in a smart card are organized in a tree structure. The topmost file is the Master File (MF). The MF has one or more Application Definition Files (ADF). Inside of an ADF are Application Elementary Files (AEF) that contain data. An ADF can be selected by Application Identifier (AID) or by Short File Identifier (SFI).

There is an agreement on SFI values: 1-10 governed by the specification, 11-20 Payment Systems specific, 21-30 Issuer-specific, so that there is no conflict in the file identification.

In some cases, the ICC can build the list of data elements, and the terminal receives the data in more handful format. Depending on the type of data, there are four types of Data Object Lists (DOL):

- Processing Options DOL (PDOL)
- Card Risk Management DOLs (CDOLs)
- Transaction Certificate DOL (TDOL) – data for generation of certificate hash value
- Dynamic Data Authentication DOL (DDOL)

Commands

After receiving the ATR message with ICC card configuration, the terminal starts sending the commands, so that the card and the terminal communicate. As mentioned, applications use APDUs to support communication. These data units provide a command-response mechanism, so that the terminal retrieves and passes needed information.

The command (C-ADPU) consists of Header and Body (Figure 2.6). The Header is mandatory for all commands, and the Body is conditional. The structure of the command Header is fixed: class (CL), instruction (INS), the first parameter (P1), and the second parameter (P2). Some commands do not use parameters, in that case the parameter bytes have value '00' in hexadecimal format.

If the Body of the message exists, the structure is: the length of data sent (Lc), Data and the maximum length of data expected in response (Le). The response (R-ADPU) to the command consists of three elements: Data, status byte one (SW1), status byte two (SW2).

The Header in the command has 4 bytes, one for each of its fields.

The status bytes in the response represent the result of the processing state of the command. Possible statuses are shown in Table 2.3.

When Le is present and contains the value "00", the terminal expects a return value with a length up to 256.

Example of READ RECORD command:

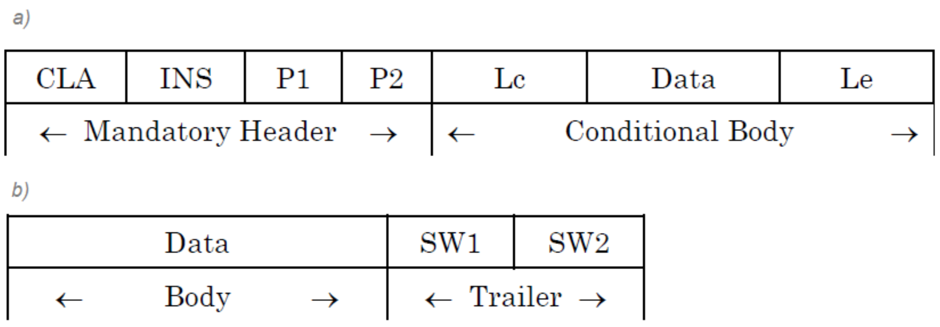


Figure 2.6: Commands structure: a) C-ADPU structure; b) R-ADPU structure

1. the terminal sends command “00B2A10A00”, where “A1” is the record number, “0A” is file identifier. The command does not have Body and expects data with any allowed length in response.
2. in the case of success the card sends back the record data in hex format plus value “9000”, where SW1 = “90” and SW2 = “00”.

More examples of commands can be found in Table 2.4.

Table 2.4: Example of Commands

Command Name	CLA	INS	P1	P2	**Lc**	**Data**	**Le**	**Description**
APPLICATION BLOCK	8x	1E	00	00	Lc	MAC	–	Makes the current application invalid
APPLICATION UNBLOCK	8x	18	00	00	Lc	MAC	–	Rehabilitates the current application
CARD BLOCK	8x	16	00	00	Lc	MAC	–	Permanently disables all ICC applications
EXTERNAL AUTHENTICATE	00	82	00	00	8-16	Cryptogram	–	Requests ICC verification of the cryptogram of Issuer Authentication Data

GENERATE AC	80	AE	xx	00	Lc	Transaction data	00	Requests ICC to generate application cryptogram. P1 specifies what data ICC should return (AAC, TC, ARQC, as described in EVM security subsection), also it specifies if CDA algoritm should use signature or not. Expects CDOL1 and CDOL2
GET CHALLENGE	00	84	00	00	–	–	00	Requests ICC to generate unpredictable number
GET DATA	80	CA	xx	xx	–	–	00	Retrieves ICC data. Parameters specify which of four data objects is requested: 9F36 (Transaction counter), 9F13 (Online transaction counter), 9F17 (PIN counter) or 9F4F (Log format)
GET PROCESSING OPTIONS	80	A8	00	00	Lc	PDOL	00	Requests initialisation the transaction using PDOL, that was obtained from ICC earlier (from tag 83)
INTERNAL AUTHENTICATE	00	88	00	00	Lc	DDOL	00	Requests ICC to generate Signed Dynamic Application Data using provided DDOL (authentication data)
PIN CHANGE / UNBLOCK	8x	24	00	00	Lc	PIN+MAC	–	Requests ICC to change PIN or unblock card. The terminal sends PIN enciphered
READ RECORD	00	B2	xx	xx	–	–	00	Requests a file record in a linear file in ICC. P1 specifies record number. P2 specifies short file identificatior (SFI)
VERIFY	00	20	00	xx	Lc	Encrypted PIN	–	Requests offline PIN verification by ICC. P2 specifies PIN format (Plaintext, Enciphered or proprietary way)
(RFU)	8x	$\frac{\text{Dx}}{\text{Ex}}$	xx	xx	xx	any data	xx	Reserved for use for the Payment Systems

(RFU)	9x xx xx xx	xx any data xx	Reserved for use for manufactures
(RFU)	Ex xx xx xx	xx any data xx	Reserved for issuers

2.3.2 EMV Transaction Flow

After establishing a connection between the card and the terminal, reading ATR message by the terminal and setting I/O line to proper mode, the application session is started. The transaction flow can differ for different card types and terminal types, however some flow stages are the same. Figure 2.7 illustrates the most common transaction flow.

1. Application initialisation – exchange with transaction-related data, sending PDOL to ICC with GET PROCESSING OPTIONS command, receiving Application Interchange Profile, check the need of CDA authentication, requesting Application File Locator, language preference, and other data.

2. Reading application data – the terminal requests data needed for application with READ RECORD commands, including the need of online or offline authentication

3. Terminal Risk Management – protection from fraud. If the terminal processes offline, it periodically performs Data Authentication online (namely sends to issuer for authentication). The authentication to go online is selected based on different characteristics: high amount value (the terminal has a floor value to recognise the transaction as an high-value transaction), frequent card usage (or velocity checking, GET DATA command is used to get the last online authentication), suspicious issuer country, random selection, and so on. The risk management happens before the first GENERATE AC command call. Based on Terminal Risk management the application authentication validation is online or offline.

   Moreover, some terminals have a black list of PANs (Primary Account Numbers), namely card numbers. Suspicious cards can be placed in the list and any transaction with such a card is rejected before the processing of the transaction.

4. Offline data authentication – verify ICC data with SDA, DDA or CDA mechanism as described in subsection *EMV Security Mechanisms*. The first GENERATE AC command is sent. When ICC extra data is needed, the terminal sends READ RECORD commands. In the end of the stage the terminal has verified authentication data.

5. Processing restrictions – checking the compatibility of the terminal application and the card application. The application data retrieved on stage 2 compared with application data of the terminal. Comparison includes checking of application version numbers, application expiration dates, domestic or international transaction, support of purchase indicators (goods, services), supports cashback and so on.

Table 2.3: Possible statuses of the command processing

SW1 SW2	**Meaning**	**Process result**
90xx	Normal	Completed
62xx or 63xx	Warning	
64xx or 65xx	Execution	Aborted
67xx – 6Fxx	Checking	

6. Cardholder verification – verification of the person who uses the card. On stage 2 the terminal finds out what verification method is required (online or offline PIN, signature, no verification required). In case of offline PIN verification, the terminal sends GET DATA (to get Try Counter to enter PIN correctly), GET CHALLENGE (get unpredictable number generated by ICC), VERIFY (to verify entered PIN) commands. The cardholder verifies ownership in accordance with what is required by the terminal method.

7. Terminal Action Analysis – checking if transaction should be completed offline (request of ARQC, terminal action – Default), rejected offline (request of AAC, terminal action – Denial) or completed online (request of TC, terminal action – Online). The Offline transaction processing is described in *C. EMV Security Mechanism*. However, the decision to go offline can be changed on the next stage. To request any of the certificates the terminal sends the GENERATE AC command.

8. Card Action Analysis – the card can have its own mechanism of risk management (issuer adds its own protection from fraud to any card). As in the case of Terminal Risk Management and Action Analysis, the ICC can force the terminal to process the transaction offline (sends ARQC in the response to the first GENERATE AC command), online (sends TC) or reject transaction offline (sends ACC). The ICC can allow transaction offline approving only if the terminal decision is also to approve offline.

9. Issuer authentication together with online processing – verifies transaction by the issuer bank. The terminal sends an online request with transaction data to issuer bank. Then the terminal sends an issuer response to ICC card using EXTERNAL AUTHENTICATE command.

10. Script processing – the terminal runs issuer script over ICC. Issuer script can be preloaded in ICC for offline processing, or can be added during online processing. In the case of online processing, the issuer adds a script that ICC should perform after the transaction. In the offline transaction case, the terminal runs only the preloaded in the ICC scripts. The script can contain commands, data, files.

11. Transaction completion – closure of processing of a transaction. This step is processed even if the transaction terminated with an error.

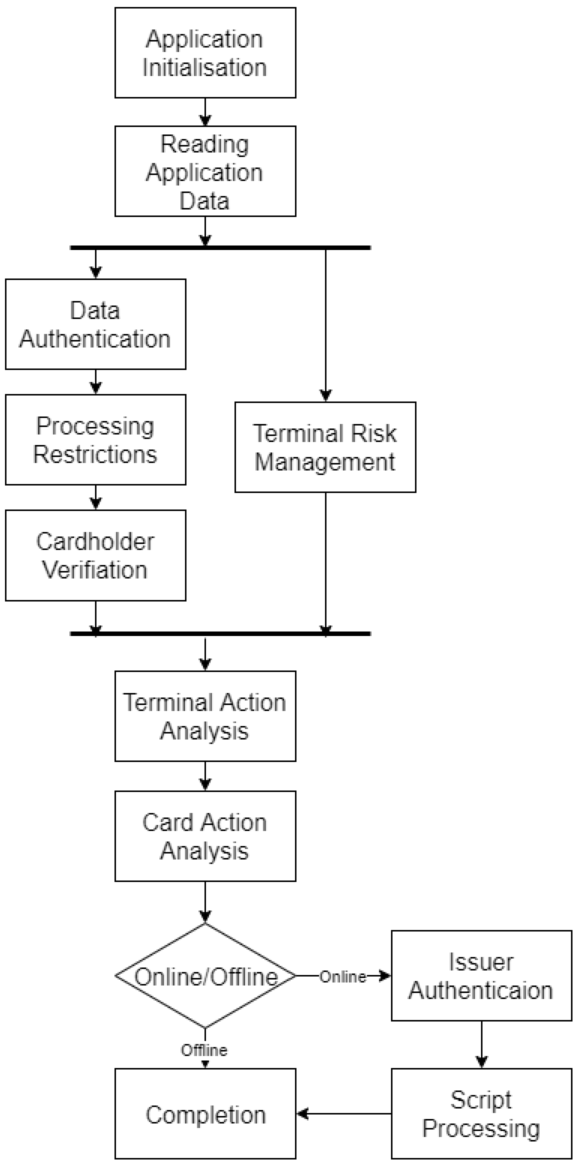


Figure 2.7: Common transaction flow

2.3.3 EMV Security Mechanisms

Not all transactions are processed online, and that is why some operations appear in bank account statements with delay. Sometimes it needs three days to finalise a payment made with the card. Moreover, there are cases of offline payments by terminal, for instance in plane during a flight or in trains in tunnels. Such operations are processed offline and then all transactions from the terminal are processed, causing the acquirer and issuer banks to receive payment transaction information when the terminal reaches the Internet. Being online, the terminal sends then the batch of transactions to process.

In the case of an offline transaction, the terminal shall process card authentication offline as well.

Each ICC card stores as minimum one public key, and as maximum two public keys and a private key. And it's one of the reasons why banking cards have expiration dates. Due to security reasons the keys should be periodically updated. At the same time, the terminals shall store corresponding public key to verify the signature produced by the card or the issuer.

Before sending the terminal to merchant, the terminal is configured with acquirer or merchant information, also the terminal shall store the certificate authority public key. Moreover, since the terminals should accept different card types, the terminal shall be able to store a minimum of six certification authority public keys. At the same time, before usage, the card is configured by Issuer. Since the transactions run offline, the application data should be able to be verified without intervention of Issuer and Acquirer Banks.

The command GENERATE AC can be processed twice: the first command is for offline data authentication, and in case of needing an online authentication (due to Risk Management, for instance), the terminal sends the second GENERATE AC command.

Certification authorities provide the Issuer and Acquirer with the same public key. Issuer uses it to sign its own public key to produce the Issuer public key certificate. The acquirer stores the certification authority public key. Once a compatible ICC card is inserted in the terminal, the terminal uses the public key to verify the Issuer public key certificate. This key is used for authentication, as well as for PIN enchipherment.

Depending on the terminal type and the card type, authentication can be executed with one of three offline authentication mechanisms [14]:

1. SDA – Static Data Authentication, protects against unauthorised data alteration
2. DDA – Dynamic Data Authentication, protects against counterfeiting of the card. The ICC card stores it's own unique key pair
3. CDA – Combined DDA and Application Cryptogram Authentication. Only CDA requires running GENERATE AC command twice.

All three mechanisms require the participation of a Certificate Authority, that signs the issuer public keys (provides public key certificates).

Static Data Authentication

SDA uses a static signature generated by the Issuer. Figure 2.8 illustrates the SDA algorithm (two-layer certification scheme):

1. Configuration of the card before usage:
    (a) Sign Issuer public key with Certification Authority private key. Store the Issuer public key certificate in the card
    (b) Sign ICC application data with Issuer private key and store Signed Static Application Data (SSAD)

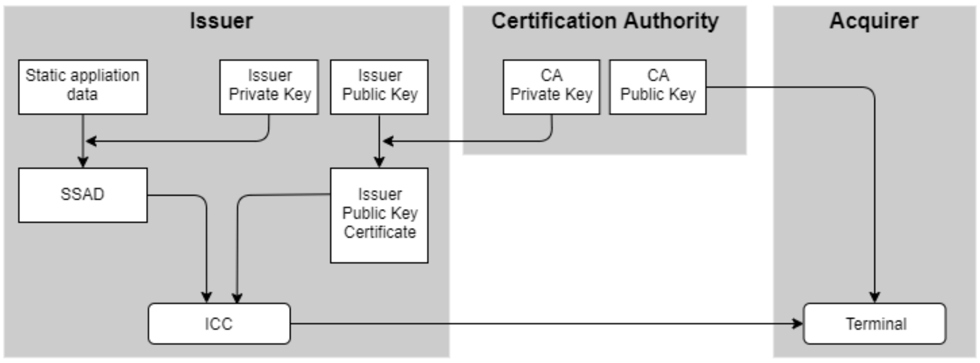


Figure 2.8: Diagram of signing static application data

2. Configuration of the terminal before usage:
    (a) Store Certification Authority public key in the terminal
3. Sending the data:
    (a) The card sends SSAD (signed with Issuer private key) and Issuer public key certificate to the terminal
    (b) The terminal verifies, that Issuer public key certificate is signed by Certification Authority using Authority public key, and obtains Issuer public key.
    (c) The terminal verifies, that the SSAD was signed by Issuer, using Issuer public key

Dynamic Data Authentication

DDA generates dynamic signature using its own private key. Also includes SDA steps, and extends it with dynamic data signature. Figure 2.9 illustrates the DDA algorithm (three-layer certification scheme):

1. Configuration the card before usage:
    (a) Sign Issuer public key with Certification Authority private key. Store the Issuer public key certificate in the card
    (b) Sign ICC public key and Application data with Issuer private key and store together ICC public key certificate and SSAD
    (c) Sign dynamic data with ICC private key when dynamic data needs to be sent
2. Configuration of the terminal before usage:
    (a) Store Certification Authority public key in the terminal

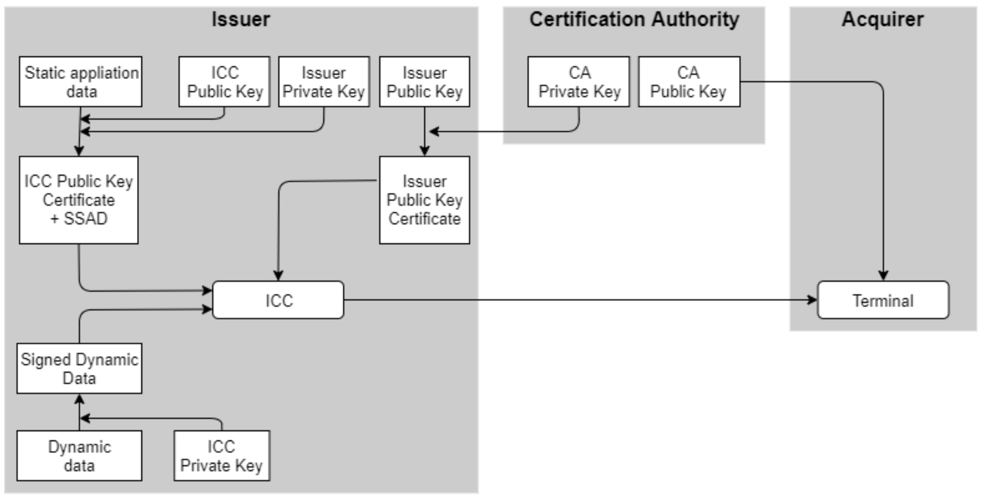


Figure 2.9: Diagram of offline dynamic data authentication

3. Sending the data:

   (a) The card signs dynamic data with ICC private key.

   (b) The card sends ICC public key certificate (together with SSAD), Issuer public key certificate and signed dynamic data to the terminal

   (c) The terminal verifies, that Issuer public key certificate is signed by Certification Authority using Authority public key, and obtains Issuer public key.

   (d) The terminal verifies, that ICC public key certificate is signed by Issuer using Issuer public key, and obtains ICC public key.

   (e) The terminal verifies, that the dynamic data was signed by ICC using ICC public key

To sign dynamic data, the terminal sends the GENERATE AC command, then the card sign dynamic data using ICC private key.

Combined DDA and Application Cryptogram Authentication

CDA generates dynamic signature using ICC private key, but signature includes generated Application Cryptogram. The terminal calls GENERATE AC command twice.

The pre-usage configuration of the card and the terminal is the same as for DDA. However, the process flow differs.

As described earlier in DDA algorithm, in the CDA algorithm the terminal retrieves public keys from the card (ICC public key, Issuer public key).

The first Application cryptogram is taken offline. The card, together with cryptogram, can respond with Signed Dynamic Application Data, that is signed with transaction certificate(TC) or Application Authentication Cryptogram (AAC) or Authorisation Request Cryptogram (ARQC). To process TC the terminal **shall** use CDA signature, to process AAQC the terminal **can** use CDA signature, and for AAC the terminal **shall not** request CDA signature. From the cryptogram, the terminal recovers Signed Dynamic Application Data.

And to verify this data from the ICC card, the terminal should process second GENERATE AC.

In the case of when the ICC, or the terminal, does not support CDA, then the DDA mechanism is used. If the ICC, or the terminal, does not support DDA as well, then SDA mechanism is used.

One more security mechanism that ICC card can have is PIN encipherment. Some types of cards sends PIN via plain text (section 6.5 in [15]), some - enciphered. To perform PIN encipherment, the card stores one more key pair. The terminal retrieve ICC PIN public key with DDA mechanism. Moreover, the PIN value is extended with unpredictable number (the challenge is generated by ICC) that could be mere random number, or random number extended with other transaction data. The encipherment is done with RSA (Rivest–Shamir–Adleman) algorithm.

Message exchange is protected with message signature and encipherment. Data integrity and issuer authentication are achieved using a MAC. Data confidentiality is achieved using encipherment of the data field. Separate MAC Master key and Encipherment Master key are used to produce unique MAC Session key and Encipherment Session key respectively. For message encipherment DES, Tripple DES or AES algorithm is used.

2.3.4 EMV Security Vulnerabilities

The appearance of international standards, that guide payment systems and banks and bring best practices into the area, significantly improved security in payment processing. One could fall into the trap of assuming that all smart cards are more secure than magnetic stripe cards and that the presence of the gold contacts implies that this is a smart card. These assumptions are incorrect and dangerous [34]. Although the EMV specifications are developed to reach as secure payment processing as possible, a number of vulnerabilities still exists [2].

In the paper [30] the authors made an overview of EMV vulnerabilities. The most relevant from them are listed bellow:

1. Acquirer banks and their clients (merchants) try to make the terminal support many payment schemes (Visa, Mastercard, Diners Club, and so on). Since the schemes have different communication requirements, provide different sets of data, use different verification mechanisms, it is possible to introduce vulnerabilities [4, 12] by switching some card detail, or alternatively terminal data. Then the connection becomes vulnerable to a downgrade attack.

   Bypass the usage of card private key. As was described in EMV Security Mechanisms subsection, the card has non-copyable private key. In fact, the card with private key can be copied, but the terminal rejects the verification of that card. However there is a possibility to make a copy of NFC card without the card key and then reconfigure the capabilities of the copy to force the terminal request processing transaction using magnetic stripe, which is less secure [12].

2. One vulnerability in EMV protocol was found in PIN verification stage [36]. When using PoS terminal offline, an attacker could force the terminal to show the message “No PIN required” to the cardholder, while sending to the Acquirer Bank status of successful pin verification. So that the transaction could be processed without PIN verification [12].

3. The weakest data authentication method in smart cards is SDA. It is also known to be even less secure than magnetic stripe communication [34]. Data from smart cards with SDA method can be easily copied and the cloned card can be used in offline transactions. And on top of that this case of fraud can be detected only by the issuer and not by PoS terminal, which gives to the attacker more time to act [6].

4. Sensitive data is easy to access in the PoS terminal. Following the EMV specification, the acquirer bank sends PAN and card expiry date in clear plain text just after the card application [31, 11]. It means, the terminal stores sensitive card data even before the cardholder verification (for example PIN entering). Recently this vulnerability was eliminated by using Tokenisation [17]. Moreover, some payment system companies introduce encryption of sensitive data received from the acquirer [32].

5. Sensitive card data can be accessed not only from the terminal memory [45]. After transaction processing, the terminal prints two receipts: the first is for the client, the second - for the merchant. The cardholder’s receipt contains PAN in masked format, nevertheless the merchant’s one has PAN and expiry date in clear text [23].

To sum up, the goal of EMV specifications is to increase security of payment transaction and to provide international guidelines for the payment systems. However, several research works describe possible security vulnerabilities of the EMV protocol.

2.4 Payment Terminal Emulator Niche

After implementation of terminal software, its interface and acquirer host communication, it is necessary to ensure these are tested and certified according to acquirer requirements and EMVCo specifications. Moreover, security vulnerabilities should be taken into account. And this can often be a complex process.

The payment terminal emulator has to be developed to provide complete, automated validation of the terminal payment with a smart card.

The benefits of having such an payment terminal emulator developed for the Payment System companies are:

1. Facilitated and accelerated integration into acquirer system. Since the longest phase of introduction of a PoS solution into business is formal certification, Payment System companies, acquirers and sometimes merchants are interested in accelerating testing and certification.

2. Making the development cheaper. The payment terminal emulator leads to discovering failures and business mistakes earlier, therefore the total cost of development decreases.

3. Providing safer final solution.

So, the reasons why the development of the new payment terminal emulator is needed comprise:

1. Solution uniqueness. Every acquirer has its own transaction flow implementation, this includes different sets of issuer and merchant data to be stored, host interface characteristics, and so on. Moreover, acquirer specifies types of terminals to be used in the business, therefore PoS interface obtains unique requirements. All this together leads to the need of unique use cases that have to be covered with the payment terminal emulator.

2. Publishing implementation details. All existing payment terminal emulators are proprietary, and few reports were done in this area. The detailed report on implementation of the emulator for specific Payment System can give understanding to others who are interested in EMV certification process.

3. Getting business advantage in the FinTech market, which is highly competitive. Since Payment system companies aim to make PoS terminals integration faster, cheaper and safer to make them more attractive for the clients, the companies are interested in improving certification process, that can be done using an payment terminal emulator.

The payment terminal emulator can be used by:

1. Payment System employees (developers, QA engineers, certification experts)

2. Acquirer integration experts

3. EMV certification experts

As was mentioned, many Payment System companies developed their own emulators and the majority of them is hidden from public and has proprietary goal. However, the companies give brief emulators description in advertisement purposes. Table 2.5 illustrates eight similar verification tools that can serve for terminal emulation purpose.

2.5 Emulator Software Application

When developing new PoS terminal solution for new acquirer, the verification process of the new acquirer interface integration can be divided into the phases:

1. During development process: the verification is done by developers using test host, any test card, any test terminal, to check acquirer host connection, and message format. This phase closes the development stage.

Table 2.5: Analogs of payment terminal emulator applications

PS Company	**Application Name**	**Functional notes**
FIME	EMVeriPOS [21]	Complete, automated validation of your payment terminal's protocol components against EMV Level 1, enhancing global interoperability
WeChip	WeChip Simulator [52]	All Payment Schemes, Authorization and Scripting, Acquirer Key and Parameter, Card Profile Comparison
SmartOI	SmartOI Terminal Simulator [46]	Any kind of terminal type, all CVM, all ODA methods (SDA, DDA, CDA), configurable AIDs list for application selection, configurable CAPKs, RID and CAPK Index, Revocation List, and Default DDOL for ODA
EFTLab	BP-SIM [10]	Test all elements of the entire payment infrastructure (POS, Switch, Acquirer, Issuer, *etc.*)
IMS UL	UL Brand Test Tool [51]	Verification for PoS terminals, Mobile POS (MPOS) devices, Automated Teller Machines (ATM), Vending machines and other payment devices
Abe-Tech	EMV L1 PCD Test Solutions [1]	Covers the variety of standards such as ISO10373-6, NFC Forum, ICAO, EMVCo, EAC, ISO18013, CIPURSE, *etc.*
KaSYS Canada	KaNest®-ICC Card Simulator [27]	Provides solutions for verification over NFC, ISO14443, FeliCa, ISO15693, SWP and ISO7816 technologies
FIS	EMV Test Tools [22]	Both the pre-certification and brand certification processes

2. Pre-acquirer testing: the verification is done by the quality assurance team using a test host, a set of acquirer test cards, and a set of payment system terminals, to check if special smart cards are read properly and the transaction data transferred to the acquirer.

3. Formal certification: the verification is done by the certification experts using an acquirer host, acquirer test cards, a set of payment system terminals, to check full transaction flow in close to production environment.

The payment terminal emulator application should support test activities in all three verification phases.

2.6 Summary

After a comprehensive review of the current state-of-the-art in Payment Terminals, one can conclude that payment process includes a lot of complicated phases. The EMV specification became the international standard. It not only covers requirements for certification, but also provides a guidance to developers and certification experts in payment area all over the world. In this dissertation work, the small part of the EMV specifications was considered, namely Books 1-4 of EMV ICC Specifications for Payment Systems [13, 14, 15, 16].

Nowadays, the payment process, starting from the production of the payment mean with issuer bank data, and until the phases of PoS terminal software updates, and funds transfers between acquirer and issuer banks, is under continuous improvement due to the emergence of new technologies. After the smart card has been presented and current time, the payment process has been dramatically improved. That is why there many FinTech companies, that provide partial solution, that can be integrated with each other.

One of such companies suggested the project of development a standalone application to facilitate development and testing of the terminals software and reduce the cost of buying expensive hardware for development environment. The payment terminal emulator is an application to reproduce the full payment transaction process. The emulator helps to test integration of new acquirer interface into the payment system based on EMV protocol. Some existing analogs and possible solution niche were studied.

Chapter 3

Requirements

In this chapter we provide the detailed view on requirements of the requested application. We start with general overview of the application in Section 3.2, than in Section 3.3 we list all requirements that were agreed with the company in the beginning of the project development. Moreover, the same requirements are used in project acceptance test (the results are provided in Chapter 6). The Use Case diagram in Section 3.4 illustrates the main scenarios that the user can perform with the application. Section 3.5 provides the initial scheme of the planned architecture of the application. Finally, the list of technologies to be used during development was defined, and we include it in Section 3.6.

3.1 Introduction

Besides the study of the existing EMV specifications and the general payment process in the bank and payment system network, the dissertation work includes development of a standalone application, that emulates the behaviour of the real terminal. The payment terminal emulator application should have the GUI, that looks like the real terminal and should be easy to understand by 3C TechLab developers and certification experts. Based on gained knowledge and existing GUI of the application, the project work was planned. The application name "3C Emulator" was proposed.

3.2 Expected system behavior

The main objective for the project is to process payment, *e.g.* the first general requirement was that the application should process the payment in the test environment. It is important that the payment is processed following the EMV specification, as discussed in Chapter 2. However, some EMV requirements can be omitted in the application due to the unfeasibility of implementation of hardware-specific algorithms, such as low level processing of the card data (reading card data by its ICC contacts). In this solution it is supposed that the card data is read and provided to the payment terminal emulator application as a list of parameters.

The application should implement different payment types, such as Sale, Refund, and Reversal. The corresponding message should be sent to the 3C TechLab UAT server, where the request should be mapped in the message format that selected acquirer expect, and the message is send to the acquirer. The application should write all passed steps into a log file or/and to a database to clarify the flow and to provide payment processing evidence. The processing steps should follow the EMV specification and include the steps from Figure 2.7.

Finally, the application should emulate printing a receipt after the processing of the transaction. Since the software application does not have an active receipt printer, the application should save the receipt in a text file. The receipt should contain transaction data, as well as the processing result.

After the implementation, the testing against the list of requirements and EMV specification should be performed. Then, it is expected that the payment terminal emulator application to be integrated into the 3C TechLab development and certification process.

Finally, the finished product has to follow company naming policies, namely have the **3C** acronym. Therefore, the name "3C Emulator" was proposed.

3.3 Requirements

In the beginning of this dissertation work, the list of requirements was defined. However, during the year of the implementation, the list had some alteration. The final list of requirements to the software project was the following:

1. Project requirements:
    - 1.1. Provide a terminal-like software solution;
    - 1.2. Integrate the application with the existing 3C TechLab system interfaces, such as IntegraFE, NAS, DCC service and others;
    - 1.3. Design good software architecture, *e.g.* use established design patterns to improve the code structure and application performance;
    - 1.4. Process unit testing and provide final test report;
2. Application requirements:
    - 2.1. The application should have an appearance similar to the terminals of Worldline YOMANI (Figure 3.1) type.
    - 2.2. The application should have display, digits keyboard, buttons OK, Stop, Correct, and Menu. The application can optionally have the buttons Standalone, up, down, and select;
    - 2.3. The application should show what types of readers are activated;

Figure 3.1: One of the Worldline YOMANI terminals [53]

2.4. The application should allow the user selection of the cards from the list;

2.5. The application should allow the user to add, edit and delete test cards manually. The application can optionally support the usage of an external ICC reader to save card data in the system;

2.6. The application should support showing the messages from the standard EMV terminal messages list (Table 2.1) in different languages. The terminal can optionally support other informational messages to facilitate the application usage;

2.7. The application should not use any third party's libraries, but use the libraries that were developed inside the 3C Payment company, as well as trusted libraries from Apache, Java and Google projects;

3. Payment process requirements:

3.1. The application should communicate with the 3C TechLab UAT payment server. The addresses of the external services should be located in the application configuration file, so that the user can change the services the application communicates with;

3.2. The application by default should be configured to process transactions in the fake acquirer network (BNKSM). However, the application should be able to be configured to reach real acquirer test networks to process a transaction;

3.3. The application should emulate the processing card data reading only by two card readers: magnetic stripe and ICC readers. The application can optionally emulate Bluetooth and NFC readers;

3.4. The application should support the following card types: Visa, Mastercard, Maestro. The application can optionally support Diners, JCB, American Express and other card types;

3.5. The application should process the payment transaction following the EMV specification. The processing should include Data Authentication, Cardholder Verification, Terminal Risk Management, Terminal Action Analysis, Offline Authentication. The application can optionally support Card Action Analysis, Online Issue Authentication, Script Processing;

3.6. The application should build transaction messages in ISO8583 format [25] and send it to 3C TechLab server. The payment result from the server, which is also in ISO8583 format, should be parsed and the result should be shown in the application display;

3.7. The application in the case of successful transaction processing, should show the authorisation code from the acquirer as well;

3.8. The application should connect to the Certification Authority storage and process the key signing algorithm;

3.9. The application should support "T = 1" protocol only. The application can optionally support "T = 0" protocol;

3.10. The application should process PIN verification offline. The PIN processing can be omitted if the transaction does not require PIN verification, *e.g.* small amount transactions with trusted card;

3.11. The application by default should process authentication following the CDA algorithm. The application can optionally support authentication via DDA and SDA algorithms. In this case the switcher between authentication algorithms should be implemented;

3.12. The application should emulate printing a receipt, *e.g.* provide receipt within a text file.

3.4 Use Cases

The Use Case diagram in Figure 3.2 was built to summarize the list of requirements and illustrate the minimum features of the application, that the user can perform.

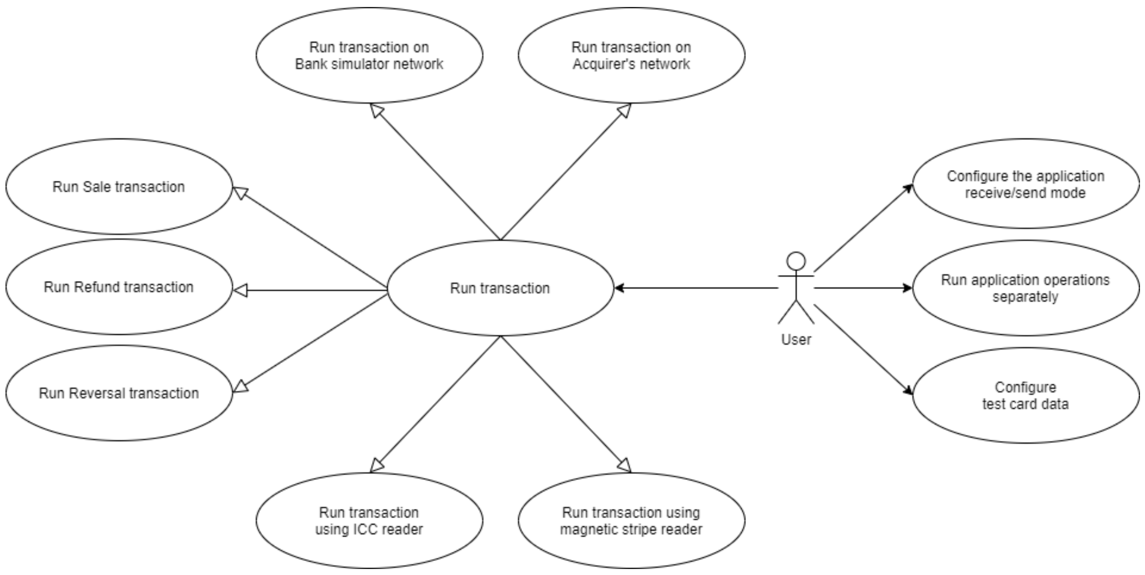

Run transaction on Bank simulator network
Run transaction on Acquirer's network
Run Sale transaction
Run Refund transaction
Run Reversal transaction
Run transaction
User
Configure the application receive/send mode
Run application operations separately
Configure test card data
Run transaction using ICC reader
Run transaction using magnetic stripe reader

Figure 3.2: The use case diagram of the 3C Emulator application

In the diagram, the actor User is an user of the 3C Emulator application. Any developer, certification engineer in 3C TechLab is a potential user of the application.

The application behavior has four main scenarios and the list of sub-scenarios:

1. Configure the application receive/send mode – the user can configure 3C Emulator in receive events or/and send events (after operations execution) mode;

2. Run application operations separately – the user can run any 3C Emulator operations (make a beep sound, turn the ICC reader on, process card swipe, and other operations: all possible operations are presented in Appendix A.2);

3. Configure test card data – the user can add, edit, and delete test card data. 3C Emulator has a list of pre-configured test cards, however, the user can extend it, if needed;

4. Run transaction – the user can process the payment. This scenario is essential for the application. Moreover, the application should provide the options to the user to process the transaction in different ways, namely:

   (a) Run transaction on Bank simulator or Acquirer's network – the user can configure the application to make the NAS service use acquirer network interface or use the fake network to process the payment;

   (b) Run Sale, or Refund, or Reversal transaction – during the phase of introducing the payment data, the user can select which type of transaction should be run (see 2.2.3). Based on the selected option, 3C Emulator should request corresponding data from the user (such as amount, card data, RRN of the original Sale transaction, and other data). The type of the built message should correspond to the selected transaction type;

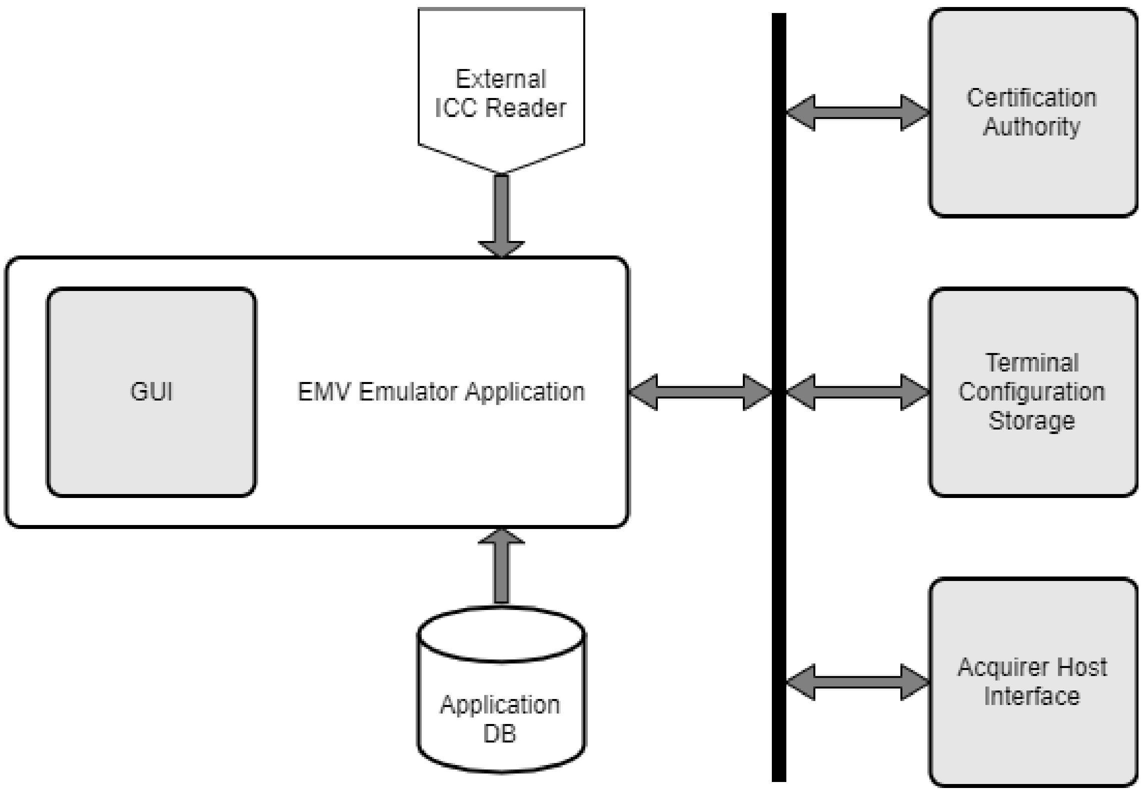


Figure 3.3: Proposed Application Architecture

(c) Run transaction using ICC or magnetic stripe reader – the user can select how to use the test card for the payment. 3C Emulator can use four types of readers, however, for this work only two reading capabilities should be implemented.

3.5 System Model

In the beginning of this dissertation work, a possible architecture was schematized. The proposed architecture scheme can be found in Figure 3.3.

The application, as was discussed above, contains a GUI with terminal-like appearance. The interface should be implemented using java framework in a separate java project, so that it facilitates the reading of project code.

Next, the application should include all business logic, including dependencies, in the same application package, so that the application can be easily installed in any PC of a developer or certification expert. The same is applicable to the application database. The database should be light enough to be stored inside the application package. Therefore, the SQLite [47] database was proposed.

Optionally, the application can use the external ICC reader to facilitate adding new test card data. If not, the application should provide a way to insert the test card data manually using the GUI, or in the application configuration file, or directly in the Application DB.

The application should communicate with a number of an external interfaces. In Figure 3.3 three interfaces are presented. The biggest is the Acquirer Host Interface, which include complicated chain of interfaces, namely the 3C TechLab services (IntegraFE, NAS, DCC service) interfaces and the Acquirer network interface itself. A more detailed 3C TechLab system architecture is presented in Chapter 4.

The application also should communicate with the Certification Authority service. This service provides the business logic of signing public keys, sharing the keys between the terminals, issuer banks and Payment System services.

Finally, the application can optionally have interface with Terminal Configuration Storage. If not, the configuration should be stored locally in the application package.

3.6 Project Technology Stack

During the project development the following technologies were chosen to be used, in concertation with 3C TechLab:

- **Java** – is a general-purpose programming language that is class-based, object-oriented, and designed to have as few implementation dependencies as possible [38]. The application is written in this language, using version 8;

- **Java AWT (Abstract Window Toolkit)** – is Java's original platform-dependent windowing, graphics, and user-interface widget toolkit, preceding Swing [3]. In the project the toolkit was used to develop the GUI of the application;

- **Eclipse** – is an integrated development environment (IDE) [9] that was used to implement the project in Java. Version 4.11.0 was used;

- **Gerrit** – is is a free, web-based code review tool based on different Git repositories [43]. In the project, the tool was used together with GitExtension application [41];

- **Trello** – is a web-based Kanban-style list-making application which is a subsidiary of Atlassian [50]. In the project the application was used to manage project tasks along the period of implementation and testing;

- **SQLite** – SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. [47]. In the project is used to implement the application database;

- **JUnit4** – is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks [26]. In the project the library is used to build and run unit tests, as well as build test report;

- **Bluecove** – is a Java library for Bluetooth that currently interfaces with the Mac OS X, WIDCOMM, BlueSoleil and Microsoft Bluetooth stack found in Windows XP SP2 or Windows Vista and WIDCOMM and Microsoft Bluetooth stack on Windows Mobile [5]. In the project the library is used to implement the Bluetooth reader feature;

- **Protobuf (Protocol buffers)** – are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler [8]. In the project is used to facilitate send/receive operations and events logic between application components;

- **Xerces Parser** – is an Apache library, the reference implementation of XNI. Other parser components, configurations, and parsers can be written using the Xerces Native Interface [54]. In the project the library is used to parse XML messages using the DOM tree;

- **log4j** – is a reliable, fast and flexible logging framework (APIs) written in Java, which is distributed under the Apache Software License [29]. In the project is used to write and save log files;

- **IntegraFE libraries** – libraries, developed inside 3C TechLab by he IntegraFE team;

3.7 Summary

The second part of the dissertation work is development of software application. Following the usual lifecycle of software project, before any work to be started, the project objectives need to be set. Therefore, we started with establishing the requirements, both functional and non-functional.

The project of development of the payment terminal emulator is not new in the company, however the existing solution is still missing essential parts, such as payment with ICC card. In addition, the existing code need to be maintained and improved using the best software design practices.

The scheme of possible application architecture was proposed by 3C TechLab, moreover, the company already has many components, which 3C Emulator should communicate with. Together with architecture scheme, the interfaces of the application were defined. More experienced developers of 3C TechLab company suggested the technologies that are better to be used in the project. At the same time, the project planning was done from the point of view of the user, so that several user scenarios were consider. This facilitate understanding of the application GUI behavior.

Chapter 4

Architecture

This chapter presents the view to the application architecture from two sides – from outside (from the complete system of 3C TechLab) and inside (the inner components of the 3C Emulator application).

4.1 Introduction

The 3C TechLab company is a relatively new company founded in 2016 as a subsidiary of 3C Payment. The given name can explain the original objective of the creation of the new company. It was a sort of laboratory to research and realize fresh FinTech ideas into life. At the same time, 3C Payment is well-known payment solution provider in the FinTech market with more than 15 years of experience. During that time the company built their own full business solutions and won trust of a big number of merchants all over the world.

Since 3C TechLab is a subsidiary company of 3C Payment company, they develop and use a common system. Therefore, in this chapter we use the 3C Payment company name as an owner of the system services. However, this dissertation work was accomplished inside the 3C TechLab company.

4.2 System Architecture

The 3C Payment architecture is represented in a set of components. The components are loosely coupled, so that the system remains to be easy-scalable. In Figure 4.1 the system components are represented as standalone servers, meanwhile, a few servers joined with a Load Balancer to spread the load equally stay behind the image of the one component.

Figure 4.1 illustrates the main components that the company architecture contains:

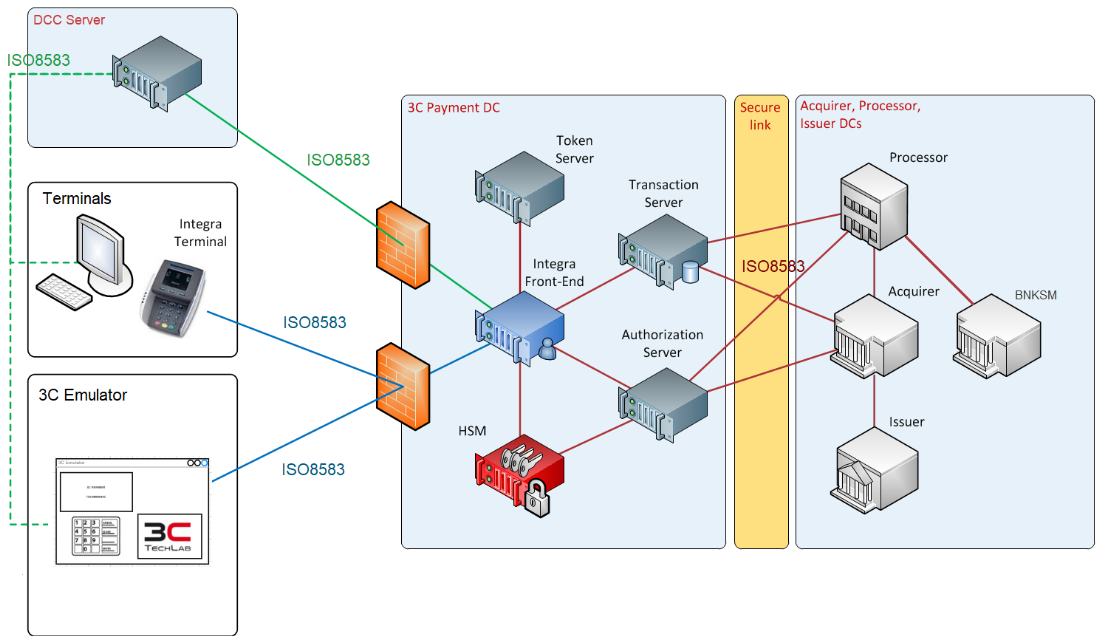


Figure 4.1: 3C Payment system architecture

- **Terminals system (TE)** – includes terminal hardware (PoS, mPoS, attended and unattended locations, and other) and its software. Moreover, the TE components include several peripheral servers, such as the Public Key Storage server. The component has three main interfaces – with DCC and FE servers. TE system uses 3CXml message format and uses the TCP/IP protocol to pass the messages through the network.

- **Integra Front-End (or IntegraFE, or FE)** – despite FE does not have any web or graphical interface, the system was named as Front End. This is because FE represents the communication interface between the merchant and the 3C Payment system, being the the first module that the TE system communicates with to execute a transaction. In this case, FE is a complicated interface of the whole 3C Payment payment system. It uses Oracle Payment Interface to facilitate structure and provides a more secure interface between TE and FE. Also it uses Ethernet configuration protocol/loop (CTP). The component communicates with 3C Payment database using the JDBC API.

- **(New) Authorization Service module (NAS)** – contains a significant portion of 3C Payment's business logic, has the main function of receiving the transaction representative message and, through various mechanisms, forwards it to the respective banking interface, either syntactically or semantically. These messages, which normally occur in a request-response dialectic, are formatted by the ISO8583 standard. However, some other message formats were also created, by extending ISO8583, in order to comprise all transaction data, needed by acquirers and other 3C Payment components.

- **Transaction server** – the component of passing the transaction data to the processor. Some acquirers use a facilitated scheme of message builders, so that there is no need to include NAS processing. The component prepares the messages to be sent to the acquirer network.

- **Token server** – is a service that aims to generate a unique identifier or token to replace payment card data, thus eliminating confidential payment information on the network. Tokenisation allows secure cross-channel payment flows for the same transaction, *e.g.* in pre-authorizations and reversals data travels through different channels securely, making external attacks difficult. The communication with Token server is done using a REST API.

- **Hardware Security Module (HSM)** – is a component that is responsible for managing the authentication keys and provides the encryption to IntegraFE and also to the NAS. It is a physical device that provides extra security for sensitive data [40].

- **Processors** – the number of services-interfaces that represent acquirers or the Bank Simulator interface. The components are responsible for connection timeouts, sending diagnostics messages (Echo, Sign in, Sign out, and other), as well as caching special acquirer values that can be used in several transactions, such as STAN, RRN, Pre-authorisation amount value, and other data. Each processor is configured in correspondence with acquirer network requirements.

- **Acquirers and Bank Simulator (BNKSM) systems** – although the acquirer systems are included in the diagram, those systems are located on the acquirer side. Normally, acquirers also have services-interfaces as a first point of reaching from 3C Payment system. The Bank Simulator is a fake-acquirer network, that is located in the 3C Payment system, and is used for testing reasons.

- **Issuers systems** – these systems are not located in the 3C Payment system. The acquirer directly communicate with the Issuer to process the transfer of funds to finish the transaction.

- **Dynamic Currency Conversion (DCC) service** – provides the mechanism of converting one currency to another. In this way, the payment terminal immediately recognizes foreign cards and offers the customer the option of paying in their national currency, thus perceiving the value with the applied exchange rate. Moreover, the service is used by FE and NAS. Some acquirers prefer to not use on-terminal conversion and process the conversion in the payment system. All components that use currency conversion communicate with the DCC service through a SOAP API.

- **3C Emulator application** – is an automatic payment terminal emulator with an interface similar to a payment terminal. This dissertation work mainly contributed to the design and development of this component.

The communication between 3C Payment components in the majority of cases is done using a network with the TCP/IP protocol. The messages are built using ISO8583 format and sent to the

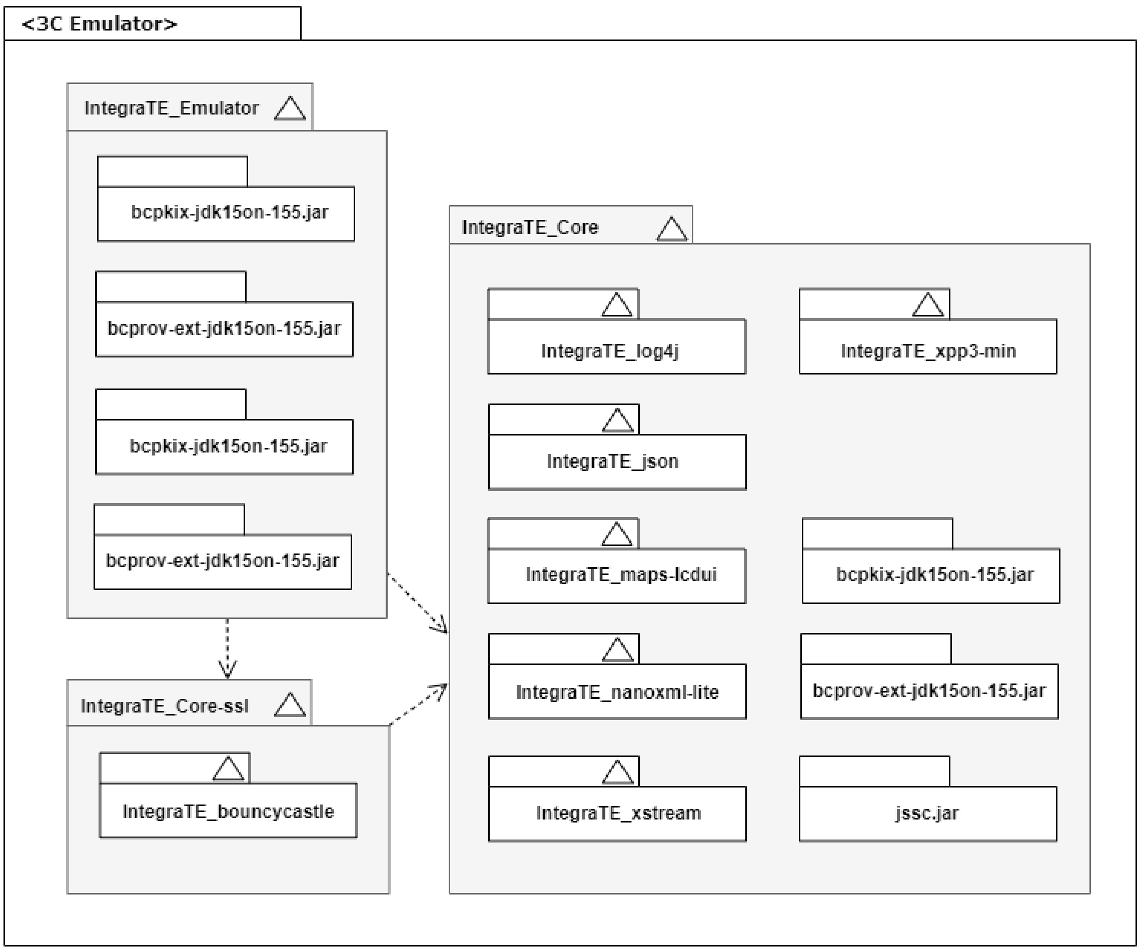


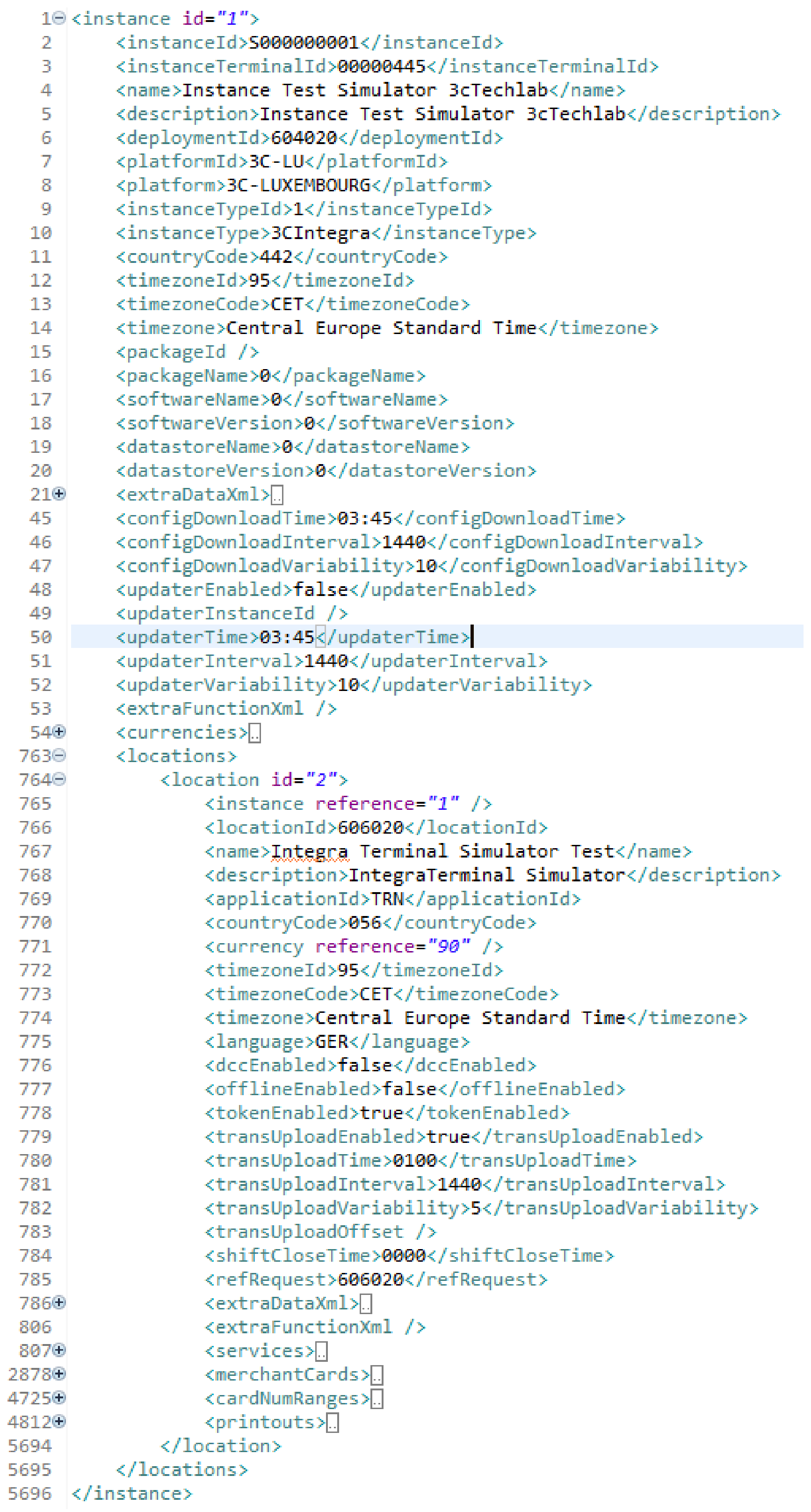
Figure 4.2: Package Diagram of the 3C Emulator application

corresponding interface. Clarifying the format ISO8583 we should mention that it was extended by the company in order to support all needed fields in the message. As a result, the company uses its own message formats (3CAS ISO8583 and 3Cxml ISO8583). The same is applicable to the acquirer network message format. Some acquirers use mere ISO format, however others prefer to extend it to meet their business needs.

4.3 Application Architecture

The 3C Emulator application is a standalone application and can be packed in a running file. Despite simplicity of the user cases that the GUI provides, the application has a nested structure in order to separate application logic into several Java projects. As illustrated on Figure 4.2, the application consists of 3 main components:

1. IntegraTE_Emulator – contains all GUI implementations. The Java AWT framework is used to link graphical components, such as dropdown menus, buttons, *etc.*, with the events and operations, that the application receives and processes. The project has six references: four



```xml
<instance id="1">
    <instanceId>S000000001</instanceId>
    <instanceTerminalId>00000445</instanceTerminalId>
    <name>Instance Test Simulator 3cTechlab</name>
    <description>Instance Test Simulator 3cTechlab</description>
    <deploymentId>604020</deploymentId>
    <platformId>3C-LU</platformId>
    <platform>3C-LUXEMBOURG</platform>
    <instanceTypeId>1</instanceTypeId>
    <instanceType>3CIntegra</instanceType>
    <countryCode>442</countryCode>
    <timezoneId>95</timezoneId>
    <timezoneCode>CET</timezoneCode>
    <timezone>Central Europe Standard Time</timezone>
    <packageId />
    <packageName>0</packageName>
    <softwareName>0</softwareName>
    <softwareVersion>0</softwareVersion>
    <datastoreName>0</datastoreName>
    <datastoreVersion>0</datastoreVersion>
    <extraDataXml>
    <configDownloadTime>03:45</configDownloadTime>
    <configDownloadInterval>1440</configDownloadInterval>
    <configDownloadVariability>10</configDownloadVariability>
    <updaterEnabled>false</updaterEnabled>
    <updaterInstanceId />
    <updaterTime>03:45</updaterTime>
    <updaterInterval>1440</updaterInterval>
    <updaterVariability>10</updaterVariability>
    <extraFunctionXml />
    <currencies>
    <locations>
        <location id="2">
            <instance reference="1" />
            <locationId>606020</locationId>
            <name>Integra Terminal Simulator Test</name>
            <description>IntegraTerminal Simulator</description>
            <applicationId>TRN</applicationId>
            <countryCode>056</countryCode>
            <currency reference="90" />
            <timezoneId>95</timezoneId>
            <timezoneCode>CET</timezoneCode>
            <timezone>Central Europe Standard Time</timezone>
            <language>GER</language>
            <dccEnabled>false</dccEnabled>
            <offlineEnabled>false</offlineEnabled>
            <tokenEnabled>true</tokenEnabled>
            <transUploadEnabled>true</transUploadEnabled>
            <transUploadTime>0100</transUploadTime>
            <transUploadInterval>1440</transUploadInterval>
            <transUploadVariability>5</transUploadVariability>
            <transUploadOffset />
            <shiftCloseTime>0000</shiftCloseTime>
            <refRequest>606020</refRequest>
            <extraDataXml>
            <extraFunctionXml />
            <services>
            <merchantCards>
            <cardNumRanges>
            <printouts>
        </location>
    </locations>
</instance>
```

Figure 4.3: The screenshot of configuration file *cccterminalinal-3cixml-default.xml* in partially collapsed form

of them to standalone libraries, as the JSSC (Java Simple Serial Connector) for communications; and the other two – to the application projects of the repository IntegraTE_Core and IntegraTE_Core-ssl;

2. IntegraTE_Core – stores all application logic, namely building, sending, receiving and parsing messages, making database updates, configuring the connection with external services, *etc.* The java project includes six small projects placed in the same repository, where three are external libraries;

3. IntegraTE_Core-ssl – is a relatively small project, that implements network connection logic. For example it has implementations using the SSL or TLS protocols, selection of network certificates, *etc.* The project uses only one reference, another small project from the same repository.

The application stores all configuration values in one place – folder **/IntegraTE_Emulator /IntegraTES/appconfig/cccterminal/3cixml**. All configurations files have the *.xml* extension. The screenshot of the configuration file is presented on Figure 4.3. In the configuration file the user can configure the following values:

1. Terminal configurations, such as Instance ID, Deployment ID, location (country, address, timezone, software version), TID and MID values, extra data for maximum size of log file and maximum storage capacity, and other configurations. Although the values are mandatory for any real terminal software, 3C Emulator uses these values in a way to emulate the real terminal processing;

2. List of currencies supported by the terminal – each terminal can have several supported currencies. In 3C Emulator we have 101 currencies supported;

3. List of acquirer locations associated with the terminal – when the real terminal is installed into the merchant location, the address of the acquirer network should be assigned to the terminal. In the application by default we use the BNKSM location. The reason is that during development a huge number of transaction were processed, and this can heavily load the acquirer network, its interface and 3C Payment processor interface. The location can also include the address of the 3C Payment system components, such as IntegraFE, however by default it uses the UAT service address.

4.4 Summary

One of the important phase of a software project development is planning of system and application architecture. In this chapter we considered the architecture of the company system and highlighted the place of the 3C Emulator application. Then we designed the scheme of the application itself in order to provide the view from the inside. Moreover, the configuration file of the application was presented.

Chapter 5

Implementation

This chapter presents in detail all the work developed for this dissertation, as well as the improved software solution. We provide explanation of GUI behavior, some implementation detail of business logic, database structure, the interface with external services, and explain a main design solutions. Finally, we include the test report of unit tests that were written to verify the software application quality.

5.1 Introduction

Before the implementation of the payment terminal emulator, the known application analogs were studied (Table 2.5). Since the applications are proprietary, the detailed description is not provided. At the beginning of the work, an existing GUI for the emulator, developed by 3C TechLab, was studied and improved. The application already had some functional features implemented, however the main EMV payment flow was missing. The main part of the work performed in this dissertation was the specification and implementation of payment flow and testing.

5.2 Graphical User Interface

The main objective of the application was to provide a terminal-like solution, making usable the validation and test of payment flow. As a default terminal, we selected the most used in development and certification process terminal type, which is Worldline YOMANI [53].

The initial screen of the application, shown on Figure 5.1, represents the terminal physical appearance, *e.g.* the digits keys, "Stop", "Correct" and "OK" keys and the management key "standalone". Some terminals of that model have three buttons right on the top of the digit keyboard, some do not [53]. In the application the buttons are presented. The left button "standalone" is used to start a payment transaction, then the other two buttons become active to navigate up and down

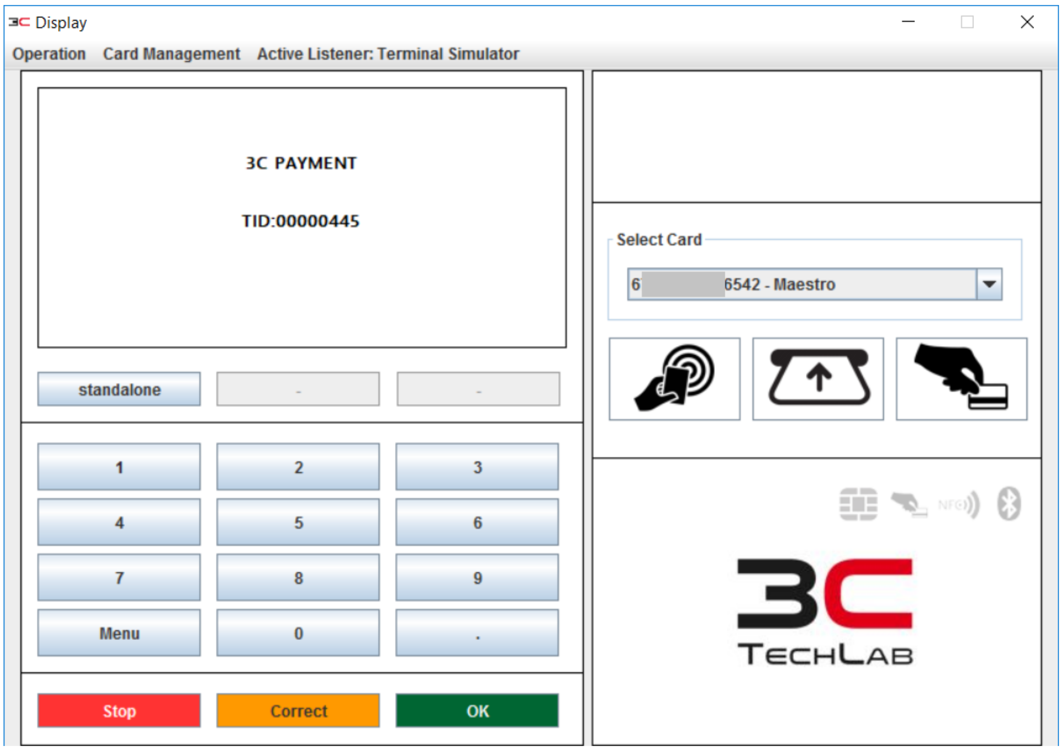


Figure 5.1: The screen of the initial state of 3C Emulator

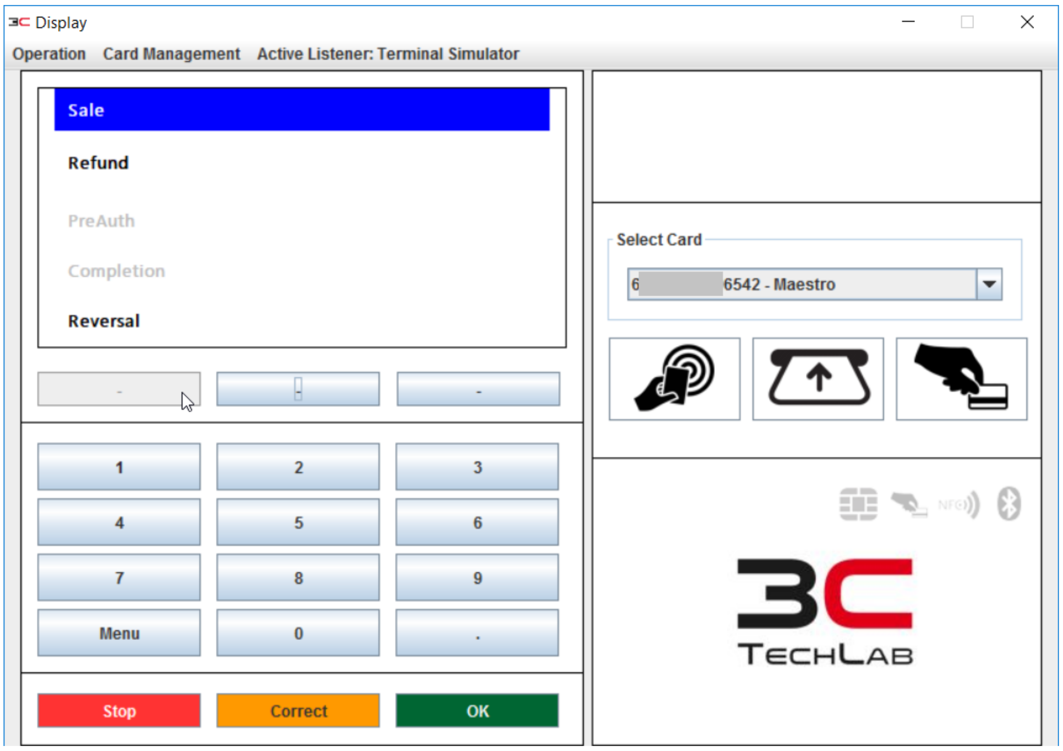
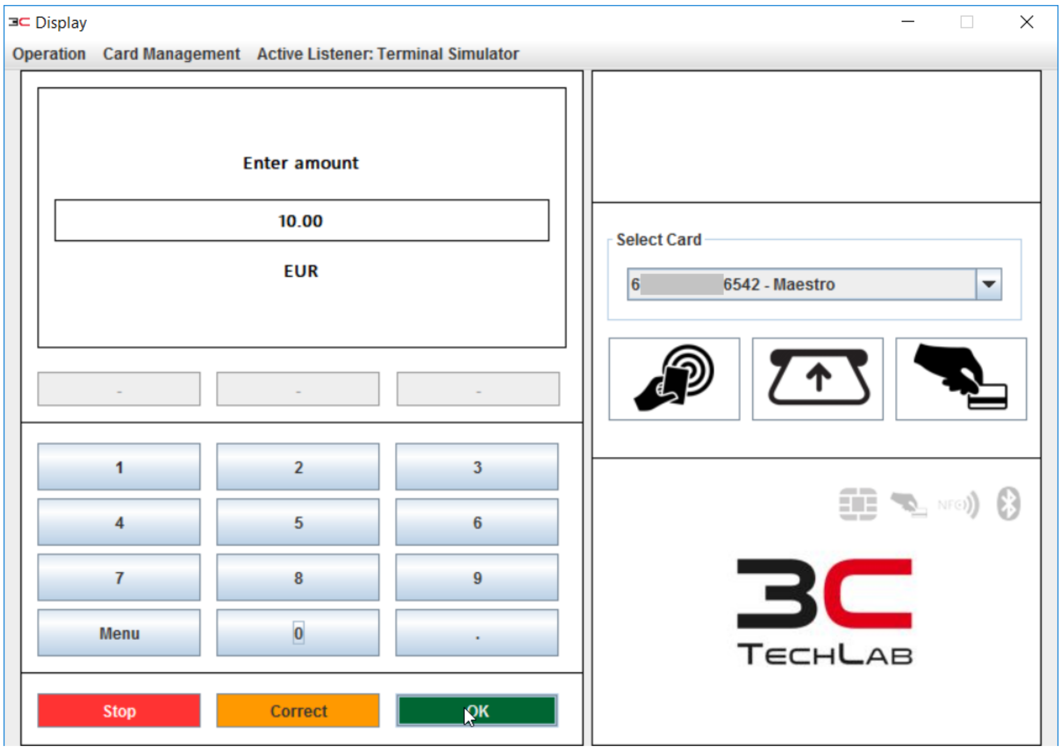


Figure 5.2: The screen for selecting the payment transaction type



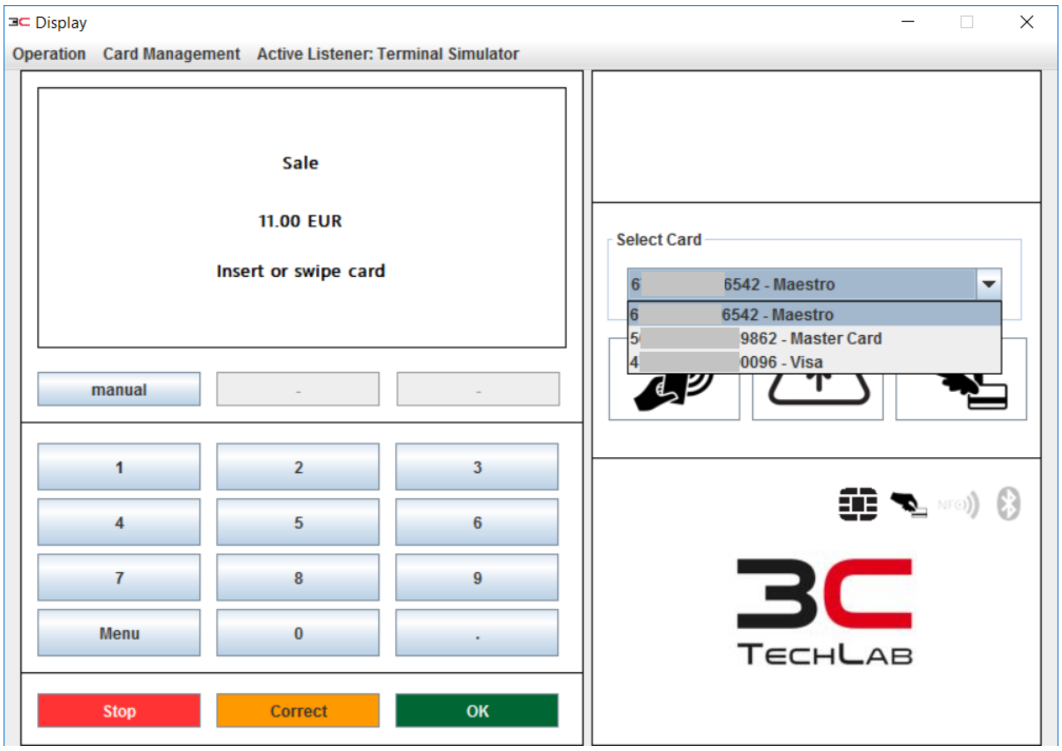
Enter amount

10.00

EUR

6[redacted]6542 - Maestro

Figure 5.3: The screen for the input the transaction amount



Sale

11.00 EUR

Insert or swipe card

6[redacted]6542 - Maestro

6[redacted]6542 - Maestro

5[redacted]9862 - Master Card

4[redacted]0096 - Visa

Figure 5.4: The screen for selecting the card and the method of the payment

in the standalone menu. Moreover, the same standalone menu can be opened (in 3C Emulator and real terminals) via the Menu button.

The right panel provides the cardholder interaction actions. In the top panel the user selects the card from the dropdown menu. Then the user selects the way of terminal and card interaction (via ICC chip, via magnetic stripe, contactless NFC or Bluetooth ways). In the current state the application supports two interaction ways – via ICC chip and via magnetic stripe. The text on the display shows the possible interaction actions, namely “Insert or swipe card”.

The bottom panel shows the company logo and four icons of available readers. Figure 5.1 illustrates all four readers in a disabled state, however in Figure 5.4 two readers are enabled. When the transaction requires providing the card from the user, the icons become more intense to inform what terminal features are enabled.

5.2.1 Payment Transaction

To start payment transaction, the user clicks on the standalone button (or opens the standalone list by clicking on the Menu button, *e.g.* Menu → Standalone). Then the user selects the transaction business type. Currently, 3C Emulator supports only three types, namely Sale, Refund and Reversal, as illustrated on Figure 5.2. More transaction types can be found in Table 2.2 and explanation in Section 2.2.3.

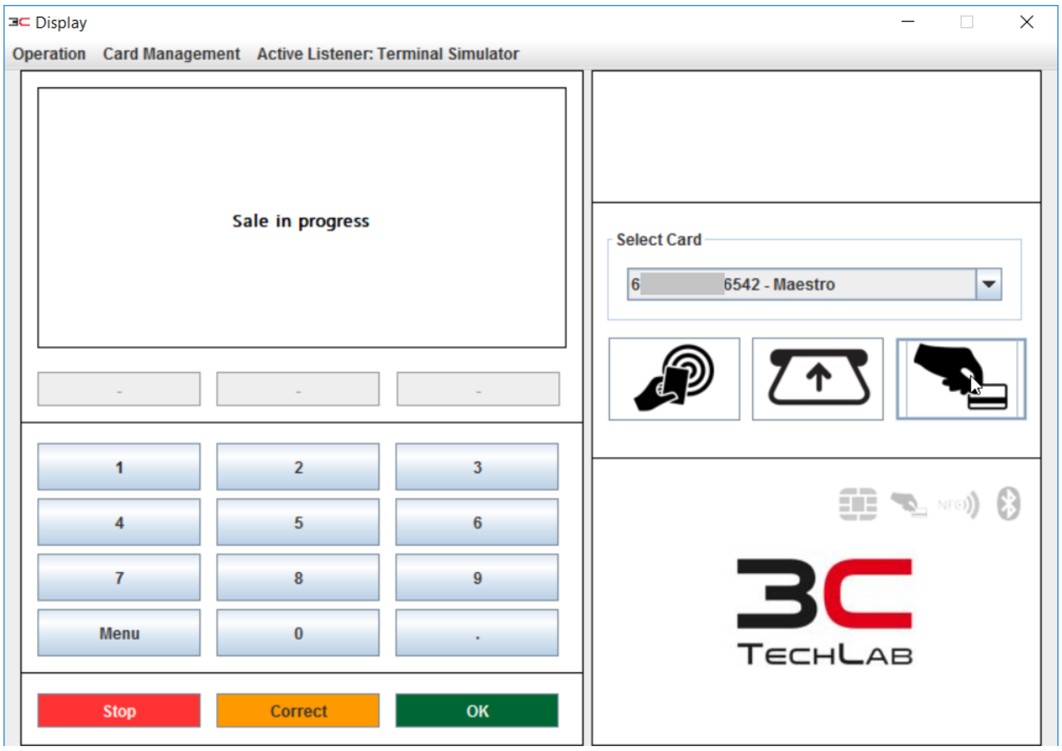
From that stage on all business types require the same terminal interaction, considered of: enter the transaction amount, request a card, pack the card data into ISO format message, send the message to the payment server, show the processing result, and prepare receipt. Depending on the card data, the terminal can request to enter a PIN code, however, for demonstration we use the simplest flow without PIN entering.

To illustrate the flow suppose that the user clicks on Sale to process a sale transaction. The terminal starts by requesting the merchant to enter the transaction amount (Figure 5.3). Currently, 3C Emulator is configured with the default currency EUR, however, the list of currencies is stored in the application configuration file, so that the default currency can be changed before running the application. All currencies, supported in 3C Emulator, are listed in the Appendix A.1.

3C Emulator produces a beep sound in the way the real terminal does. Before input of the amount the application signals the user that an input is required. Then, a second beep is produced when the cardholder needs to provide the card. The last beep sounds when the transaction is processed (with an error or with success). The last beep signals the user to take a receipt, however, in 3C Emulator there is no external device connected, so the beep is used only for a simulation reason.

After the amount value is entered, 3C Emulator requests cardholder action and the application is ready to interact with card data (Figure 5.4). As was mentioned above, currently 3C Emulator only supports simulation of two card reader modes – ICC and magnetic stripe readers. The corresponding text message is shown on the display.

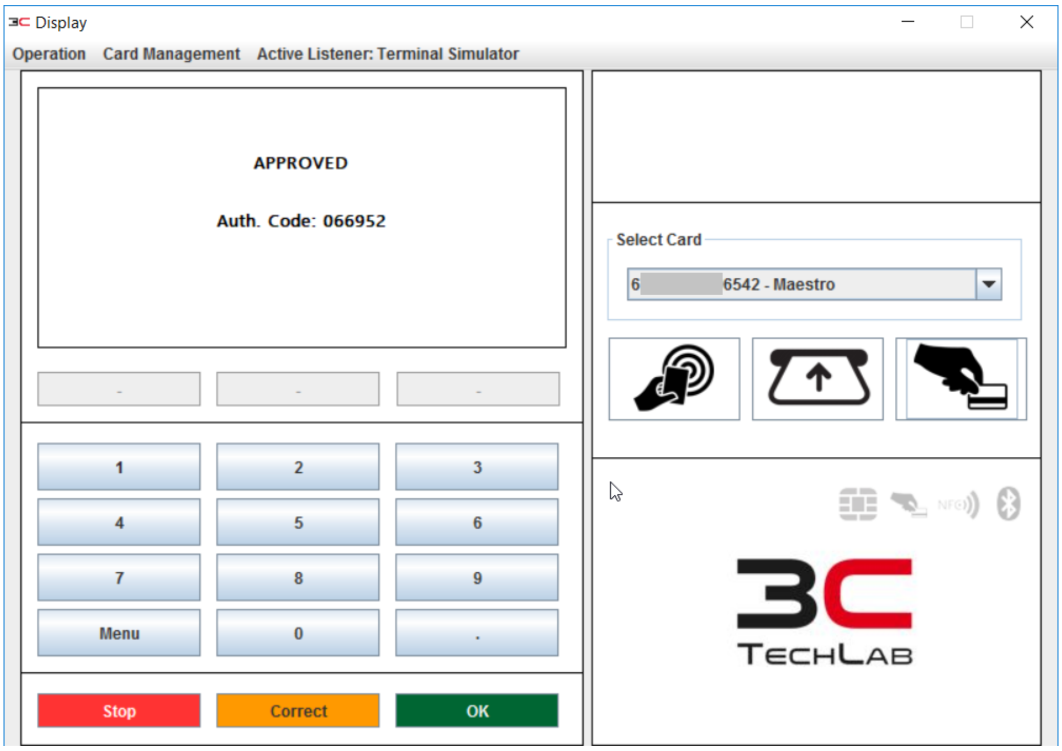
Then the user selects the card form dropdown menu. By default, the application has three test cards saved, which are Maestro, Mastercard and Visa cards. Each test card has preconfigured



Sale in progress

6 [redacted] 6542 - Maestro

Figure 5.5: The screen of the payment process

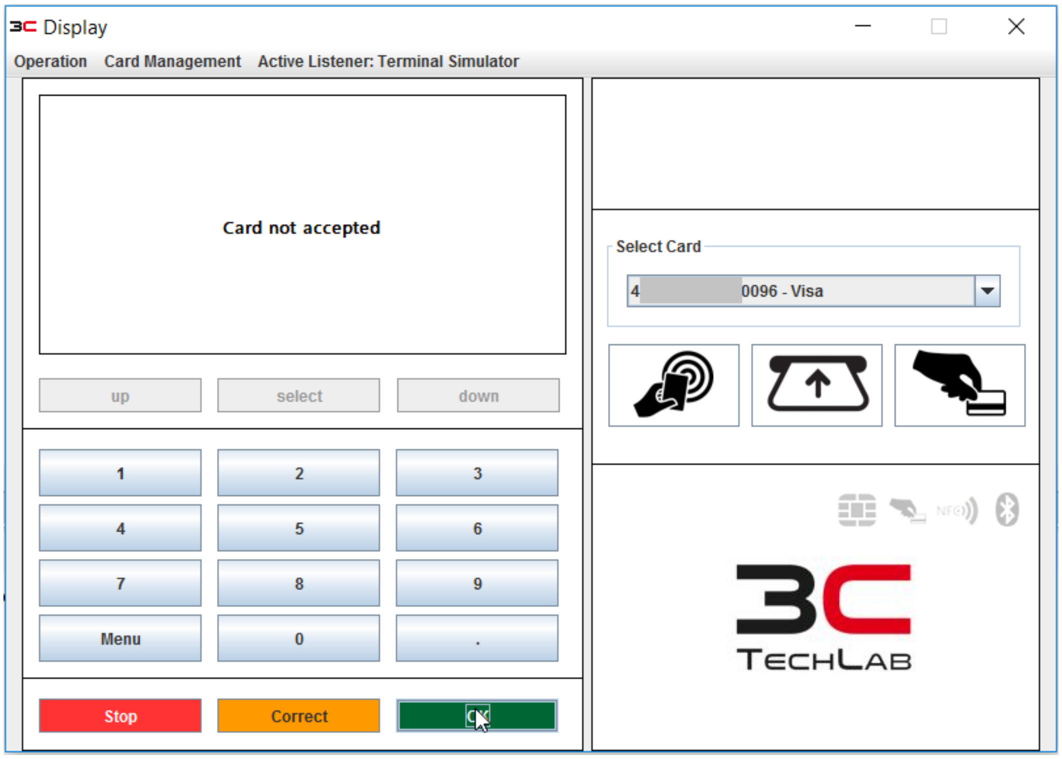


APPROVED

Auth. Code: 066952

6 [redacted] 6542 - Maestro

Figure 5.6: The screen of the payment result with success



Card not accepted

Select Card: 4[redacted]0096 - Visa

Figure 5.7: The screen of the payment result with a card error

settings, *e.g.* Track 2 data, ICC data, public key certificates, *etc.*, so that no real card is needed to use the application. To extend the list of test cases, one can add any card with the application menu Card Management → Open Card Management Window. The detailed explanation how to add a new test card is in Subsection 5.2.4.

When the card is selected from the dropdown menu, the user selects the preferred interaction way: contactless, insert card or swipe. If the test card is configured properly to use the selected interaction way, the processing of the transaction is started (Figure 5.5).

To process the transaction, the 3C Emulator application builds the ISO message and passes it to the payment server using TLS protocol. The server communicates with the acquirer network, and the acquirer bank processes the transaction (3C Emulator is configured to communicate with an UAT server, which in turn is connected to the acquirer test network). The payment server replies with the response message that also has the ISO format. Then the response message is parsed by the application and the response status is shown to the user in the display. If the transaction has been processed successfully, the display shows the text "APPROVED" and the authorisation code, that was returned from the acquirer bank (Figure 5.6). In the case of error during the payment, the "DECLINED" text is shown. If the card is configured in the wrong way, the display shows the "Card not accepted" message (Figure 5.7).

5.2.2 Application Menu

In addition to the main terminal interface, the application has the areas for the application configuration. The menu bar in the top of the application window contains three options, which in their turn have also sup-options, as listed next:

- → Operation
  - → Execute Operation
- → Card Management
  - → Open Card Management Window
- → Active Listener
  - → Terminal Simulator
  - → Socket
    - → to receive Operations
    - → to send Operations
  - → App

5.2.3 Execute Operations

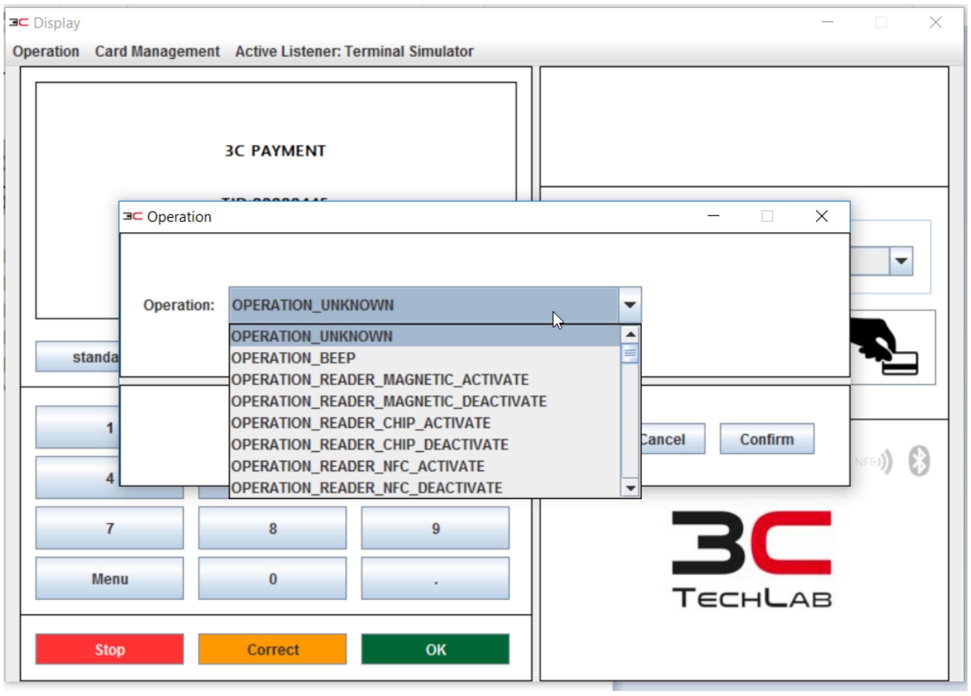
The application is built in such a way, that all application operations are independent and can be run separately. This was achieved by using the pattern Observer (Section 5.5). In the window Execute Operation the user can run any operation from the list, which is contained in the dropdown menu of the window (Figure 5.8). The full list of operations is presented in Appendix A.2.

The user can select one operation from the list, *e.g.* OPERATION_BEEP and click on Confirm button. As the result, the application makes a sound and the window shows the informational message "Operation Secceeded". Inactive terminal readers, such as Bluetooth and NFC, can be activated via the Operation menu.

5.2.4 Test Card Configuration

As was mentioned above, the dropdown menu with test cards is configurable. A test card can specified in three different ways:

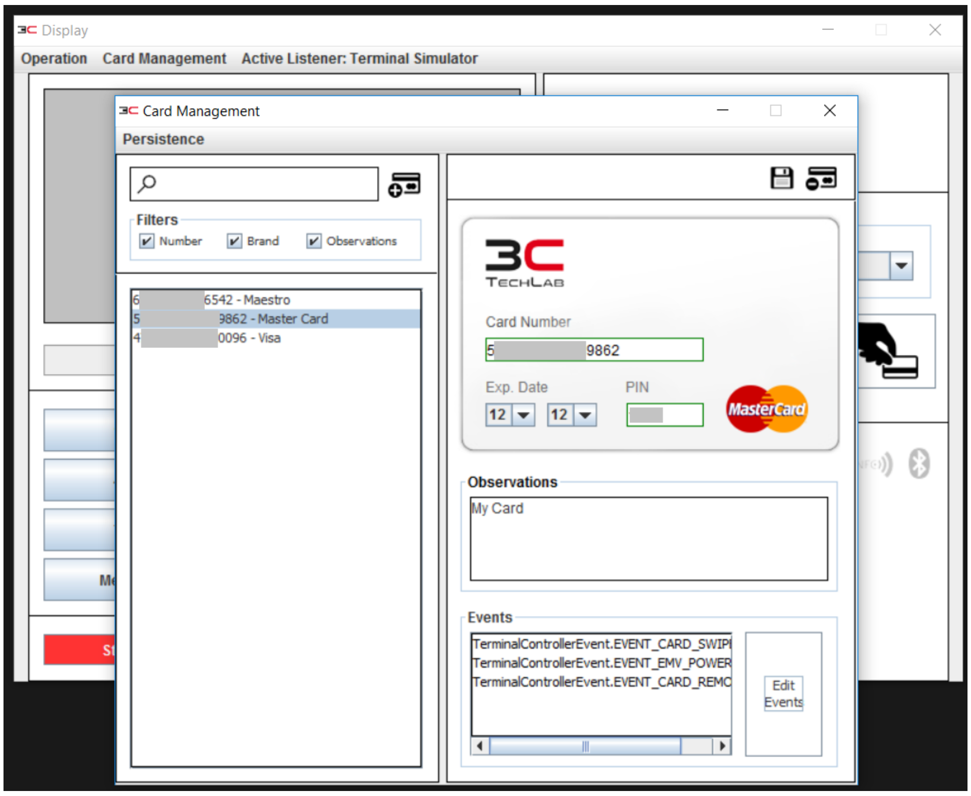
1. in the configuration file before running the application;
2. in the application database;
3. in the application window using the menu.



Operation: OPERATION_UNKNOWN

```
OPERATION_UNKNOWN
OPERATION_BEEP
OPERATION_READER_MAGNETIC_ACTIVATE
OPERATION_READER_MAGNETIC_DEACTIVATE
OPERATION_READER_CHIP_ACTIVATE
OPERATION_READER_CHIP_DEACTIVATE
OPERATION_READER_NFC_ACTIVATE
OPERATION_READER_NFC_DEACTIVATE
```

Figure 5.8: The screen of the Execute Operation window



Filters: [✓ Number] [✓ Brand] [✓ Observations]

```
6[redacted]6542 - Maestro
5[redacted]9862 - Master Card
4[redacted]0096 - Visa
```

Card Number: 5[redacted]9862

Exp. Date: 12 12

PIN: [redacted]

Observations: My Card

Events:

```
TerminalControllerEvent.EVENT_CARD_SWIPI
TerminalControllerEvent.EVENT_EMV_POWER
TerminalControllerEvent.EVENT_CARD_REMC
```

Figure 5.9: The screen of test cards data configuration

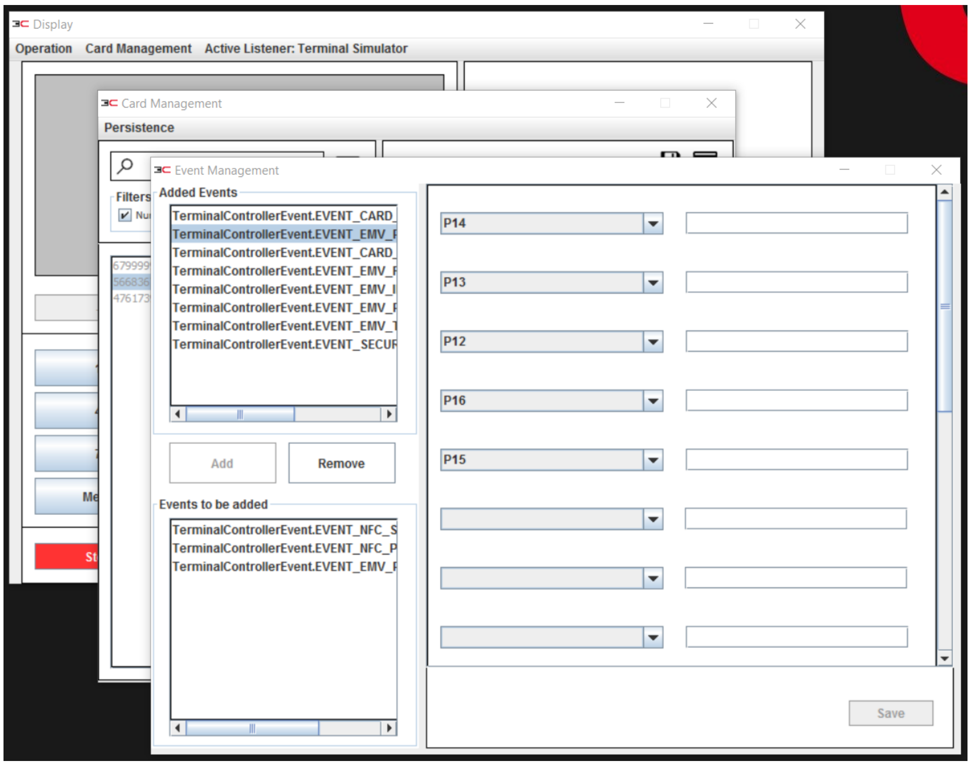

Display
Operation
Card Management
Active Listener: Terminal Simulator
Card Management
Persistence
Event Management
Added Events
TerminalControllerEvent.EVENT_CARD_
TerminalControllerEvent.EVENT_EMV_F
TerminalControllerEvent.EVENT_CARD_
TerminalControllerEvent.EVENT_EMV_F
TerminalControllerEvent.EVENT_EMV_I
TerminalControllerEvent.EVENT_EMV_F
TerminalControllerEvent.EVENT_EMV_T
TerminalControllerEvent.EVENT_SECUR
Add
Remove
Events to be added
TerminalControllerEvent.EVENT_NFC_S
TerminalControllerEvent.EVENT_NFC_P
TerminalControllerEvent.EVENT_EMV_F
P14
P13
P12
P16
P15
Save

Figure 5.10: The screen of the parameters of one test card

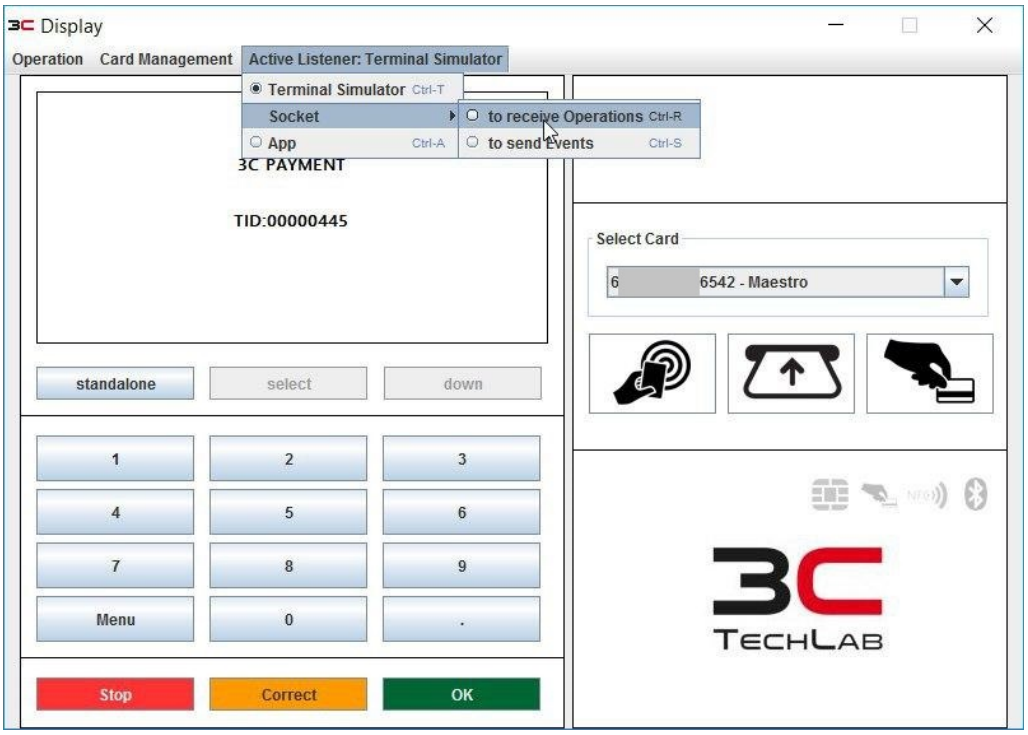


Figure 5.11: The menu of selecting Active Listener

The application GUI gives the possibility to the user to add, change, and delete test card data. To open the configuration window, the user can go to Card Management → Open Card Management Window (Figure 5.9). In the opened window, the user can add a new card by entering the card number (PAN), expiration date (month and year) and PIN.

Then to configure the card read capability, the user can add events (Figure 5.10). For instance, to mark the card as magnetic stripe, the user need to add the event EVENT_CARD_SWIPED and to mark as ICC – EVENT_EMV_POWERED_UP, EVENT_EMV_INITIALIZED, EVENT_CARD _REMOVED. The full list of events, that can be used to configure cards, can be found in Appendix A.3.

The events support a different number of parameters, that can be used during processing the event. For instance, Track 2 data, ICC data, Public key certificate data and so on.

And finally, the 3C Emulator application can be configured to not only emulate the real terminal (Active Listener → Terminal Simulator) (Figure 5.11). In addition, the application can also be used as non-connected to the UAT server application (Active Listener → App) in a way to run transaction without making workload on the server. Moreover, the 3C Emulator active features can be reduced to only send action requests (Active Listener → Socket → to receive Operations) or only receive the action requests (Active Listener → Socket → to send Events).

Despite the GUI being already developed by 3C TechLab company, a number of improvements were also included for the dissertation work. The reason is that the existing interface did not satisfy the requirements of the business logic development.

However, the interface is planed to be improved, since it does not have some useful features, *e.g.* selecting other terminal types appearance, selecting the payment server to process the transactions without application restart, masking sensitive authentication data such as PAN, PIN, ICC data and so on, making simulation of receipt print more clear, and other GUI features. There is still a room to improve the interface.

5.3 Business Logic Implementation

The application implements the following steps of the EMV payment flow: Data Authentication, Cardholder Verification, Terminal Risk Management, Terminal Action Analysis, Offline Authentication. Most of the steps is impossible emulate without having physical terminal, for instance, Terminal Risk Management. In such cases the facilitated solution was selected.

In the case of ICC payment, the data needed for Risk Management, Terminal Action Analysis and others are passed using parameters. The values for the parameters were taken from the real card, so that the application just check its presence and verify if the format is correct. All the steps of the EMV flow were created, but the behavior of 3C Emulator hardly resembles the processing inside a real PoS terminal. The application still has room for improvement and is included in the future work plan.

```java
// Start of heading
public static final char SOH  = '\u0001';
// Start of text
public static final char STX  = '\u0002';
// End of text
public static final char ETX  = '\u0003';
// End of transmission
public static final char EOT  = '\u0004';
// Synchronous idle
public static final char SYN  = '\u0016';
// End of transmission block
public static final char ETB  = '\u0017';
// Field separator
public static final char FS   = '\u001C';
// Data link escape
public static final char DLE  = '\u0010';
```

Listing 5.1: The code example that implements the part of IntegraFE connection

5.3.1 "T = 1" Protocol

The transmission of the messages is done using the "T = 1" protocol, meaning that the message is passed with the blocks. The block frames are marked with special characters, whose values are stored in a **Const** class presented in Listing 5.1. When the transaction message is ready to be sent, the method **send** from **DatalinkPayAtTable** is called. The method wrap the message in the expected format and send it (Listing 5.2).

5.3.2 CDA Algorithm

The application supports only the CDA algorithm of authentication, realised in a simplified way. The data that is needed lies in tag "95" in the bit 55 of the message. This data is passed as the parameter in the test card data and can be entered in the Card Management Window. The authentication data can not be produced without a real card, so the best way to implement the flow was to use a real card value in the test card data. However, Listing 5.3 illustrates the code from the **BrickEmvTransactionProcess**, which is the beginning of processing EMV transaction, and one of the steps is checking the terminal capability data. If the terminal supports the same authentication algorithm, the transaction is continued.

5.3.3 Cardholder Verification

In order to verify the cardholder a number of verification methods can be applied. For example, using a signature, or processing PIN verification offline or online. At the current stage, the application supports only offline PIN and signature verification. Since in the development and certification process there is no need to verify the signature, from the user point of view the transaction is

```java
private int send(String szData, int nChannelId, int nMessageSequenceNumber, boolean
     bRetransmission, int nMessageId) {
    String szApplicationSequence = null;
    String szApplicationData     = null;
    String szCheckSum            = null;
    String szMessage = null;
    try {
      szApplicationSequence = String.valueOf(nMessageSequenceNumber);
      while (szApplicationSequence.length() < 2) {
        szApplicationSequence = "0" + szApplicationSequence;
      }
      if (bRetransmission == false) {
        // add the retransmission flag
        szApplicationSequence = szApplicationSequence + " ";
      } else {
        szApplicationSequence = szApplicationSequence + "R";
      }

      // calculation of the Application Data
      szApplicationData = szData;

      szMessage = this.getHeader() +
                        String.valueOf(Const.STX) +
                        String.valueOf(Const.FS) +
                        szApplicationSequence +
                        szApplicationData +
                        String.valueOf(Const.ETX);

      // calculation of the CheckSum
      szCheckSum = "";
      // build the entire message
      szMessage = String.valueOf(Const.SOH) +
                        szMessage +
                        szCheckSum +
                        String.valueOf(Const.EOT);

      if (log.isDebugEnabled() == true) {
          log.debug("Sending data to ChannelId " + nChannelId);
      }
      if (log.isTraceEnabled() == true) {
          log.trace("Data being sent: \"" + HexConverter.stringToHexForDebug(
              szMessage) + "\"");
      }
      synchronized (mySemaphorDatalinkController) {
              if (this.myDatalinkController != null && this.myDatalinkController.
                  getChannel() != null) {
                    return this.myDatalinkController.getChannel().send(szMessage,
                        nChannelId, nMessageId);
              } else {
                return Const.ERROR;
              }
           }
    } catch (Exception exc) {
      return Const.ERROR;
    }
  }
```

Listing 5.2: The code example that implements the part of IntegraFE connection

```java
this.setState(STATE_EMV_TRANSACTION_PROCESS_WAIT, log);

this.displayScreenPleaseWait();

myTerminalControllerOperation = new TerminalControllerOperation(
    TerminalControllerOperation.OPERATION_EMV_PROCESS);
myTerminalControllerOperation.getData().insertTag(this.myData, Data.
    TAG_EMV_CERTIFICATION_AUTHORITY_PUBLIC_KEY_INDEX);
myTerminalControllerOperation.getData().setTagValueBoolean(Data.
    TAG_CUS_PROPRIETARY_MERCHANT_FORCES_ONLINE, false);
myTerminalControllerOperation.setTaggedObject(TerminalControllerOperation.
    TAGGED_OBJECT_EMV_APPLICATION_LIST, this.myTerminalHandlerStatus.
    getCardApplicationList());

if ((HelperOffline.getInstance() != null) &&
    (HelperOffline.getInstance().isOfflineActive() == false) &&
    (HelperOffline.getInstance().isForceOnlineFirstActive() == true)) {
    myTerminalControllerOperation.getData().setTagValueBoolean(Data.
        TAG_CUS_PROPRIETARY_APPLICATION_FORCES_ONLINE, true);
} else {
    myTerminalControllerOperation.getData().setTagValueBoolean(Data.
        TAG_CUS_PROPRIETARY_APPLICATION_FORCES_ONLINE, false);
}

if ((this.myTerminalHandlerStatus != null) &&
    (this.myTerminalHandlerStatus.getCardApplicationList() != null) &&
    (this.myTerminalHandlerStatus.getCardApplicationList().getCardApplication(0) !=
         null) &&
    (this.myTerminalHandlerStatus.getCardApplicationList().getCardApplication(0).
        getTerminalCapabilities() != null)) {

    String szTerminalCapabilitiesByte3 = this.myTerminalHandlerStatus.
        getCardApplicationList().getCardApplication(0).getTerminalCapabilities().
        substring(4, 6);
    int nTerminalCapabilitiesByte3Bit4 = Integer.parseInt(
        szTerminalCapabilitiesByte3, 16) >> 3;

    if (nTerminalCapabilitiesByte3Bit4 % 2 != 0) {
       myTerminalControllerOperation.getData().setTagValueBoolean(Data.
           TAG_CUS_PROPRIETARY_REQUEST_CDA, true);
    }
}
this.myTerminalController.doOperation(myTerminalControllerOperation);
```

Listing 5.3: The code example that implements the part of IntegraFE connection

```
    public void setCardholderVerificationMethod(String
        szCardholderVerificationMethod) {

        if (StringUtils.equalsIgnoreCase(CVM_UNKNOWN,
            szCardholderVerificationMethod) == true) {
            this.szCardholderVerificationMethod = szCardholderVerificationMethod;
        } else {

            if (((StringUtils.equalsIgnoreCase(CVM_OFFLINE_PIN, this.
                szCardholderVerificationMethod)      == true) &&
               (StringUtils.equalsIgnoreCase(CVM_SIGNATURE_BASED,
                   szCardholderVerificationMethod)       == true)) ||
                ((StringUtils.equalsIgnoreCase(CVM_SIGNATURE_BASED, this.
                    szCardholderVerificationMethod)  == true) &&
                 (StringUtils.equalsIgnoreCase(CVM_OFFLINE_PIN,
                     szCardholderVerificationMethod)         == true))
                ) {
                this.szCardholderVerificationMethod =
                    CVM_SIGNATURE_BASED_OFFLINE_PIN;
            } else if (((StringUtils.equalsIgnoreCase(CVM_ONLINE_PIN, this.
                szCardholderVerificationMethod)      == true) &&
                  (StringUtils.equalsIgnoreCase(CVM_SIGNATURE_BASED,
                      szCardholderVerificationMethod)    == true)) ||
                     ((StringUtils.equalsIgnoreCase(CVM_SIGNATURE_BASED, this.
                         szCardholderVerificationMethod) == true) &&
                      (StringUtils.equalsIgnoreCase(CVM_ONLINE_PIN, this.
                          szCardholderVerificationMethod)    == true))) {
                this.szCardholderVerificationMethod =
                    CVM_SIGNATURE_BASED_OFFLINE_PIN;
            } else {
                this.szCardholderVerificationMethod =
                    szCardholderVerificationMethod;
            }
        }
    }
```

Listing 5.4: The code example that implements the part of IntegraFE connection

processed without any verification and we provide the space for the signature in the receipt. This was illustrated in Section 5.2 while presenting the interface of the application.

The restriction of cardholder verification is implemented in class **RequestStatusInformation** and the piece of code is presented in Listing 5.4.

5.3.4 External Service Evidence

As was described in Chapter 4, the application communicates with several external services. Here we briefly explain the mechanism of communication with the IntegraFE server.

Before sending, the message is built following the 3Cxml structure. The message is then converted into a String value and passed to the *DatalinkController* class, together with the *channelId*

```
public int send(String szData, int nMessageId) {
    try {

        if (log.isDebugEnabled() == true) {
            log.debug("Send data called, data: \"...\"");
        }
        if (log.isTraceEnabled() == true) {
            log.trace("Send data called, data: " + szData);
        }

        // Generate response
        HTTPResponse myResponse = new HTTPResponse(myHeaderLines);

        if (nHTTPError == HTTPConst.kSuccess) {
            myResponse.setHost((String) myHeaderLines.get("Host"));
            myResponse.setURL(HTTPConst.decodeURL(szURL));
            myResponse.setQuery(szQuery);
        }

        myResponse.setStatusCode(nHTTPError);
        int nRet = myResponse.sendStringResponse(myCWriter, szData);
        if (nRet == Const.NO_ERROR) {
            this.myParent.propagateChannelEventDataSent(nMessageId);
        }

        // stop the waiting loop - next step will have to be disconnect
        this.bDataSent = true;

        return nRet;

    } catch (Exception exc) {
        if (log.isWarnEnabled() == true) {
            log.warn("Exception occured while sending data: " + exc, exc);
        }
        return Const.ERROR;
    }
}
```

Listing 5.5: The code example that implements the part of IntegraFE connection

value (the value is incremented in order to provide multithreaded processing on the same connection) and the *messageId* (also an auto-incremented value, that is used to identify the received response for the sent message). The code example can be found in Listing 5.5.

After, the protocol is selected based on the configuration file of the application. We consider the case of a simple HTTP request, that is implemented in the class **HTTPRequest.java**. In that class the HTTP request is prepared.

As the result, the request reaches IntegraFE server, which in turn prepares the message in the ISO8583 format to the NAS server. Figure 5.12 illustrates the log of the transaction request and Figure 5.13 – the corresponding transaction response, that has reached the NAS server. The transaction was processed in the BNKSM processor, so that there is no need to mask much data, however, due to security reasons, we hide sensitive data (PAN and other data) in the log screenshot.

5.3.5 Certification Authority

The implementation of the connection to the Certification Authority, obtaining its public keys, retrieving original data from signed certificate is implemented in the **IntegraTE_bouncycastle** project. As an example we can consider just a small part, building the request to Certification Authority location. This is implemented in **CertificateRequest** class, and part of the code is presented in Listing 5.6. The method encode the request using distinguished encoding rules.

5.3.6 Receipt Printing

The application can prepare a receipt after the transaction processing. Since the application does not have the printer connected, it prepares the receipt in text format and then save it in file. An example of approved sale transaction is presented in Figure 5.14.

The transaction was run on the acquirer network, and due to this, the data in the receipt is considered sensitive. However, the common image of the built receipt is given. Since the main part of the receipt is prepared on the acquirer side, on the application side we need to parse it and save the text message in the file. The piece of code of parsing the acquirer receipt data is presented in Listing 5.7

5.3.7 Multi-language Support

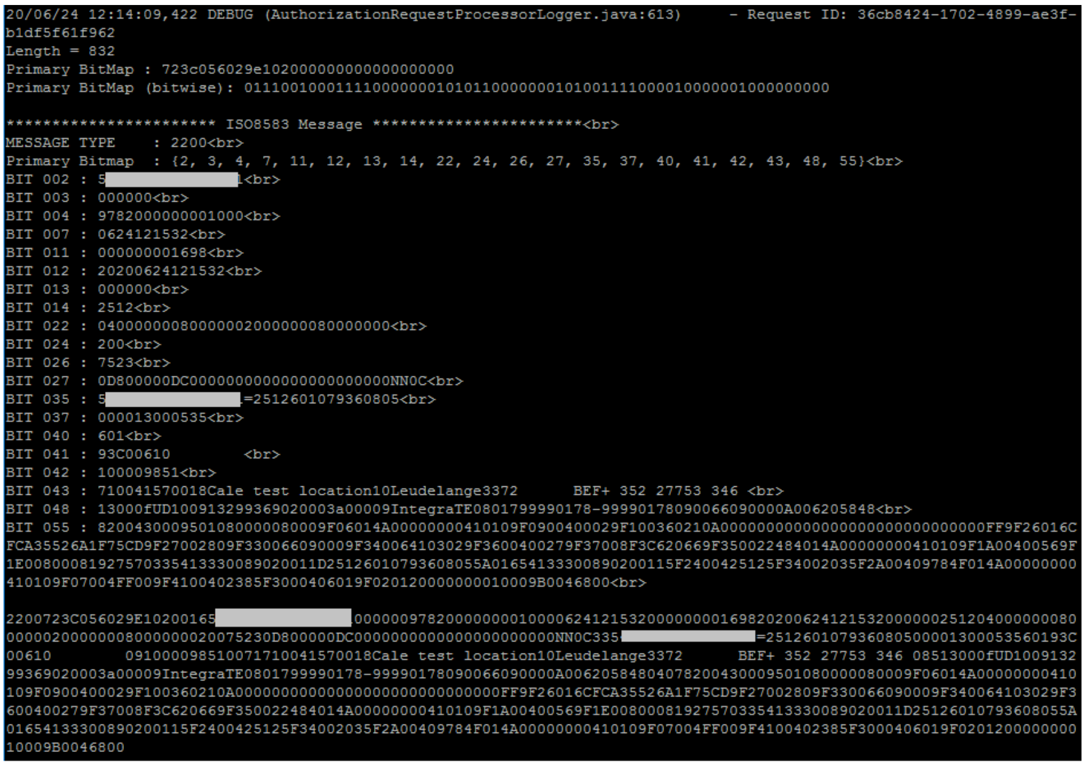
The application supports different languages in its GUI. To achieve this, set of resource files was added (Figure 5.15). Since 3C TechLab already has merchants all over the world, the translation of static text values was already done. Currently, the application supports 23 languages, and English is selected as the default one. The translation of the application context following the selected language is done using standard Java library *ResourceBundle*. The method *getBundle* gets a resource bundle using the specified base name, locale, and class loader. If in the resource file any key is missing, the application will use the key and its value from the default language resource file, which is *resources.properties*.

```java
public void encode(OutputStream output) throws IOException {
        if (certificateTypes == null || certificateTypes.length == 0) {
            TlsUtils.writeUint8(0, output);
        } else {
            TlsUtils.writeUint8ArrayWithUint8Length(certificateTypes, output);
        }

        if (supportedSignatureAlgorithms != null) {
            TlsUtils.encodeSupportedSignatureAlgorithms(
                supportedSignatureAlgorithms, false, output);
        }

        if (certificateAuthorities == null || certificateAuthorities.isEmpty()) {
            TlsUtils.writeUint16(0, output);
        } else {
            Vector derEncodings = new Vector(certificateAuthorities.size());

            int totalLength = 0;
            for (int i = 0; i < certificateAuthorities.size(); ++i) {
                X500Name certificateAuthority = (X500Name)certificateAuthorities.
                    elementAt(i);
                byte[] derEncoding = certificateAuthority.getEncoded(ASN1Encoding.
                    DER);
                derEncodings.addElement(derEncoding);
                totalLength += derEncoding.length + 2;
            }

            TlsUtils.checkUint16(totalLength);
            TlsUtils.writeUint16(totalLength, output);

            for (int i = 0; i < derEncodings.size(); ++i) {
                byte[] derEncoding = (byte[])derEncodings.elementAt(i);
                TlsUtils.writeOpaque16(derEncoding, output);
            }
        }
    }
```

Listing 5.6: The code of encoding the certificate request

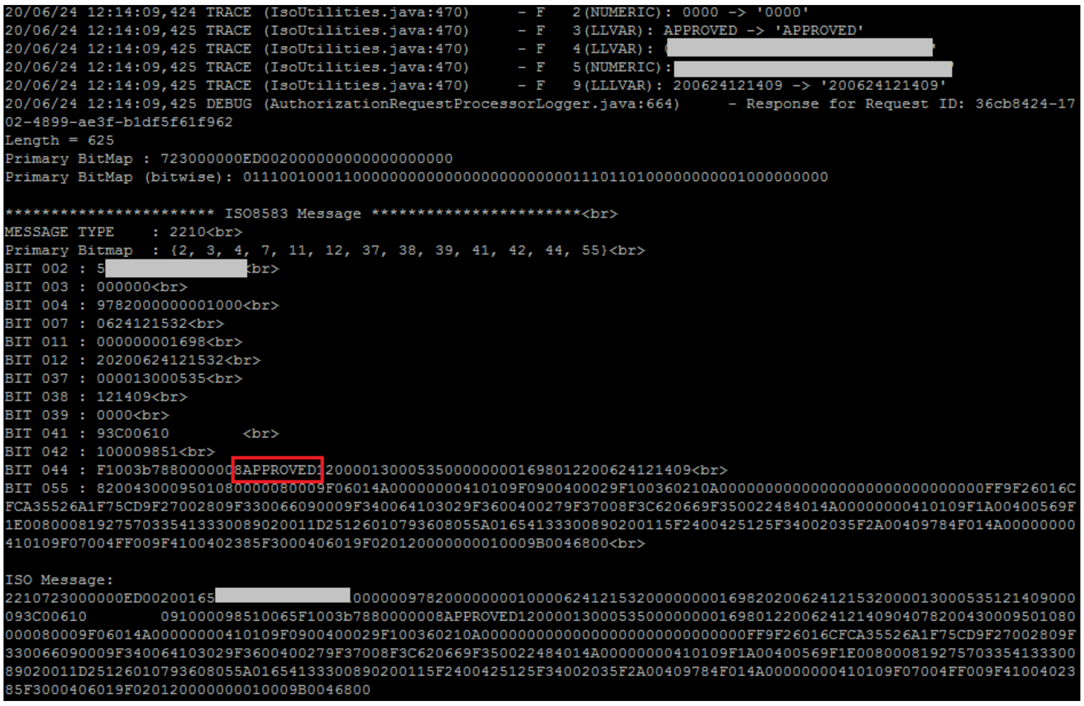
```java
public String createReceipt(String szPrintoutDefinition) {

    String szResult = null;
    StringBuffer myStringBufferReceipt = null;

    XMLElement xml = null;
    StringReader myXMLReader = null;

    try {

      myStringBufferReceipt = new StringBuffer();

      if (StringUtils.isNullOrEmpty(szPrintoutDefinition, true) == true) {
            szResult = null;
            return szResult;
        }

        if (log.isDebugEnabled() == true) {
            log.debug("createForm from PrintoutDefinition: \n" +
                szPrintoutDefinition);
        }

        myXMLReader = new StringReader(szPrintoutDefinition);

        xml = new XMLElement();
        xml.parseFromReader(myXMLReader);

        if (myXMLReader != null) {
          myXMLReader.close();
          myXMLReader = null;
        }

        if ((xml.getName() == null) ||
          (StringUtils.equalsIgnoreCase(xml.getName(), DISPLAY_TAG_SCREEN) ==
              false)) {
            szResult = null;
            return szResult;
        }

        this.walkElement(xml, 0, myStringBufferReceipt, new StringBuffer(),
            false);

    } catch (Exception exc) {
        if (log.isErrorEnabled() == true) {
            log.error("Exception in createForm(Printout myPrintout, int
                nDisplay)", exc);
        }
        szResult = null;
        return szResult;
    }

    return myStringBufferReceipt.toString();
}
```

Listing 5.7: The code that implements the part of preparing the receipt



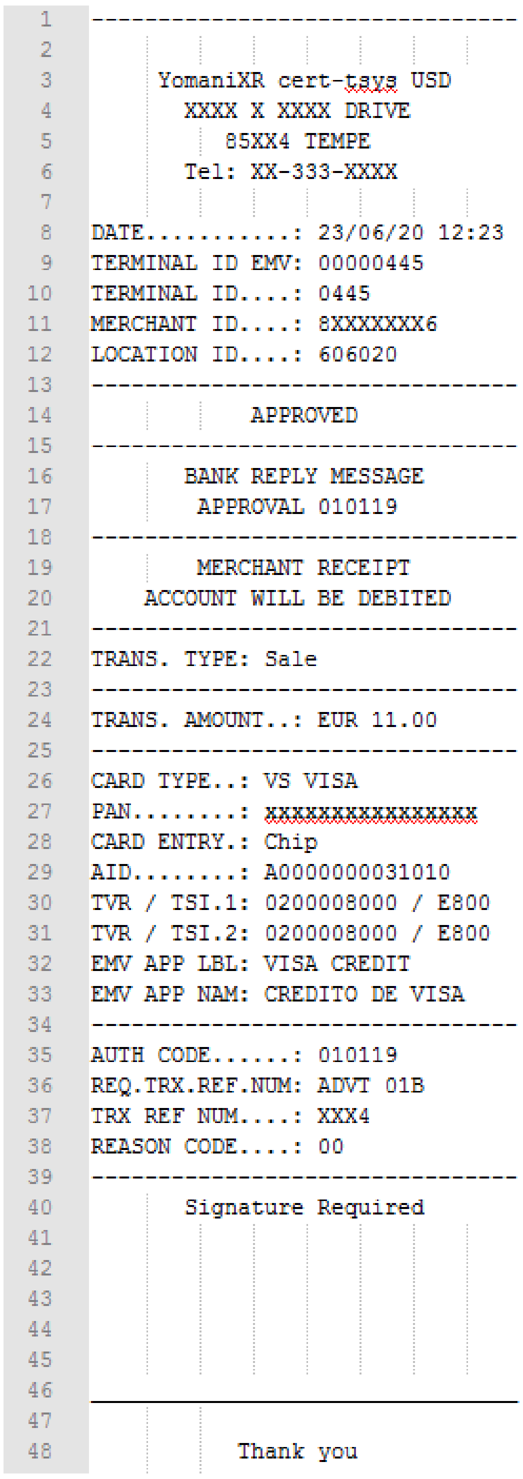
```
20/06/24 12:14:09,422 DEBUG (AuthorizationRequestProcessorLogger.java:613)     - Request ID: 36cb8424-1702-4899-ae3f-
b1df5f61f962
Length = 832
Primary BitMap : 723c056029e10200000000000000000
Primary BitMap (bitwise): 0111001000111100000001010110000000101001111000010000001000000000

*********************** ISO8583 Message ***********************<br>
MESSAGE TYPE    : 2200<br>
Primary Bitmap  : {2, 3, 4, 7, 11, 12, 13, 14, 22, 24, 26, 27, 35, 37, 40, 41, 42, 43, 48, 55}<br>
BIT 002 : 5[redacted]1<br>
BIT 003 : 000000<br>
BIT 004 : 9782000000001000<br>
BIT 007 : 0624121532<br>
BIT 011 : 000000001698<br>
BIT 012 : 20200624121532<br>
BIT 013 : 000000<br>
BIT 014 : 2512<br>
BIT 022 : 040000000800000020000000080000000<br>
BIT 024 : 200<br>
BIT 026 : 7523<br>
BIT 027 : 0D800000DC000000000000000000000NN0C<br>
BIT 035 : 5[redacted]=2512601079360805<br>
BIT 037 : 000013000535<br>
BIT 040 : 601<br>
BIT 041 : 93C00610        <br>
BIT 042 : 100009851<br>
BIT 043 : 710041570018Cale test location10Leudelange3372      BEF+ 352 27753 346 <br>
BIT 048 : 13000fUD100913299369020003a00009IntegraTE0801799990178-99990178090066090000A006205848<br>
BIT 055 : 820043000950108000008009F06014A0000000410109F0900400029F100360210A000000000000000000000000000000FF9F26016C
FCA35526A1F75CD9F27002809F330066090009F340064103029F3600400279F37008F3C620669F350022484014A0000000410109F1A00400569F
1E0080008192757033541333008902001 1D2512601079360805 5A01654133300890200115F2400425125F34002035F2A00409784F014A00000000
410109F07004FF009F4100402385F3000406019F0201200000000010009B0046800<br>

2200723C056029E10200165[redacted]0000009782000000001000062412153200000000169820200624121532000000251204000000080
000002000000008000000020075230D800000DC000000000000000000000NN0C335[redacted]=25126010793608050000130005356019 3C
00610        0910000985100717100415700 18Cale test location10Leudelange3372      BEF+ 352 27753 346 08513000fUD1009132
99369020003a00009IntegraTE0801799990178-99990178090066090000A0062058480407820043000950108000008009F06014A0000000410
109F0900400029F100360210A000000000000000000000000000000FF9F26016CFCA35526A1F75CD9F27002809F330066090009F340064103029F3
600400279F37008F3C620669F350022484014A0000000410109F1A00400569F1E0080008192757033541333008902001 1D2512601079360805 5A
01654133300890200115F2400425125F34002035F2A00409784F014A00000000410109F07004FF009F4100402385F3000406019F0201200000000
10009B0046800
```

Figure 5.12: The screenshot of transaction request message in ISO8583 format



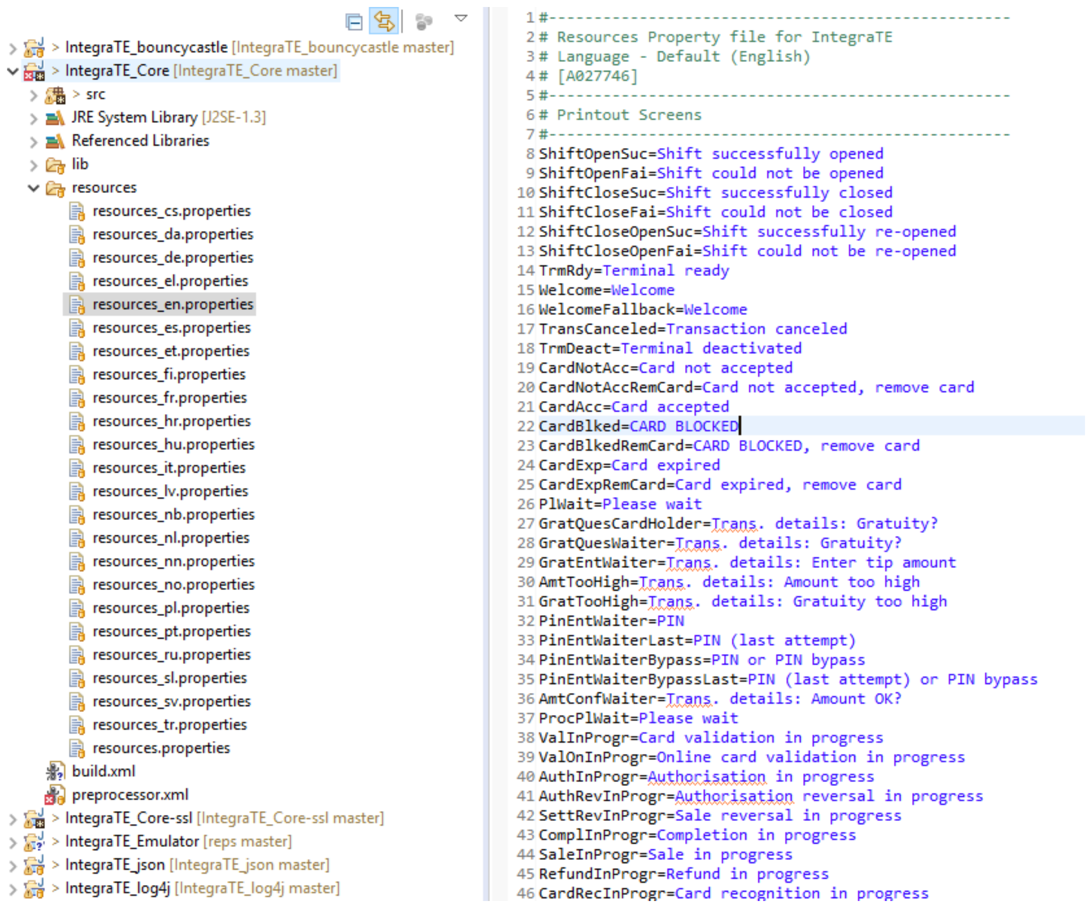
```
20/06/24 12:14:09,424 TRACE (IsoUtilities.java:470)     - F   2(NUMERIC): 0000 -> '0000'
20/06/24 12:14:09,425 TRACE (IsoUtilities.java:470)     - F   3(LLVAR): APPROVED -> 'APPROVED'
20/06/24 12:14:09,425 TRACE (IsoUtilities.java:470)     - F   4(LLVAR): [redacted]
20/06/24 12:14:09,425 TRACE (IsoUtilities.java:470)     - F   5(NUMERIC): [redacted]
20/06/24 12:14:09,425 TRACE (IsoUtilities.java:470)     - F   9(LLLVAR): 200624121409 -> '200624121409'
20/06/24 12:14:09,425 DEBUG (AuthorizationRequestProcessorLogger.java:664)     - Response for Request ID: 36cb8424-17
02-4899-ae3f-b1df5f61f962
Length = 625
Primary BitMap : 723000000ED00200000000000000000
Primary BitMap (bitwise): 0111001000110000000000000000000000001110110100000000000001000000000

*********************** ISO8583 Message ***********************<br>
MESSAGE TYPE    : 2210<br>
Primary Bitmap  : {2, 3, 4, 7, 11, 12, 37, 38, 39, 41, 42, 44, 55}<br>
BIT 002 : 5[redacted]<br>
BIT 003 : 000000<br>
BIT 004 : 9782000000001000<br>
BIT 007 : 0624121532<br>
BIT 011 : 000000001698<br>
BIT 012 : 20200624121532<br>
BIT 037 : 000013000535<br>
BIT 038 : 121409<br>
BIT 039 : 0000<br>
BIT 041 : 93C00610        <br>
BIT 042 : 100009851<br>
BIT 044 : F1003b7880000008APPROVED12000013000535000000001698012200624121409<br>
BIT 055 : 820043000950108000008009F06014A0000000410109F0900400029F100360210A000000000000000000000000000000FF9F26016C
FCA35526A1F75CD9F27002809F330066090009F340064103029F3600400279F37008F3C620669F350022484014A0000000410109F1A00400569F
1E0080008192757033541333008902001 1D2512601079360805 5A01654133300890200115F2400425125F34002035F2A00409784F014A00000000
410109F07004FF009F4100402385F3000406019F0201200000000010009B0046800<br>

ISO Message:
2210723000000ED00200165[redacted]0000009782000000001000062412153200000000169820200624121532000013000535121409000
093C00610        0910000985100 65F1003b7880000008APPROVED120000130005350000000016980122006241214090407820043000950108 0
000080009F06014A0000000410109F0900400029F100360210A000000000000000000000000000000FF9F26016CFCA35526A1F75CD9F27002809F
330066090009F340064103029F3600400279F37008F3C620669F350022484014A0000000410109F1A00400569F1E00800081927570335413330 0
89020011D2512601079360805 5A01654133300890200115F2400425125F34002035F2A00409784F014A00000000410109F07004FF009F41004023
85F3000406019F0201200000000010009B0046800
```

Figure 5.13: The screenshot of transaction response message in ISO8583 format



```
--------------------------------

     YomaniXR cert-tsys USD
       XXXX X XXXX DRIVE
          85XX4 TEMPE
       Tel: XX-333-XXXX

DATE...........: 23/06/20 12:23
TERMINAL ID EMV: 00000445
TERMINAL ID....: 0445
MERCHANT ID....: 8XXXXXXX6
LOCATION ID....: 606020
--------------------------------
           APPROVED
--------------------------------
       BANK REPLY MESSAGE
        APPROVAL 010119
--------------------------------
        MERCHANT RECEIPT
    ACCOUNT WILL BE DEBITED
--------------------------------
TRANS. TYPE: Sale
--------------------------------
TRANS. AMOUNT..: EUR 11.00
--------------------------------
CARD TYPE..: VS VISA
PAN........: XXXXXXXXXXXXXXXX
CARD ENTRY.: Chip
AID........: A0000000031010
TVR / TSI.1: 0200008000 / E800
TVR / TSI.2: 0200008000 / E800
EMV APP LBL: VISA CREDIT
EMV APP NAM: CREDITO DE VISA
--------------------------------
AUTH CODE......: 010119
REQ.TRX.REF.NUM: ADVT 01B
TRX REF NUM....: XXX4
REASON CODE....: 00
--------------------------------
       Signature Required






___________________________________

           Thank you
```

Figure 5.14: The example of the receipt after Sale transaction (with sensitive data masked)



```
> IntegraTE_bouncycastle [IntegraTE_bouncycastle master]
v > IntegraTE_Core [IntegraTE_Core master]
  > > src
  > JRE System Library [J2SE-1.3]
  > Referenced Libraries
  > lib
  v resources
      resources_cs.properties
      resources_da.properties
      resources_de.properties
      resources_el.properties
      resources_en.properties
      resources_es.properties
      resources_et.properties
      resources_fi.properties
      resources_fr.properties
      resources_hr.properties
      resources_hu.properties
      resources_it.properties
      resources_lv.properties
      resources_nb.properties
      resources_nl.properties
      resources_nn.properties
      resources_no.properties
      resources_pl.properties
      resources_pt.properties
      resources_ru.properties
      resources_sl.properties
      resources_sv.properties
      resources_tr.properties
      resources.properties
    build.xml
    preprocessor.xml
> > IntegraTE_Core-ssl [IntegraTE_Core-ssl master]
> > IntegraTE_Emulator [reps master]
> > IntegraTE_json [IntegraTE_json master]
> > IntegraTE_log4j [IntegraTE_log4j master]
```

```
#---------------------------------------------------
# Resources Property file for IntegraTE
# Language - Default (English)
# [A027746]
#---------------------------------------------------
# Printout Screens
#---------------------------------------------------
ShiftOpenSuc=Shift successfully opened
ShiftOpenFai=Shift could not be opened
ShiftCloseSuc=Shift successfully closed
ShiftCloseFai=Shift could not be closed
ShiftCloseOpenSuc=Shift successfully re-opened
ShiftCloseOpenFai=Shift could not be re-opened
TrmRdy=Terminal ready
Welcome=Welcome
WelcomeFallback=Welcome
TransCanceled=Transaction canceled
TrmDeact=Terminal deactivated
CardNotAcc=Card not accepted
CardNotAccRemCard=Card not accepted, remove card
CardAcc=Card accepted
CardBlked=CARD BLOCKED
CardBlkedRemCard=CARD BLOCKED, remove card
CardExp=Card expired
CardExpRemCard=Card expired, remove card
PlWait=Please wait
GratQuesCardHolder=Trans. details: Gratuity?
GratQuesWaiter=Trans. details: Gratuity?
GratEntWaiter=Trans. details: Enter tip amount
AmtTooHigh=Trans. details: Amount too high
GratTooHigh=Trans. details: Gratuity too high
PinEntWaiter=PIN
PinEntWaiterLast=PIN (last attempt)
PinEntWaiterBypass=PIN or PIN bypass
PinEntWaiterBypassLast=PIN (last attempt) or PIN bypass
AmtConfWaiter=Trans. details: Amount OK?
ProcPlWait=Please wait
ValInProgr=Card validation in progress
ValOnInProgr=Online card validation in progress
AuthInProgr=Authorisation in progress
AuthRevInProgr=Authorisation reversal in progress
SettRevInProgr=Sale reversal in progress
ComplInProgr=Completion in progress
SaleInProgr=Sale in progress
RefundInProgr=Refund in progress
CardRecInProgr=Card recognition in progress
```

Figure 5.15: The screenshot of language resources files in the IDE

5.4 Database

The standalone application 3C Emulator uses its own database, however, the data is extremely simple. Since the application does not actually process the payment transactions and the only task is to prepare the message in ISO format to transfer to the payment server, there is no need to use a complicate structure of tables and all the fields that can be defined in an actual terminal, in the database.

The application uses a SQLite database [47], namely to store only one file in *.db* format. When the data changes and the application is restarted, the database (the file in the application code directory) is changed, so that after restart the user sees the last changes he made.

For now, we found out that 3C Emulator needs only four tables (Figure 5.16):

1. **Card**, containing all test card information, that can be edited in the windows Card Management (Figure 5.8);

2. **DataElement**, used to store all parameters of the test card events, that can be edited in the Event Management window (Figure 5.9);

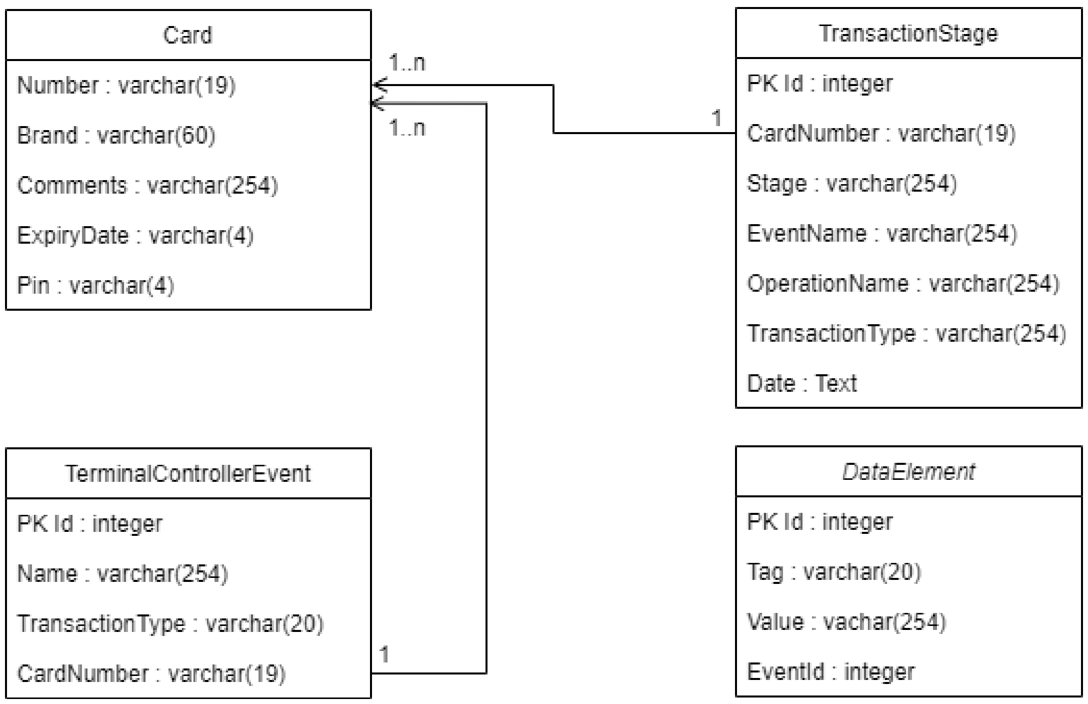


Figure 5.16: Class Diagram of database structure

3. **TerminalControllerEvent**, used to return to the last processed events;

4. **TransactionStage**, containing flags of passing the stage during transaction processing and serving as an additional log file.

5.5 Design Solutions

The application software is growing with any new change added. In the beginning of the project existing code repository already had frightening scale. One of the requirements of the project was to maintain readability of the application source code using best design practice and improve the project structure.

The application project is organised in a set of Java projects. Two of the most loaded projects are **IntegraTE_Core** and **IntegraTE_Emulator**. In order to improve code structure, design patterns were used. Here we provide five examples of design patterns applied in the project: Abstract Factory, Observer, Façade, Builder and Singleton

A design pattern is a general repeatable solution to a commonly occurring problem in software design. Patterns allow developers to communicate using well-known, well understood names for software interactions. A design pattern isn't a finished design that can be transformed directly into

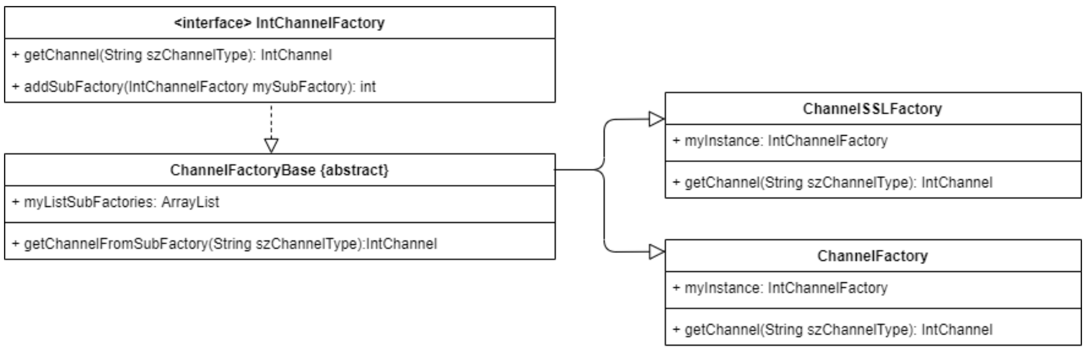

<interface> IntChannelFactory
+ getChannel(String szChannelType): IntChannel
+ addSubFactory(IntChannelFactory mySubFactory): int
ChannelFactoryBase {abstract}
+ myListSubFactories: ArrayList
+ getChannelFromSubFactory(String szChannelType):IntChannel
ChannelSSLFactory
+ myInstance: IntChannelFactory
+ getChannel(String szChannelType): IntChannel
ChannelFactory
+ myInstance: IntChannelFactory
+ getChannel(String szChannelType): IntChannel

Figure 5.17: Class Diagram of the example of the Abstract Factory pattern

code. It is a description or template for how to solve a problem that can be used in many different situations [24].

Abstract Factory

Abstract Factory is a creational design pattern that lets you produce families of related objects without specifying their concrete classes. The pattern was applied to separate the logic of SSL and non-SSL connection channel like shown on Figure 5.17.

Three classes and one interface were created. The interface *IntChannelFactory* has two methods declared: *getChannel(String szChannelType)* and *addSubFactory(IntChannelFactory mySubFactory)*. Then the abstract class *ChannelFactoryBase* implements the interface, defines the method *addSubFactory(IntChannelFactory mySubFactory)* and stores the ArrayList *myListSubFactories*. Finally, two classes extend *ChannelFactoryBase*. In classes *ChannelFactory* and *ChannelSSLFactory* the method *getChannel(String szChannelType)* has the logic implemented.

Observer

The Observer pattern is a one-to-many dependency between objects so that when one object

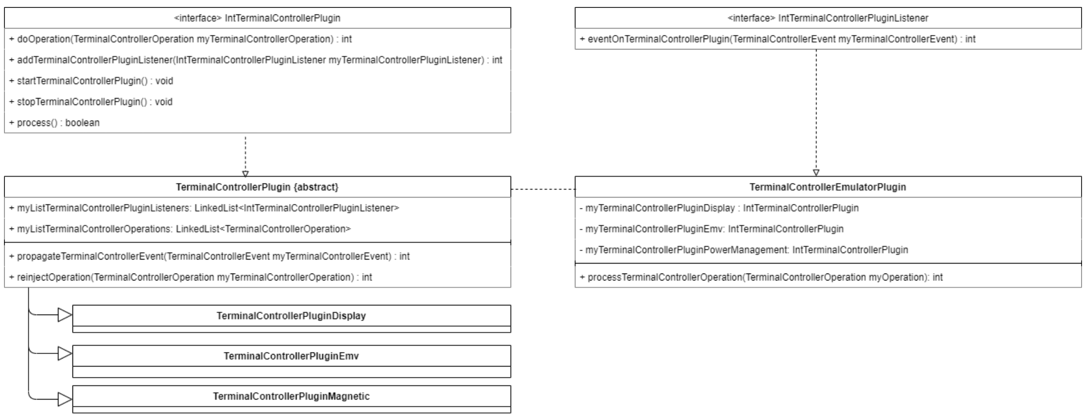

<interface> IntTerminalControllerPlugin
+ doOperation(TerminalControllerOperation myTerminalControllerOperation) : int
+ addTerminalControllerPluginListener(IntTerminalControllerPluginListener myTerminalControllerPluginListener) : int
+ startTerminalControllerPlugin() : void
+ stopTerminalControllerPlugin() : void
+ process() : boolean
<interface> IntTerminalControllerPluginListener
+ eventOnTerminalControllerPlugin(TerminalControllerEvent myTerminalControllerEvent) : int
TerminalControllerPlugin {abstract}
+ myListTerminalControllerPluginListeners: LinkedList<IntTerminalControllerPluginListener>
+ myListTerminalControllerOperations: LinkedList<TerminalControllerOperation>
+ propagateTerminalControllerEvent(TerminalControllerEvent myTerminalControllerEvent) : int
+ reinjectOperation(TerminalControllerOperation myTerminalControllerOperation) : int
TerminalControllerEmulatorPlugin
- myTerminalControllerPluginDisplay : IntTerminalControllerPlugin
- myTerminalControllerPluginEmv: IntTerminalControllerPlugin
- myTerminalControllerPluginPowerManagement: IntTerminalControllerPlugin
+ processTerminalControllerOperation(TerminalControllerOperation myOperation): int
TerminalControllerPluginDisplay
TerminalControllerPluginEmv
TerminalControllerPluginMagnetic

Figure 5.18: Class Diagram of the example of the Observer pattern

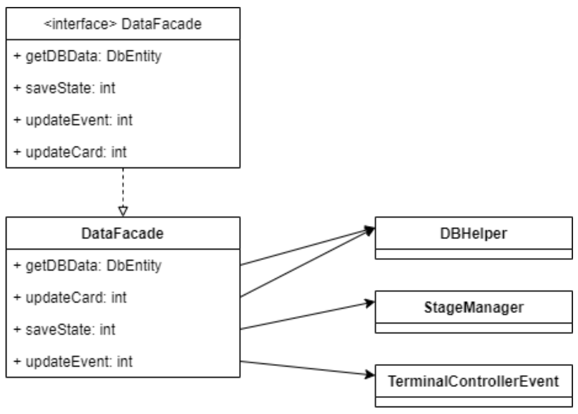


Figure 5.19: Class Diagram of the example of the Façade pattern

changes state, all its dependents are notified. This is typically done by calling one of their methods. The application supports many standalone operations and events (full list can be found in Appendix A). In order to reduce the number of direct dependencies between classes, the Observer pattern was used [35].

When one object needs to run one process step, it calls *doOperation()* to add the step into the *TerminalControllerPlugin* queue of the actions to be performed. The queue is processed and if it has at least one Operation, it calls *eventOnTerminalControllerPlugin()*. Figure 5.18 illustrates the implementation of the Observer pattern in the application.

Façade

Façade is a structural design pattern that provides a simplified interface to a library, a framework, or any other complex set of classes. When the code should work with a broad set of objects that belong to a sophisticated library or framework, all these objects can be initialized, dependencies are kept tracked, methods are executed in the correct order, *etc.* [35].

The application uses its own database. Despite the usual database operations, like select, delete, update, having their method implementation, there is no need to confuse the developer with a huge number of other possible methods. Using the Façade pattern we hide non-frequently used methods. Moreover, we added some data oriented logic into *DataFacade* class, as presented in Figure 5.19.

Builder

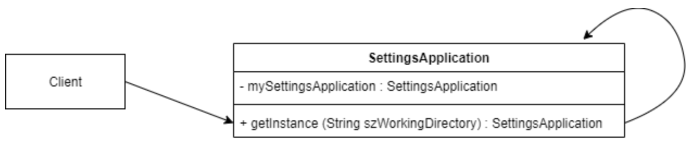
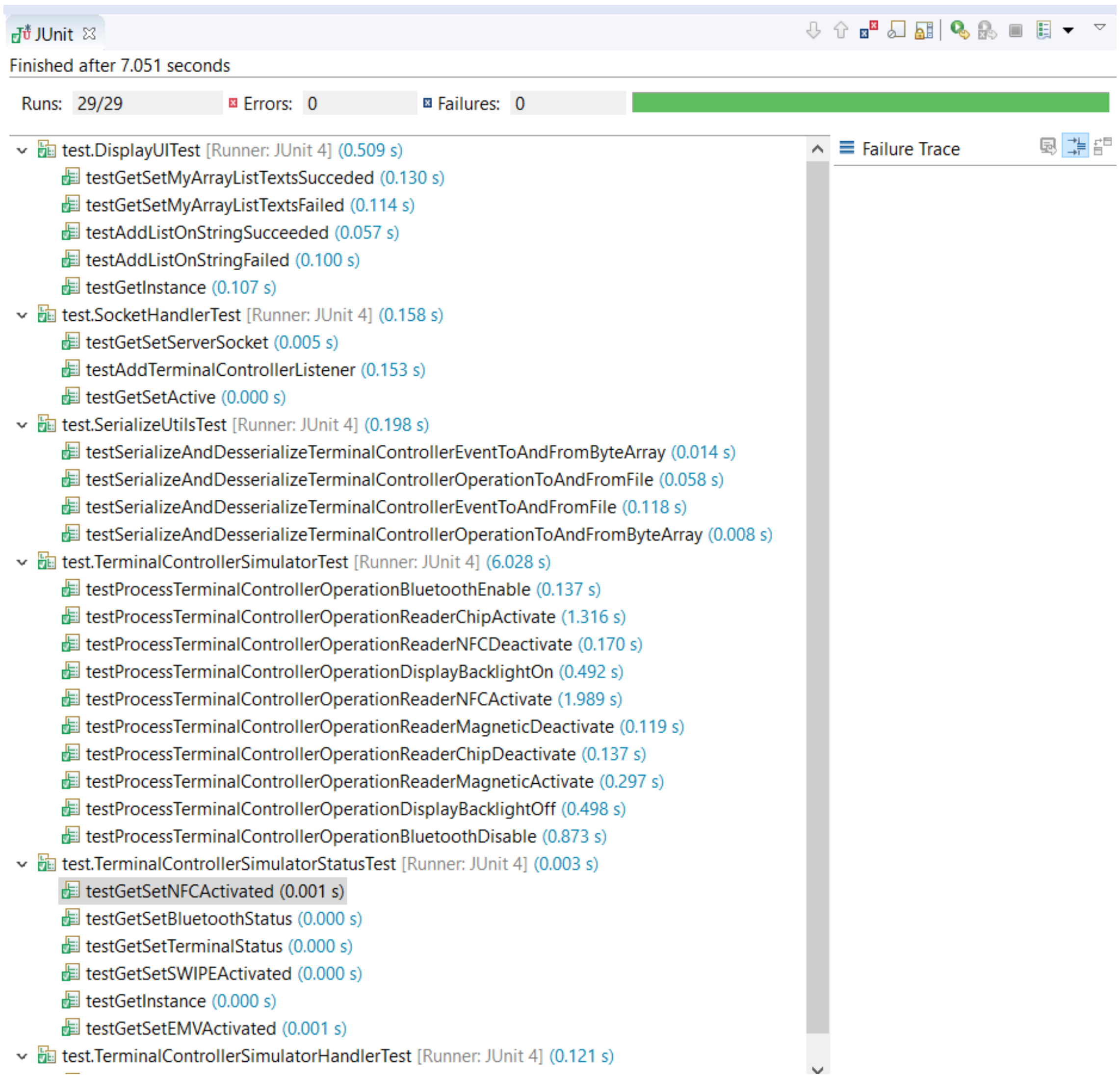

Client
SettingsApplication
- mySettingsApplication : SettingsApplication
+ getInstance (String szWorkingDirectory) : SettingsApplication

Figure 5.20: Class Diagram of the example of the Singleton pattern



```
JUnit
Finished after 7.051 seconds
Runs: 29/29    Errors: 0    Failures: 0

test.DisplayUITest [Runner: JUnit 4] (0.509 s)
    testGetSetMyArrayListTextsSucceded (0.130 s)
    testGetSetMyArrayListTextsFailed (0.114 s)
    testAddListOnStringSucceeded (0.057 s)
    testAddListOnStringFailed (0.100 s)
    testGetInstance (0.107 s)
test.SocketHandlerTest [Runner: JUnit 4] (0.158 s)
    testGetSetServerSocket (0.005 s)
    testAddTerminalControllerListener (0.153 s)
    testGetSetActive (0.000 s)
test.SerializeUtilsTest [Runner: JUnit 4] (0.198 s)
    testSerializeAndDesserializeTerminalControllerEventToAndFromByteArray (0.014 s)
    testSerializeAndDesserializeTerminalControllerOperationToAndFromFile (0.058 s)
    testSerializeAndDesserializeTerminalControllerEventToAndFromFile (0.118 s)
    testSerializeAndDesserializeTerminalControllerOperationToAndFromByteArray (0.008 s)
test.TerminalControllerSimulatorTest [Runner: JUnit 4] (6.028 s)
    testProcessTerminalControllerOperationBluetoothEnable (0.137 s)
    testProcessTerminalControllerOperationReaderChipActivate (1.316 s)
    testProcessTerminalControllerOperationReaderNFCDeactivate (0.170 s)
    testProcessTerminalControllerOperationDisplayBacklightOn (0.492 s)
    testProcessTerminalControllerOperationReaderNFCActivate (1.989 s)
    testProcessTerminalControllerOperationReaderMagneticDeactivate (0.119 s)
    testProcessTerminalControllerOperationReaderChipDeactivate (0.137 s)
    testProcessTerminalControllerOperationReaderMagneticActivate (0.297 s)
    testProcessTerminalControllerOperationDisplayBacklightOff (0.498 s)
    testProcessTerminalControllerOperationBluetoothDisable (0.873 s)
test.TerminalControllerSimulatorStatusTest [Runner: JUnit 4] (0.003 s)
    testGetSetNFCActivated (0.001 s)
    testGetSetBluetoothStatus (0.000 s)
    testGetSetTerminalStatus (0.000 s)
    testGetSetSWIPEActivated (0.000 s)
    testGetInstance (0.000 s)
    testGetSetEMVActivated (0.001 s)
test.TerminalControllerSimulatorHandlerTest [Runner: JUnit 4] (0.121 s)

Failure Trace
```

Figure 5.21: Screenshot of the Test Report after running Unit Tests

As the name implies, the builder pattern is used to build objects. Sometimes, the objects we create can be complex, made up of several sub-objects or require an elaborate construction process. The exercise of creating complex types can be simplified by using the builder pattern. A composite or an aggregate object is what a builder generally builds [35].

The Builder pattern was used in the coupling Operations and Events objects. As illustrated on Figure 5.18, in the class *TerminalControllerEmulatorPlugin* there are a set of objects that were implemented with the interface *IntTerminalControllerPlugin*. Each of the classes has the method *addTerminalControllerPluginListener*, in which, based on the input object type, the listener added to the corresponding controller list. This mechanism helps to achieve loose coupling between the classes.

Singleton

The singleton pattern is used to limit the creation of a class to only one object. This is beneficial when one (and only one) object is needed to coordinate actions across the system. There are several examples of where only a single instance of a class should exist, including caches, thread pools, and registries [35].

The pattern is used in several places in the application. The best example of Singleton usage is the configuration file instance. Many components uses the *SettingsApplication* class simultaneously, and there is no need to create an object everytime the component needs to read one specific field. To achieve that, we create a *private static SettingsApplication mySettingsApplication* field, and hide the constructor of the class, making it private. Than we create the static method *getInstance*, that returns the unique instance of the class *SettingsApplication*.

5.6 Test Report

One of the requirements was to write unit tests to check basic features of the application. The unit tests were written using the JUnit library [26]. The obtained test report is provided in Figure 5.21.

5.7 Summary

In this chapter we focused on the implementation part of the project. Based on the given requirements and agreed architecture of the application, the application logic was done and the GUI was improved. The main objective of the application, namely the application should process payment using the 3C TechLab UAT system, was achieved. Despite some components were implemented in simplified way, the EMV transaction flow was covered.

During the development, we strove to use a software project management plan. In particular, the project was started with establishing requirements, and then a detailed architecture was proposed. During the implementation, the work was divided into independent tasks. The refactoring was done by studying design pattern examples and implementing some of them. Finally, the testing phase was performed in order to ensure the product quality.

Chapter 6

Results

The main objectives of this dissertation work were studying the EMV specification for ICC terminal payments and the development of a software application, capable of emulating the behaviour of a real terminal payment process.

The studying of specification, together with its known vulnerabilities, and research on existing analogs of the developed application, were made. The knowledge gained is presented in brief form in Chapter 2.

As it is shown in Chapter 5 the final application provides the requested solution. However, in this chapter we provide the details on how each of the initial requirements from Chapter 3 is satisfied. So that one can be convinced that 3C Emulator meets all requirements that were set in the beginning of the development by the 3C TechLab company.

6.1 Results

In this section we repeat the list of the requirements from Chapter 3 and provide the application verification result. Here we have listed only the mandatory requirements ("should" requirements).

R.1.1. Provide a terminal-like software solution

The application 3C Emulator was developed. It is a standalone application and it has an appearance like a real terminal (Figure 5.1).

R.1.2. Integrate the application with existing 3C TechLab system interfaces, such as IntegraFE, NAS, DCC service and others

The application communicates with 3C TechLab servers, including IntegraFE, NAS and DCC service servers. The detailed architecture is described in Chapter 4.

R.1.3. Design good software architecture, e.g. use established design patterns to improve the code structure and application performance

This was achieved by using the list of design patterns. The code were written in a way to facilitate code reading and improve the application performance. The details can be found in section 5.5.

R.1.4. Process unit testing and provide final test report

The project includes a number of unit tests, written using the required library (JUnit [26]). The test results are presented in section 5.6.

R.2.1. The application should have an appearance similar to the terminals of Worldline YOMANI (Figure 3.1) type

As described in Chapter 5, the GUI template we used was the Worldline YOMANI terminal type. The similarity can be verified by comparing Figure 3.1 with Figure 5.1.

R.2.2. The application should have display, digits keyboard, buttons OK, Stop, Correct, and Menu

Figure 5.1 illustrates that the application has all these listed buttons.

R.2.3. The application should show what types of readers are activated

As can be seen in Figure 5.4, the application has the panel with the available readers icons. Once the card interaction is requested, the application show the icons as active.

R.2.4. The application should allow the user selection of the cards from the list

Figure 5.4 illustrates that the user can select one of the test cards from the list.

R.2.5. The application should allow the user to add, edit and delete test cards manually

As was described in section 5.2, 3C Emulator has the area where the user can configure its test cards. To open the configuration window the user should open Card Management → Open Card Management Window.

R.2.6. The application should support showing the messages from the standard EMV terminal messages list (Table 2.1) in different languages

The language bundle structure was used in order to support multi language display messages. The details can be found in section 5.3

R.2.7. The application should not use any third party's libraries, but use the libraries that were developed inside the 3C Payment company, as well as trusted libraries from Apache, Java and Google projects

The project technology stack was defined in the beginning of the project and the list of technologies can be found in section 3.6. The listed tools were enough to develop the full project, so there was no need to use third party's libraries.

R.3.1. The application should communicate with the 3C TechLab UAT payment server. The addresses of the external services should be located in the application configuration file, so that the user can change the services the application communicates with

As was described in Chapter 4, the application communicates with several 3C TechLab services. As was requested, the address of the services are stored in the configuration file of the application.

R.3.2. The application by default should be configured to process transactions in the fake acquirer network (BNKSM). However, the application should be able to be configured to reach real acquirer test networks to process a transaction

The same as for R.3.1., the network can be configured at the same configuration file of the application (Chapter 4).

R.3.3. The application should emulate the processing card data reading only by two card readers: magnetic stripe and ICC readers

R.3.4. The application should support the following card types: Visa, Mastercard, Maestro

The application by default has test cards which are Visa, Mastercard and Maestro (Figure 5.4). However, Diners, JCB and American Express cards can be added in the Card Configuration window.

R.3.5. The application should process the payment transaction following the EMV specification. The processing should include Data Authentication, Cardholder Verification, Terminal Risk Management, Terminal Action Analysis, Offline Authentication. The application can optionally support Card Action Analysis, Online Issue Authentication, Script Processing

As described in section 5.3, the application process the transaction following the EMV specification, namely process required steps – Data Authentication, Cardholder Verification, Terminal Risk Management, Terminal Action Analysis, Offline Authentication.

R.3.6. The application should build transaction messages in ISO8583 format [25] and send it to the 3C TechLab server. The payment result from the server, which is also in ISO8583 format, should be parsed and the result should be shown in the application display

The log from NAS server was provided in section 5.3, and the messages to and from the server have the ISO8583 format.

R.3.7. The application in the case of successful transaction processing, should show the authorisation code from the acquirer as well

As shown in Figure 5.6, the display of 3C Emulator shows the Authorisation code, that was sent back by the acquirer.

R.3.8. The application should connect to the Certification Authority storage and process the key signing algorithm

In section 5.3 the mechanism of processing public key signature is described.

R.3.9. The application should support "T = 1" protocol only

All 3C Emulator transactions are processed supporting "T = 1". More details can be found in section 5.3. However, the application does not support the "T = 0" protocol.

R.3.10. The application should process PIN verification offline. The PIN processing can be omitted if the transaction does not require PIN verification, e.g. small amount transactions with trusted card

Despite the payment example presented in section 5.2 does not include PIN entering, the emulator processes the PIN during transaction if the card is configured in a proper way. This means that before configuring the test card, the user should not only enter plain text PIN value, but also the data should be included in ICC parameters in the Event Manager window. If the test card with PIN verification is configured and used in the payment, the application processes offline PIN verification (section 5.3).

R.3.11. The application by default should process authentication following the CDA algorithm

All 3C Emulator transactions are processed using the CDA algorithm. More details can be found in section 5.3. The DDA and SDA are not supported.

R.3.12. The application should emulate printing a receipt, e.g. provide receipt within a text file

Since the application does not have appropriate hardware connected to print the receipt of the payment, the application saves the transaction result in a text file.

6.2 Summary

The acceptance testing phase of the project was performed using the list of the requirements, that was established in the beginning of the work. Despite some requirements not satisfied in a full form, the project is considered meeting all requirements. In addition, not only the application achieved its objectives, the executed work covered all planned activities, such as test report, improving the source code, *etc.*

Chapter 7

Conclusions and Future Work

7.1 Conclusion

The result of the dissertation work can be presented in two parts: studying of the official specification and development of the software project.

First, a comprehensive analysis of EMV specification and business process of the payment with PoS terminal was performed. We considered the different algorithms of providing a secure payment solution, and on top of that, known vulnerabilities were studied. Despite not using in the development part of all presented information in the research result, we found it important to include the analysis result in detail. Thereby, this dissertation work can be used as an EMV specification and payment process summary, as it was already used in the Portuguese FinTech company, 3C TechLab, to explain general payment mechanisms to the employees, who do not work close with the PoS terminal software.

Second part of the work took place the design and development of the software application. Before implementations, we started looking through existing solutions in the area and the most relevant ones were presented in this document. In particular, this is motivated by the current work of the author in collaboration with 3C TechLab. The company proposed the project for this dissertation work and during almost one year the development was performed in their environment. Not many features of real EMV terminal were included in the emulator application, however, the mandatory requirements of the project were met.

The developed application was named 3C Emulator, and it has an appearance similar to a real terminal. The application can process payment transactions, and can reach the payment network, as was requested. Currently, the application supports the emulation of two card readers features: ICC and magnetic swipe readers.

The 3C Emulator application was tested in 3C TechLab by developers, but, unfortunately, the application was not integrated into development and certification process yet, mainly due to delays caused by the current pandemic situation. We expect to continue improving the 3C Emulator application, so that it can facilitate the time-consuming testing process.

7.2 Further Work

In view of both EMV specification review and application development, there are many areas for improvement.

As was noticed during the time of writing this dissertation, every week new articles about EMV specifications, PoS terminal security algorithms, ICC card vulnerabilities, and other relevant topics are published, and many of the articles deserve consideration. The knowledge base of the FinTech market technology is wide and ever changing.

Despite the 3C Emulator application development met initial requirements set in the beginning of the dissertation work, the application still has much room to continue growing. The developed task is very flexible as it allows adding different features and scaling up, but it still involves work and some knowledge of its code. Extending compatibility to existing systems in the market would be the most practical route in terms of future work.

Future work also includes extension of the application features list, namely uploading ICC data using an external card reader device, covering other terminal readers' (NFC and Bluetooth) interfaces, implementation of the "T = 0" protocol. Moreover, only the CDA algorithm was implemented, but support of authentication with the DDA and SDA algorithms can be considered. The graphical user interface, in spite of having all needed elements, can be improved to be more similar to the real terminal appearance.

Finally, due to COVID-19, in 2020 3C TechLab has to adapt the usual company routine and project plans to the prevailing situation. The integration of the final solution was put on hold without a scheduled time frame. Therefore, for future work, it is planned to integrate 3C Emulator into development and certification process to improve the verification process.

References

[1] Abe-Tech. EMV L1 PCD Test Solutions. Available at http://www.abe-tech.com/En/abe/products/erminal_testing/EMV_L1_PCD_Test_Bench.html, June 2020.

[2] B. Adida, M. Bond, J. Clulow, A. Lin, S. J. Murdoch, R. Anderson, and R. Rivest. *Phish and chips: Traditional and new recipes for attacking EMV. In Security Protocols*, volume 5087 of LNCS. Springer, 2009.

[3] Java AWT. Tutorial. Available at https://www.javatpoint.com/java-awt, June 2020.

[4] A. Barisani, D. Bianco, A. Laurie, , and Z. Franken. Chip and pin is definitely broken. *Presentation at CanSecWest Applied Security Conference*, 2011.

[5] Java BlueCove. About BlueCove JSR-82 Project. Available at http://www.bluecove.org/, June 2020.

[6] J. Breekel, D. Ortiz-Yepes, E. Poll, and J. Ruiter. *EMV in a nutshell*. IBM, 2016.

[7] Z. Chen. *Java Card Technology for Smart Cards*. Addison Wesley, 2000.

[8] Google Developers. About Protocol Buffers. Available at https://developers.google.com/protocol-buffers, June 2020.

[9] Eclipse. The Eclipse Foundation. Available at https://www.eclipse.org/ide/, June 2020.

[10] EFTLab. BP-SIM. Available at https://www.eftlab.co.uk/bp-sim/, June 2020.

[11] M. Emms and A. van Moorsel. Practical attack on contactless payment cards. *Wealth and Identity Theft*, 2011.

[12] M. J. Emms. *Contactless payments: usability at the cost of security?* PhD thesis, Newcastle University, Newcastle, 2016.

[13] EMVCo. *Book 1: Application Independent ICC to Terminal Interface Requirements*. EMV® Integrated Circuit Card Specifications for Payment Systems, 4.3 edition, 2011.

[14] EMVCo. *Book 2: Security and Key Management*. EMV® Integrated Circuit Card Specifications for Payment Systems, 4.3 edition, 2011.

[15] EMVCo. *Book 3: Application Specification*. EMV® Integrated Circuit Card Specifications for Payment Systems, 4.3 edition, 2011.

[16] EMVCo. *Book 4: Cardholder, Attendant, and Acquirer Interface Requirements*. EMV® Integrated Circuit Card Specifications for Payment Systems, 4.3 edition, 2011.

[17] EMVCo. *Technical Framework*. EMV® Payment Tokenisation Specification, 2.1 edition, 2019.

[18] EMVCo. Document Search. Available at https://www.emvco.com/document-search, June 2020.

[19] EMVCo. Emvco home. Available at https://www.emvco.com/, June 2020.

[20] EMVCo. Overview of EMVCo. Available at https://www.emvco.com/about/overview/, June 2020.

[21] FIME. EMVeriPOS. Available at https://www.fime.com/products/analog-protocol/emveripos.html, June 2020.

[22] FIS. EMV Test Tools. Available at https://www.fisglobal.com/solutions/payments/-/media/fisglobal/files/brochure/emv-terminal-integration-test-solutions.pdf, January 2020.

[23] OOREKA Agent France. Ticket de carte bancaire. Available at https://carte-bancaire.ooreka.fr/astuce/voir/515831/ticket-de-carte-bancaire, June 2020.

[24] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., USA, 1995.

[25] ISO. *ISO 8583-1:2003 Financial transaction card originated messages — Interchange message specifications*. technical documentation, 2003.

[26] JUnit. About JUnit4. Available at https://junit.org/junit4/, June 2020.

[27] KaSYS. KaNest®-ICC Card Simulator. Available at https://www.kasys.ca/en-index.html, June 2020.

[28] R. Kumar and S. O'Brien. *2019 Findings from the Diary of Consumer Payment Choice*. CPO Federal Reserve System, 2019.

[29] The Apache Log4j. About Log4j v.2. Available at https://logging.apache.org/log4j/2.x/, June 2020.

[30] N. Madhoun, E. Bertin, and G. Pujolle. *An Overview of the EMV Protocol and Its Security Vulnerabilities*. The 4th IEEE International Conference MobiSecServ, 2018.

[31] N. El Madhoun and E. Bertin. Magic always comes with a price: Utility versus security for bank cards. *The 1st IEEE Cyber Security in Networking Conference*, 2017.

[32] N. El Madhoun and G. Pujolle. Security enhancements in EMV protocol for NFC mobile payment. *The 15th IEEE International Conference on Trust*, 2016.

[33] R. Mann. Credit cards and debit cards in the united states and japan. *SSRN Electronic Journal*, 2002.

[34] K. Mayes and K. Markantonakis. *Smart Cards, Tokens, Security and Applications*. Springer, Second edition, 2017.

[35] Medium.com. The 7 Most Important Software Design Patterns. Available at https://medium.com/educative/the-7-most-important-software-design-patterns-d60e546afb0e, June 2020.

[36] S. J. Murdoch, S. Drimer, R. Anderson, and M. Bond. *Chip and PIN is Broken. In Symposium on Security and Privacy*. IEEE, 2010.

[37] O. Ogundele, P. Zavarsky, D. Lindskog, and R Ruhl. The implementation of a full emv smartcard for a point-of-sale transaction. *World Congress on Internet Security*, 2012.

[38] Oracle. Java 8 Central. Available at https://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html, June 2020.

[39] 3C Payment. Home page. Available at https://www.3cpayment.com/, June 2020.

[40] PCI. *PCI HSM Security Requirements v1.0*. PCI® Security Standards Council LLC, 1.0 edition, 2009.

[41] GitHub repository. Git Extensions. Available at http://gitextensions.github.io/, June 2020.

[42] Research and Markets. Global EMV POS Terminals Market 2017-2021. Available at https://www.researchandmarkets.com/research/dc4513/global_emv_pos, June 2020.

[43] Gerrit Code Review. Home page of Gerrit Code Review. Available at https://www.gerritcodereview.com/index.html, June 2020.

[44] J. Ruiter and Poll E. *Formal analysis of the EMV protocol suite*, volume 6993 of LNCS. Springer, 2012.

[45] SecureList. The Great Bank Robbery: the Carbanak APT. Available at https://securelist.com/the-great-bank-robbery-the-carbanak-apt/68732/, June 2020.

[46] SmartOI. SmartOI Terminal Simulator. Available at https://sites.google.com/a/smartoi.com/smartoi/s-w/smartoi-terminal-simulator, June 2020.

[47] SQLite. About SQLite. Available at https://www.sqlite.org/about.html, June 2020.

[48] R. Srinivasan. Integrated debit and credit card with dynamic card selection and card locking. *Journal of Theoretical and Applied Information Technology*, page 31, 2013.

[49] 3C TechLab. Home page. Available at http://3ctechlab.com/, June 2020.

[50] Trello. Home Page. Available at https://trello.com, June 2020.

[51] IMS UL. UL Brand Test Tool. Available at https://ims.ul.com/ul-brand-test-tool, June 2020.

[52] WeChip. WeChip Simulator. Available at https://www.wechiptech.com/product-brochures/EMV-Simulator.pdf, June 2020.

[53] Worldline. Terminal YOMANI. Available at https://terminals.worldline.com/en/home/products/payment-terminals/countertop-terminals/yomani.html, June 2020.

[54] The Apache Xerces. About Xerces2 Project. Available at https://xerces.apache.org/, June 2020.

Appendix A

Static values of the 3C Emulator application

A.1 3C Emulator Currency codes

Table A.1: Currency codes used in 3C Emulator

ID	ISO Code	ISO Text Code	Description	Minors
13	012	DZD	Algerian Dinar	2
14	032	ARS	Argentine Peso	2
15	036	AUD	Australian Dollar	2
16	044	BSD	Bahamian Dollar	2
17	048	BHD	Bahraini Dinar	3
18	050	BDT	Bangladeshi Taka	2
19	051	AMD	Armenia Dram	3
20	052	BBD	Barbados Dollar	2
21	060	BMD	Bermudan Dollar	2
22	072	BWP	Botswana Pula	2
23	084	BZD	Belize Dollar	2
24	096	BND	Brunei Dollar	2
25	112	BYB	Belarus Ruble	2
26	124	CAD	Canadian Dollar	2
27	136	KYD	Cayman Islands Dollar	2
28	144	LKR	Sri Lankan Rupee	2
29	152	CLP	Chile Peso	2
30	156	CNY	Yuan China	2

31	170	COP	Columbian Peso	2
32	188	CRC	Costa Rica Colon	2
33	191	HRK	Croatian Kuna	2
34	196	CYP	Cyprus Pound	2
35	203	CZK	Czech Republic Korun	2
36	208	DKK	Danish Krone	2
37	233	EEK	Estonia Kroon	2
38	328	GYD	Guyana Dollar	2
39	332	HTG	Haiti Gourde	2
40	344	HKD	Hong Kong Dollar	2
41	348	HUF	Hungarian Forint	0
42	352	ISK	Iceland Krone	2
43	356	INR	India Roupie	2
44	360	IDR	Indonesia Rupiah	0
45	368	IQD	Iraqi Dinar	3
46	376	ILS	Israel New Shekels	2
47	388	JMD	Jamaican Dollar	2
48	392	JPY	Japanese Yen	0
49	398	KZT	Kazakhstani Tenge	2
50	400	JOD	Jordanian Dinar	3
51	404	KES	Kenyan Shilling	2
52	410	KRW	South Korean Won	0
53	414	KWD	Kuwaiti Dinar	3
54	422	LBP	Lebanese Pound	2
55	428	LVL	Latvian Lats	2
56	434	LYD	Libya Dinar	3
57	440	LTL	Lithuanian Litas	2
58	458	MYR	Malaysian Ringgit	2
59	470	MTL	Malta Livre	2
60	480	MUR	Mauritian Rupee	2
61	484	MXN	Mexican Peso	2
62	504	MAD	Morocco Dirhams	2
63	512	OMR	Omani Rial	3
64	532	ANG	Netherlands Antillian Guilder	2
65	533	AWG	Aruba Florin	2
66	554	NZD	New Zealand Dollar	2
67	566	NGN	Nigerian Naira	2
68	578	NOK	Norwegian Krone	2

69	586	PKR	Pakistan Rupee	2
70	590	PAB	Panamanian Balboa	2
71	608	PHP	Philippine Peso	2
72	634	QAR	Qatari Rial	2
73	643	RUB	Russian Ruble	2
74	682	SAR	Saudi Riyal	2
75	690	SCR	Seychelles Rupee	2
76	702	SGD	Singapore Dollar	2
77	704	VND	Vietnamese dong	0
78	705	SIT	Slovenian Tolar	2
79	710	ZAR	South African Rand	2
80	752	SEK	Swedish Krona	2
81	756	CHF	Swiss Franc	2
82	760	SYP	Syrian Pound	2
83	764	THB	Thailand Baht	2
84	780	TTD	Trinidad and Tobago Dollar	2
85	784	AED	United Arab Dirham	2
86	788	TND	Tunisian Dinar	3
87	818	EGP	Egyptian Pound	2
88	826	GBP	Pounds Sterling	2
89	834	TZS	Tanzanian Shilling	2
90	840	USD	US Dollar	2
91	886	YER	Yemen Rial	2
92	894	ZMK	Zambian Kwacha	2
93	901	TWD	Taiwanese Dollar	2
94	933	BYN	Belarusian Ruble	2
95	936	GHS	Ghana Cedi	2
96	937	VEF	Venezuela Bolivar Fuerte	2
97	941	RSD	Serbian Dinar	2
98	944	AZN	Azerbaijani Manat	2
99	946	RON	Romanian New Leu	2
100	949	TRY	Turkish Lira	2
101	950	XAF	CFA Franc BEAC	0
102	951	XCD	East Caribbean Dollar	2
103	952	XOF	CFA Franc BEACO	0
104	967	ZMW	Zambia Kwacha	2
105	968	SRD	Surinam Dollar	2
106	971	AFN	Afghanistan Afghani	2

107	973	AOA	Angolan kwanza	2
108	975	BGN	Bulgarian Lev	2
109	978	EUR	Euro	2
110	980	UAH	Ukranian Hryunia	2
111	981	GEL	Georgia Lari	2
112	985	PLN	Polish Zloty	2
113	986	BRL	Brazilian Real	2

A.2 3C Emulator Operations

Table A.2: Operations supported by 3C Emulator

OPERATION_UNKNOWN
OPERATION_BEEP
OPERATION_READER_MAGNETIC_ACTIVATE
OPERATION_READER_MAGNETIC_DEACTIVATE
OPERATION_READER_CHIP_ACTIVATE
OPERATION_READER_CHIP_DEACTIVATE
OPERATION_READER_NFC_ACTIVATE
OPERATION_READER_NFC_DEACTIVATE
OPERATION_BACKLIGHT_ON
OPERATION_BACKLIGHT_OFF
OPERATION_DISPLAY_TEXT
OPERATION_DISPLAY_LIST_ITEMS
OPERATION_DISPLAY_TEXT_INPUT
OPERATION_DISPLAY_PIN_ENTRY
OPERATION_PRINT_TEXT
OPERATION_EMV_RECOGNIZE
OPERATION_EMV_INITIALIZE
OPERATION_EMV_PROCESS
OPERATION_EMV_COMPLETE
OPERATION_EMV_TERMINATE
OPERATION_BEEP_ALARM
OPERATION_EMV_PIN_BYPASS
OPERATION_EMV_PIN_CONFIRM
OPERATION_EMV_PIN_CANCEL
OPERATION_EMV_DATA_PROVIDE_0
OPERATION_EMV_DATA_PROVIDE_1
OPERATION_EMV_DATA_PROVIDE_2

OPERATION_HIBERNATE
OPERATION_REACTIVATION_GET_REACTIVATION_DATA
OPERATION_REACTIVATION_SET_REACTIVATION_CODE
OPERATION_EMV_PRECONFIGURE
OPERATION_CONTROLLER_INITIALIZE
OPERATION_SRED_STATUS_GET
OPERATION_RKI_KEYS_INIT
OPERATION_RKI_KEYS_INJECT
OPERATION_SYSTEM_TIME_SET
OPERATION_SYSTEM_RESET
OPERATION_FILES_WRITE
OPERATION_FILES_READ
OPERATION_POWER_STATUS_GET
OPERATION_INFO_GET
OPERATION_CONFIG_GET
OPERATION_FILES_STREAM_WRITE
OPERATION_SECURITY_SESSION_OPEN
OPERATION_SECURITY_SESSION_CLOSE
OPERATION_SECURITY_PIN_ENCRYPT
OPERATION_SECURITY_MAC_GENERATE
OPERATION_SECURITY_MAC_VERIFY
OPERATION_SECURITY_CARD_DATA_ENCRYPT
OPERATION_SECURITY_RESET_PCI_CONTEXT
OPERATION_SECURITY_LOAD_MP1_CARD_TABLE
OPERATION_SECURITY_LOAD_MP1_FORM_PACKAGE
OPERATION_SECURITY_LUHN_DIGIT_CHECK
OPERATION_BLUETOOTH_STATUS_LOAD
OPERATION_BLUETOOTH_ENABLE
OPERATION_BLUETOOTH_DISABLE
OPERATION_BLUETOOTH_GET_PAIRED_DEVICES
OPERATION_BLUETOOTH_SCAN_DEVICES
OPERATION_BLUETOOTH_PAIR
OPERATION_BLUETOOTH_UNPAIR
OPERATION_BLUETOOTH_DEVICE_VISIBLE_ENABLE
OPERATION_BLUETOOTH_DEVICE_VISIBLE_DISABLE
OPERATION_BLUETOOTH_TETHERING_ENABLE
OPERATION_BLUETOOTH_TETHERING_DISABLE
OPERATION_TERMINAL_DUMP_LOG

A.3 3C Emulator Events

The list of all events supported in the 3C Emulator application is presented in Table A.3. The events marked as Presents in GUI can be used in the Card Management windows of the application.

Table A.3: Events supported by 3C Emulator

NAME	Presents in GUI
EVENT_UNKNOWN	
EVENT_CONTROLLER_READY	
EVENT_ERROR_API	
EVENT_ERROR_DISPLAY	
EVENT_ERROR_BACKLIGHT	
EVENT_ERROR_PRINTER	
EVENT_ERROR_READER_MAGNETIC	
EVENT_ERROR_READER_CHIP	
EVENT_ERROR_READER_CONTACTLESS	
EVENT_ERROR_EMV_ENGINE	
EVENT_ERROR_STATE_ENGINE	
EVENT_BEEPED	
EVENT_READER_MAGNETIC_ACTIVATED	
EVENT_READER_MAGNETIC_DEACTIVATED	
EVENT_READER_CHIP_ACTIVATED	
EVENT_READER_CHIP_DEACTIVATED	
EVENT_READER_NFC_ACTIVATED	
EVENT_READER_NFC_DEACTIVATED	
EVENT_BACKLIGHT_ON	
EVENT_BACKLIGHT_OFF	
EVENT_TEXT_DISPLAYED	
EVENT_EMV_RECOGNIZE_STARTED	
EVENT_EMV_INITIALIZE_STARTED	
EVENT_EMV_PROCESS_STARTED	
EVENT_EMV_COMPLETE_STARTED	
EVENT_EMV_TERMINATE_STARTED	
EVENT_KEY_PRESSED	
EVENT_EMV_POWERED_UP	$\star$
EVENT_CARD_REMOVED	$\star$
EVENT_CARD_CONTACTLESS_SHOWNUP	
EVENT_CARD_CONTACTLESS_DISAPPEARED	
EVENT_CARD_SWIPED	$\star$

EVENT_EMV_RECOGNIZED	⋆
EVENT_EMV_PIN_ENTRY_REQUIRED	
EVENT_EMV_INITIALIZED	⋆
EVENT_EMV_PROCESSED	⋆
EVENT_EMV_COMPLETED	⋆
EVENT_EMV_TERMINATED	⋆
EVENT_PIN_ENTERED	
EVENT_EMV_PIN_BYPASSED	
EVENT_EMV_PIN_CONFIRMED	
EVENT_EMV_PIN_CANCELED	
EVENT_PIN_NOT_ENTERED	
EVENT_ERROR_BEEP	
EVENT_LIST_DISPLAYED	
EVENT_TEXT_INPUT_DISPLAYED	
EVENT_PIN_ENTRY_DISPLAYED	
EVENT_SECURITY_SESSION_OPENED	
EVENT_SECURITY_SESSION_CLOSED	
EVENT_SECURITY_PIN_ENCRYPTED	
EVENT_SECURITY_MAC_GENERATED	
EVENT_SECURITY_MAC_VERIFIED	
EVENT_SECURITY_CARD_DATA_ENCRYPTED	⋆
EVENT_SECURITY_PCI_CONTEXT_RESETTED	
EVENT_SECURITY_MP1_CARD_TABLE_LOADED	
EVENT_SECURITY_MP1_FORM_PACKAGE_LOADED	
EVENT_ERROR_SECURE_SERVICE	
EVENT_ERROR_POWER_MANAGEMENT	
EVENT_NFC_STATUS_CHANGED	⋆
EVENT_NFC_PROCESSED	⋆
EVENT_EMV_DATA_MISSING_0	
EVENT_EMV_DATA_MISSING_1	
EVENT_EMV_DATA_MISSING_2	
EVENT_EMV_DATA_PROVIDED_0	
EVENT_EMV_DATA_PROVIDED_1	
EVENT_EMV_DATA_PROVIDED_2	
EVENT_TERMINAL_WOKEN_UP	
EVENT_ERROR_REACTIVATION_SERVICE	
EVENT_REACTIVATION_CHALLENGE_GENERATED	
EVENT_REACTIVATION_PROCESSED	

EVENT_NFC_OUTCOME_CHANGED	
EVENT_EMV_PRECONFIGURED	⋆
EVENT_PINPAD_CONNECTED	
EVENT_PINPAD_DISCONNECTED	
EVENT_SRED_STATUS	
EVENT_RKI_FILES_CREATED	
EVENT_RKI_FILES_INJECTED	
EVENT_SYSTEM_TIME_SET	
EVENT_PINPAD_POWER_STATUS	
EVENT_PINPAD_INFO	
EVENT_FILES_WRITTEN	
EVENT_FILES_READ	
EVENT_SECURITY_LUHN_DIGIT_CHECKED	
EVENT_SECURITY_KEY_VERSION_LOADED	
EVENT_SECURITY_KEY_CHECK_VALUE_LOADED	
EVENT_SECURITY_KEY_INJECTED	
EVENT_BLUETOOTH_STATUS_LOADED	
EVENT_BLUETOOTH_DISABLED	
EVENT_BLUETOOTH_ENABLED	
EVENT_BLUETOOTH_SCAN_FINISHED	
EVENT_BLUETOOTH_DEVICE_PAIRED	
EVENT_BLUETOOTH_DEVICE_UNPAIRED	
EVENT_BLUETOOTH_PAIRED_DEVICES	
EVENT_BLUETOOTH_DEVICE_VISIBLE_ENABLED	
EVENT_BLUETOOTH_DEVICE_VISIBLE_DISABLED	
EVENT_BLUETOOTH_PIN_DEFINITION	
EVENT_BLUETOOTH_TETHERING_ENABLED	
EVENT_BLUETOOTH_TETHERING_DISABLED	
EVENT_ERROR_BLUETOOTH	
EVENT_CARD_PRESENT	
EVENT_CARD_NOT_PRESENT	