VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2017 Bc. Roman Mravec

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

ELEKTRONICKÉ DOKLADY
ELECTRONIC ID CARDS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Roman Mravec

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Jan Hajný, Ph.D.

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Roman Mravec ***ID:*** 146064
Ročník: 2 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Elektronické doklady

POKYNY PRO VYPRACOVÁNÍ:

Cílem projektu je implementace vybraných kryptografických protokolů pro elektronické doklady na platformě čipových karet. Výstupem práce bude měření rychlosti kryptografických primitiv na kartách a implementace jednoduchého protokolu pro autentizaci založeného na eliptických křivkách na platformě MultOS. Výstupem bude software pro kartu MultOS a aplikace ověřovacího terminálu na libovolné platformě (OS Windows, Linux).

DOPORUČENÁ LITERATURA:

[1] MENEZES, Alfred, Paul C. VAN OORSCHOT a Scott A. VANSTONE. Handbook of applied cryptography. Boca Raton: CRC Press, c1997. Discrete mathematics and its applications. ISBN 0-8493-8523-7.

[2] Identity Mixer [online]. Zurich, 2016 [cit. 2016-09-12]. Dostupné z: http://www.research.ibm.com/labs/zurich/idemix/

[3] MULTOS Developer's Reference Manual [online]. , 1 - 334 [cit. 2017-02-08]. Dostupné z: https://www.multos.com/uploads/MDRM.pdf

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 24.5.2017

Vedoucí práce: doc. Ing. Jan Hajný, Ph.D.
Konzultant:

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRACT

This master thesis deals with an implementation of Diffie-Hellman protocol on smart card which is based on MULTOS OS. Defines the smart cards based on MULTOS OS and their usage. Output of this thesis are applications for a smart card and for a client using Diffie-Hellman protocol for establishing of a secret key between two communication sides through unsecured communication channel.

KEYWORDS

Elecronic ID cards, smart cards, Asymmetric cryptography, Elliptic curves, Diffie-Hellman, MULTOS, SmartDeck

ABSTRAKT

Diplomová práca sa venuje implementácii protokolu Diffie-Hellman na smart karte beziacej na operačnom systéme MULTOS. Definuje smart karty a ich použitie. Výstupom práce sú dve aplikácie, jedna pre kartu a druhú pre klienta na obsluhu karty. Tieto aplikácie sú založené na protokole Diffie-Hellman, ktorý slúži k ustanoveniu tajného spoločného kľúča pri komunikácii medzi dvomi stranami cez nezabezpečený komunikačný kanál..

KLÍČOVÁ SLOVA

Elektronické doklady, Smart karty, Asymetrická kryptografie, Eliptické křivky, Diffie-Hellman, MULTOS, SmartDeck

MRAVEC, Roman *Electronic ID cards*: master's thesis. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications, 2017. 47 p. Supervised by doc. Jan Hajný, Ph.D.

DECLARATION

I declare that I have written my master's thesis on the theme of "Electronic ID cards" independently, under the guidance of the master's thesis supervisor and using the technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis.

As the author of the master's thesis I furthermore declare that, as regards the creation of this master's thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone's personal and/or ownership rights and I am fully aware of the consequences in the case of breaking Regulation § 11 and the following of the Copyright Act No 121/2000 Sb., and of the rights related to intellectual property right and changes in some Acts (Intellectual Property Act) and formulated in later regulations, inclusive of the possible consequences resulting from the provisions of Criminal Act No 40/2009 Sb., Section 2, Head VI, Part 4.

Brno ....

author's signature

ACKNOWLEDGEMENT

I would like to thank for professional guidance, consultations, dedication and suggestive proposals to doc. Ing. Jan Hajný, Ph.D. and Ing. Petr Dzurenda. Special thanks belongs to my friends BEng. Andrej Maris and Ing. Peter Mendel for consulting the problematic of smart cards and my parents for a support.

Brno . . . . . .

. . . . . .

author's signature

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

ACKNOWLEDGEMENT

Research described in this master's thesis has been implemented in the laboratories supported byt the SIX project; reg. no. CZ.1.05/2.1.00/03.0072, operational program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .

author's signature

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

From the beginning of the century, people are using cards every day – for paying, withdrawing money, proving an identity and eligibility, compensating a key and so on. Naturally, since the cards are carrying more and more significant data, the priority is imposed for security. The main aim of manufacturers of the cards is clear – to keep the contain of the cards genuine and trusted.

Nowadays, when the trend of digitalization affects a wide spectrum of a life, it is absolutely a standard to pay with credit cards, to use keyless cards and tags to enter into a room or to record arrivals at work. Since the electronic signature has been established, a new doors for digital revolution are open. Typically, to sign the documents with electronic signature and prove an identity, the vision of usage of electronic personal identification documents such as ID cards is aroused. There are many other ideas how to use authentication of a person during using an electronic documents such as keyless opening of a car with NFC[1] (Near Field Communication) or BLE[2] (Bluetooth Low Energy)technology, to use a health card with medical records data. Generally, to use electronic documents it means to load a specific information onto a card (or similar, portable and energetically efficient wireless device). To keep those information in safe and trusted, the electronic ID cards must be effectively secured. This master thesis solves this problematic of electronic ID cards ID card (Identification Card) and brings the solution how to implement cryptographic protocol while using such a card.

[1]NFC is a technology and a set of communication protocols that enable two electronic devices, one of which is usually a portable device such as a smartphone, to establish a wireless communication by bringing them within 4 cm of each other.

[2]BLE is a wireless personal area network technology designed and marketed by the Bluetooth Special Interest Group. Compared to Classic Bluetooth, Bluetooth Smart is intended to provide considerably reduced power consumption and cost while maintaining a similar communication range.

1 CRYPTOGRAPHY

Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication. Cryptography is not the only means of providing information security, but rather one set of techniques. Cryptography as a discipline has very obvious targets:

1. *Confidentiality* is a service used to keep the content of information from all but those authorized to have it. There are numerous approaches to providing confidentiality, ranging from physical protection to mathematical algorithms which render data unintelligible.
2. *Data integrity* is a service which addresses the unauthorized alteration of data. To assure data integrity, there must be the ability to detect data manipulation by unauthorized parties. Data manipulation includes such things as insertion, deletion, and substitution.
3. *Authentication* is a service related to identification. This function applies to both entities and information itself. Two parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons this aspect of cryptography is usually subdivided into two major classes: entity authentication and data origin authentication. Data origin authentication implicitly provides data integrity (for if a message is modified, the source has changed).
4. *Non-repudiation* is a service which prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary. For example, one entity may authorize the purchase of property by another entity and later deny such authorization was granted. A procedure involving a trusted third party is needed to resolve the dispute.

Following the mentioned goals above, a major role of cryptography is to appropriately address these four areas in both theory and practice. Cryptography is focused on the prevention and detection of cheating and other malicious activities. To provide such services, it uses the cryptographic tools – **primitives**. Examples of such primitives are: encryption schemes, hash functions, digital signature schemes, etc. Parameters for consideration of a strength of cryptographic parameters and a relevance of its usage in specific cases are:

- *Level of security* This is usually difficult to quantify. Often it is given in terms of the number of operations required (using the best methods currently known) to defeat the intended objective. Typically the level of security is defined by

an upper bound on the amount of work necessary to defeat the objective. This is sometimes called the work factor.

- *Functionality* Primitives will need to be combined to meet various information security objectives. Which primitives are most effective for a given objective will be determined by the basic properties of the primitives.
- *Methods of operation* Primitives, when applied in various ways and with various inputs, will typically exhibit different characteristics; thus, one primitive could provide very different functionality depending on its mode of operation or usage.
- *Performance* This refers to the efficiency of a primitive in a particular mode of operation (For example, an encryption algorithm may be rated by the number of bits). per second which it can encrypt.
- *Ease of implementation* This refers to the difficulty of realizing the primitive in a practical instantiation. This might include the complexity of implementing the primitive in either a software or hardware environment.

An importance of these criteria depend on the application and mostly on available resources, e.g. computing power. Cryptography is closely related to the disciplines of cryptology and cryptanalysis. Cryptology is the discipline, such as number theory, and the application of formulas and algorithms. Cryptanalysis refers to the study of ciphers, ciphertext, or cryptosystems (that is, to secret code systems) with a view to finding weaknesses in them that will permit retrieval of the plaintext from the ciphertext, without necessarily knowing the key or the algorithm. This is known as breaking the cipher, ciphertext, or cryptosystem. [1] [8]

1.1 Asymmetric cryptography

Asymmetric cryptography is based on asymmetric ciphers and uses one pair of keys – **public key** of a receiver of a message for encryption and **private key** of a receiver of a message for decryption, unlike symmetric cryptography, which is purely based on one private key (for both encryption and decryption). The public key in asymmetric cryptography can be given to anyone, trusted or not, while the private key must be kept secret – just like the key in symmetric cryptography. The keys are simply large numbers which that have been paired together, but are not identical (they are asymmetric). Considering a complexity of these two different approaches of cryptography, it is obvious symmetric cryptography is more simple and therefore faster, but the risk of disclosure is much higher in comparison to two keys usage.

Asymmetric cryptography has two primary use cases: authentication and confidentiality. Using asymmetric cryptography, messages can be signed with a private

key, and then anyone with the public key is able to verify that the message was created by someone possessing the corresponding private key. This can be combined with a proof of identity system to know what entity (person or group) actually owns that private key, providing authentication. The most common asymmetric encryption algorithm is RSA. Asymmetric keys have typically length 1024 or 2048 bits. However, keys smaller than 2048 bits are no longer considered safe to use. 2048-bit keys have enough unique encryption codes that we won't write out the number here (it's 617 digits). Though larger keys can be created, the increased computational burden is so significant that keys larger than 2048 bits are rarely used. To put it into perspective, it would take an average computer more than 14 billion years to crack a 2048-bit certificate.

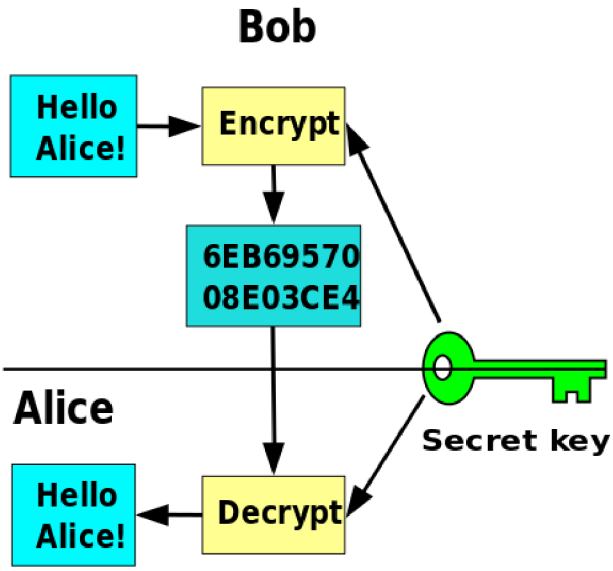

Fig. 1.1: Asymmetric cryptography principle

In asymmetric key encryption scheme (figure 1.1), anyone can encrypt messages using the public key, but only the holder of the paired private key can decrypt. Security depends on the secrecy (length) of the private key.
[2] [11] [12]

1.2 Elliptic curve cryptography

Elliptical curve cryptography is a public key encryption technique based on elliptic curve theory that can be used to create faster, smaller, and more efficient cryptographic keys. ECC generates keys through the properties of the elliptic curve

equation instead of the traditional method of generation as the product of very large prime numbers. The technology can be used in conjunction with most public key encryption methods, such as RSA and Diffie-Hellman. According to some researchers, ECC can yield a level of security with a 164-bit key that other systems require a 1024-bit key to achieve. Because ECC helps to establish equivalent security with lower computing power and battery resource usage, it is becoming widely used for mobile applications.

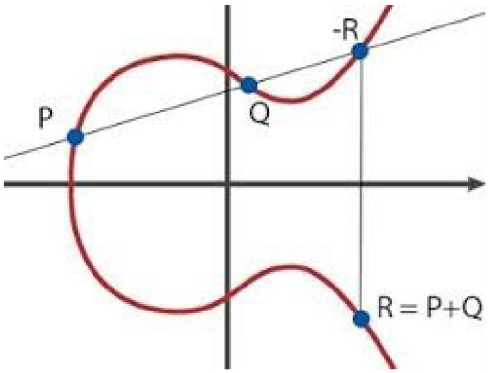

Fig. 1.2: Points on eliptic curve

The properties and functions of elliptic curves have been studied in mathematics for 150 years. Their use within cryptography was first proposed in 1985, (separately) by Neal Koblitz from the University of Washington, and Victor Miller at IBM. An elliptic curve is not an ellipse (oval shape), but is represented as a looping line intersecting two axes (lines on a graph used to indicate the position of a point as dispalyed in the figure 1.2). ECC is based on properties of a particular type of equation created from the mathematical group (a set of values for which operations can be performed on any two members of the group to produce a third member) derived from points where the line intersects the axes. Multiplying a point on the curve by a number will produce another point on the curve, but it is very difficult to find what number was used, even if you know the original point and the result. Equations based on elliptic curves have a characteristic that is very valuable for cryptography purposes: they are relatively easy to perform, and extremely difficult to reverse. This characteristics is therefore perfectly suitable for cryptographic algorithm. The following section 1.3 introduces such an algorithm based on elliptic curve cryptography.
[13]

1.3 Diffie-Hellman

Diffie-Hellman is the first asymmetric encryption algorithm (or protocol[1]), invented in 1976, using discrete logarithms in a finite field. Allows two users to exchange a secret key over an insecure medium without any prior secrets. This process is in cryptography defined as **conference keying**. A conference keying is a generalization of two-party key establishment[2] to provide three or more parties with a shared secret key. General requirements for conference keying include that distinct groups recover distinct keys (session keys); that session keys are dynamic (excepting key pre-distribution schemes); that the information exchanged between parties is non-secret and transferred over open channels; and that each party individually computes the session key. An obvious method to establish a conference key K is to arrange that each party share a unique symmetric key with a common trusted party. Consequently the trusted party may choose a new random key and distribute it by symmetric key transport individually to each member of the conference group.

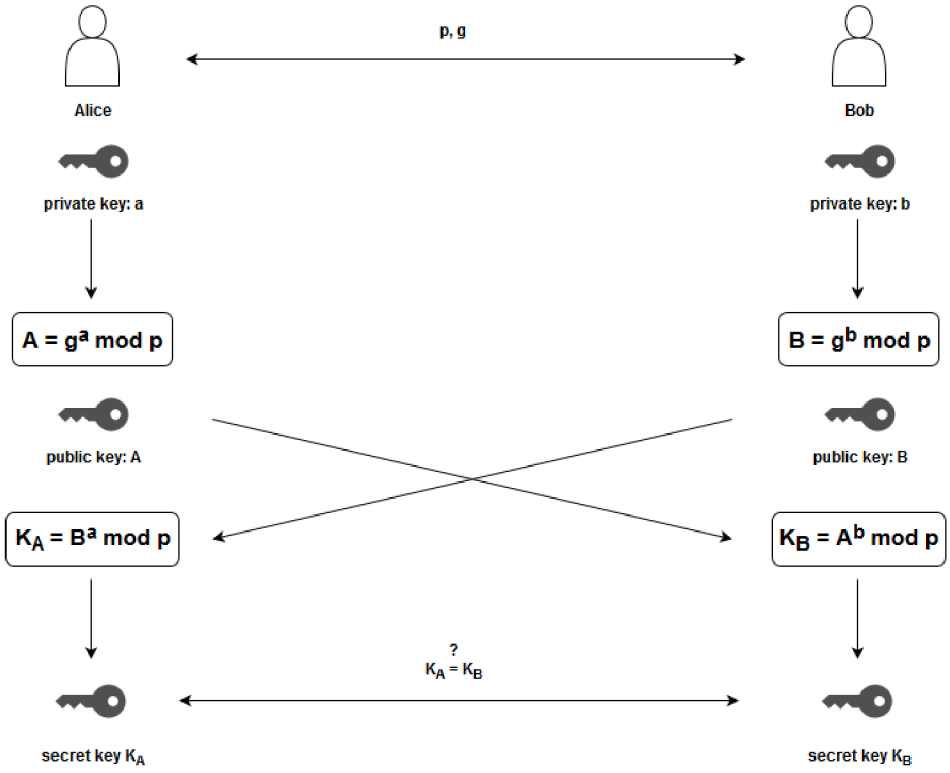

Fig. 1.3: Diffie-Hellman algorithm principle

[1]A protocol is a multi-party algorithm, defined by a sequence of steps precisely specifying the actions required of two or more parties in order to achieve a specified objective.

[2]Key establishment is a process or protocol whereby a shared secret becomes available to two or more parties, for subsequent cryptographic use.

Diffie-Hellman (DH) is a widely used key exchange algorithm. In many cryptographic protocols, two parties wish to begin communicating. However, let's assume they do not initially possess any common secret and thus cannot use secret key cryptosystems. The key exchange by Diffie-Hellman protocol remedies this situation by allowing the construction of a common secret key over an insecure communication channel. It is based on a problem related to discrete logarithms, namely the Diffie-Hellman problem. This problem is considered hard, and it is in some instances as hard as the discrete logarithm problem. The Diffie-Hellman protocol is generally considered to be secure when an appropriate mathematical group is used. In particular, the generator element used in the exponentiations should have a large period (i.e. order). Usually, Diffie-Hellman is not implemented on hardware.

1.3.1 Establishment of a key using Diffie-Hellman protocol

The implementation of the protocol uses the multiplicative group of integers modulo p, where p is prime, and g is a base. Both of these parameters are non-secret. Let's have two communication parties, Alice and Bob:

1. At first, Alice and Bob establish a modulus p = 23 and base g = 5 (which is a primitive root modulo 23).
2. Alice chooses a secret integer a = 6, then sends Bob A = g^a mod p
   - A = 5^6 mod 23 = 8
3. Bob chooses a secret integer b = 15, then sends Alice B = g^b mod p
   - B = 5^{15} mod 23 = 19
4. Alice computes $K_A = B^a$ mod p
   - $K_A = 19^6$ mod 23 = 2
5. Bob computes $K_B = A^b$ mod p
   - $K_B = 8^{15}$ mod 23 = 2
6. If $K_A = K_B$, then K was established.

Naturally, much larger values of a, b, and p would be needed to make this example secure, since there are only 23 possible results of n mod 23. However, if p is a prime of at least 600 digits, then even the fastest modern computers cannot find a given only g, p and g^a mod p. Such a problem is called the **discrete logarithm problem**. Nowadays, for modulo p with length 2048 and more bits is not possible to find an exponent in reasonable time, what enables to make Diffie-Hellman algorithm trusted.

1.3.2 Discrete logarithm problem

A security of Diffie-Hellman protocol is based on a mathematical problem, which is called **Discrete logarithm problem**:

The ordinary logarithm problem example – given a base b and a number x, find y such that $b^y = x$. So, e.g., the logarithm to base 2 of 128 is 7. This is usually done by calculating the logarithm of x to base 10, and dividing that by the logarithm of b to base 10.

It is possible to do this in modular arithmetic too. Assuming to know, for a particular choice of n,x,b, that there is a y such that $b^y = x \pmod{n}$; then finding that y is the Discrete logarithm problem modulo n.

This will not be possible for all n,x,b, but (for instance) if n is a prime number, there is always a b that makes the problem solvable for every x other than 0.

The problem generalizes further. The integers modulo a prime number p form a finite field; there are other finite fields (exactly one of size p^k for each prime p and positive integer k), and we can pose the same sort of problem in any of them. The fields of size 2^k are particularly nice to work with using computers.

The Discrete logarithm problem is useful in cryptography, for the following reason: suppose n is large; then given n,b,y it is easy to find x, but no algorithm is known that, given n,b,x, will efficiently find y. So the function that takes y to x seems to be a "one-way function", much like the one that takes two prime numbers and yields their product. One-way functions are an essential building block for public-key cryptography. The difficulty of solving the discrete logarithm problem is essential for the security of the Diffie Hellman key exchange protocol and the ElGamal cryptosystem.
[3] [1] [14]

2 ELECTRONIC ID CARDS

Electronic ID cards as a form of personal identification documents are based on an electronic identification solution of citizens or organizations, for example in view to access benefits or services provided by government authorities, banks or other companies. Apart from online authentication many electronic ID documents also give users the option to sign electronic documents with a digital signature.

Electronic ID card is basically a physical identity card that can be used for online and offline personal identification or authentication. The electronic ID card is a smartcard in ID-1 format of a regular bank credit card, with identity information printed on the surface (such as personal details and a photography) and in an embedded RFID microchip or chip with contact pad. The format and physical characteristics of identification cards:

- physical dimensions (85,60 × 53,98 mm; thickness of 0.76 mm),
- resistance to bending, flame, chemicals, temperature and humidity,
- toxicity,
- resistance to heat,

are defined by international standard ISO/IEC 7810.

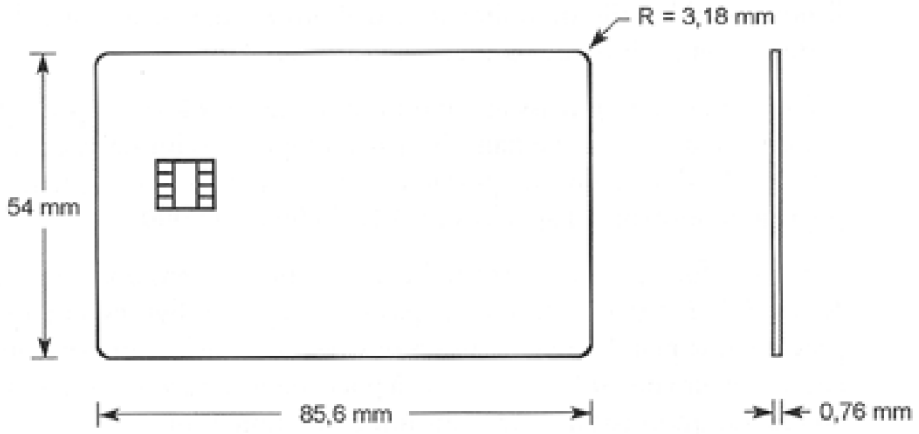

Fig. 2.1: An illustration of ISO/IEC 7810 card in ID-1 format.

The chip stores the information printed on the card (e.g. the holder's name and date of birth) and the holder's biometric photo. It may also store the holder's fingerprints. The card may be used for online authentication, such as for age verification or for e-government applications. An electronic signature, provided by a private company, may also be stored on the chip.

Electronic ID cards are sequentially provided by all governments in European Union. Countries that have already accepted government issued electronic ID cards

are Belgium, Bulgaria, Estonia, Germany, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Romania, Slovakia, Spain.

2.1 Smart cards

Smart card is any electronic ID card that has embedded integrated circuits. Depending on technical solution of a card, smartcard is either contact or contactless. There are several types of smart cards, these thesis is focused on those with chips.

Characteristics of smart cards

Every smart card consists of:
EEPROM – memory for storing applications and data; momery size is in kB
ROM – memory which contains OS running on a card; memory size is in kB
RAM – volatile memory used by microprocessor
CPU – processor of chip card; usual frequency in MHz

An advanced form of smart cards are multi application cards. Those cards support an option to load more application on a card and use them for more purposes. There is OS running on such cards, which provides memory management, application run, cryptographic functions and ensure a security. The most relevant and used multi application platforms are JavaCard and MULTOS. This master thesis deals with a card from UBM21-Z family and is developed by external company, nonetheless is running on MULTOS OS.

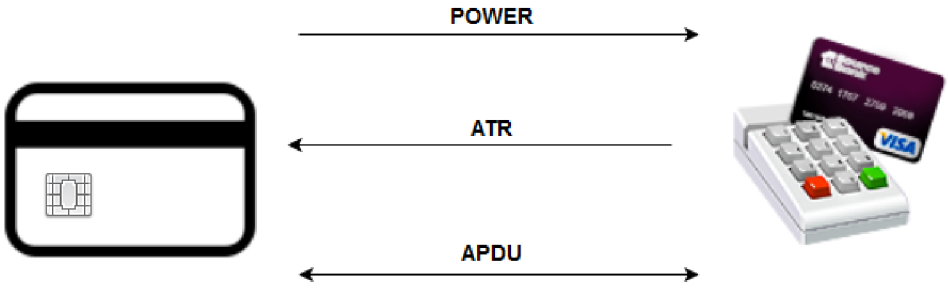

Fig. 2.2: Communication between a chip card and card reader

Communication between user and chip card is resolved by IFD – terminal (reader) for chip cards connected to PC. A part of IFD is also to power and control the communication between user's PC and card (figure 2.2).

2.1.1 Anatomy of Smart Cards with chip

Manufacture of smart card with chip consists of bringing layers from different materials properly together. Nowadays, the cards are made from PVC, Polyester or Polycarbonate. The card layers are printed first and then laminated in a large press. The next step in construction is the blanking or die cutting. This is followed by embedding a chip and then adding data to the card. In all, there may be up to 30 steps in constructing a card. The total components, including software and plastics, may be as many as 12 separate items; all this in a unified package that appears to the user as a simple device.[7]

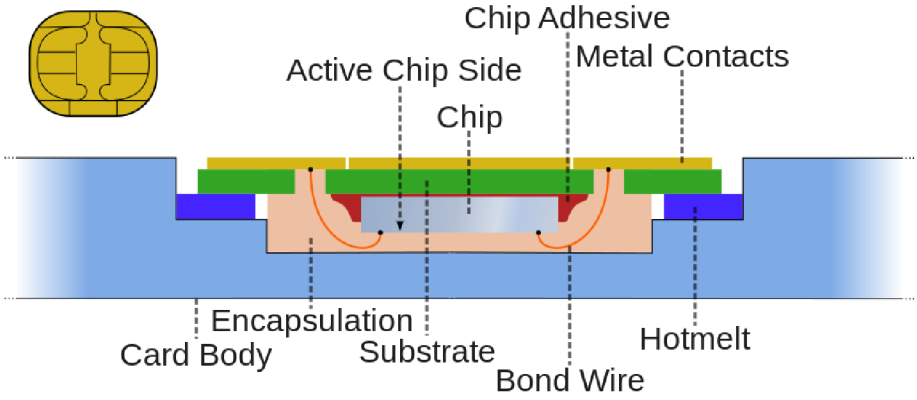

Fig. 2.3: A cross sectional view of the structure and packaging of a smart card chip.

2.1.2 ISO 7816 standard

co je ISO strucne Below are the most significant:

ISO 7816-1

Deals with physical physical characteristics of electronic ID cards, sets up limits for x-ray or UV based beams, electromagnetic fields or temperature of surrounded environment. Then determines parameters such as resistance against fold. This paragraph of ISO 7816 standard is important mostly for developers of electronic ID cards.

ISO 7816-2

Defines position, dimensions and functions of contacts on a chip. Electrical contacts located on the surface of the card are connected to a card reader when the card is inserted. This connector is bonded to the encapsulated chip in the card (picture 2.4).

Definition of **Chip pinout:**

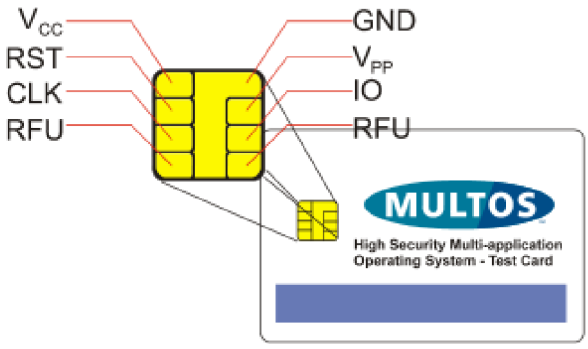

Fig. 2.4: A chip card[6]

- **V_{CC}**: Power supply
- **RST**: Reset signal (to reset the card's communication)
- **CLKn**: Clock signal (data communication timing)
- **RFU**: Reserved pin
- **GND**: Ground (reference voltage)
- **V_{PP}**: Programming voltage
- **I/O**: Serial input and output

ISO 7816-3

Specifies a communication between electronic ID card and card reader on low level (how are parameters for communication established). Then defines communication protocols used by electronic ID cards. States technical specification independently on used physical technology (for contact or contactless cards).

ISO 7816-4

Specifies data structures used during communication between electronic ID cards and card readers and defines an approach to data stored on cards. Solves options secured way of exchange of messages and safety card architecture. //pridat citaciu nejaku o el. ID kartach

2.1.3 Smartcard with MULTOS OS

Multos smart cards are working with applications which are sent and loaded to these cards by Multos utility – **MUTil**. Each application is identified and selected by its **application identifier (AID)**. AID and the naming convention of files in which

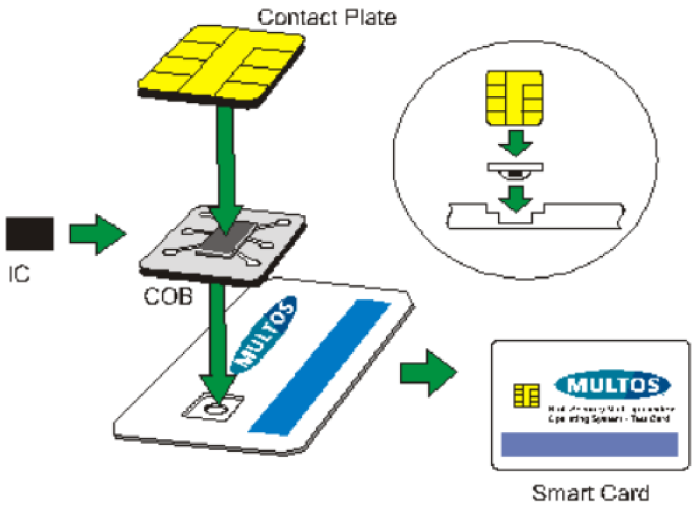

Fig. 2.5: A structure of smart card[6]

are applications kept are defined in ISO 7816. AID is a sequence of bytes and its length is between 5 and 16 bytes. The part of AID are:

- *National registered application provider* – known as a RID, its length is fixed at five bytes
- *Proprietary application identifier extension* – known as a PIX, its length is between zero and eleven bytes

The assignment of RIDs is controlled and delegated by ISO and must be unique. The assignments of PIXs is managed by companies using AID. An example of AID consisting of RID and PIX:

- RID: A0 01 02 03 04 (5 bytes)
- PIX: 00 01 (2 bytes)

Communication with a card

The process of a communication between an electronic ID card and card reader is defined as **answer to reset**. After the card is inserted to a card reader and power supply is on or receiving *RESET* signal, card sends IFD chain called as **ATR**. ATR consists of two parts – **interface characters** and **historical characters**. Interface characters define parameters of a communication while the connection with IFD is initializing. Those parameters are supported transport protocols or frequency of an hourly impulse. Historical characters are may be specific for every single card and may contain information, for instance about manufacturer of the card or a balance on electronic wallet. MULTOS OS enables two types of ATR – primary, which is sent when a card is inserted to a card reader and power supply is on, and secondary,

which is received when the *RESET* signal is confirmed as obtained.

For execution of every application are required commands. Commands are formatted and transmitted in the form of **application protocol data units (APDU)**. APDUs represent the way how a card and a card reader communicate and are completely defined in ISO 7816. During the communication between a card and a reader, there are two kinds of APDUs recognized – a **command APDU (C-APDU)** and a **response APDU (RAPDU)**. C-APDU is the command sent to a card and RAPDU returns the execution result of the command to a reader. These versions of APDU are exchanged alternately, based on request-response principle. C-APDU consists of a mandatory header with length four bytes and variable-length conditional body:

- **CLA** — *Class of instruction.* A mandatory single byte that indicates the structure and format for a category of command and response APDUs.
- **INS** — *Instruction code.* A mandatory single byte that specifies the instruction of the command.
- **P1 and P2** — *Instruction parameters.* Two mandatory single-byte parameters that provide qualifications to the instruction.
- **Lc** — *Length of command data.* An optional single byte indicating the number of bytes present in the data field of the command.
- **Data field.** An optional sequence of Lc bytes in the data field of the command.
- **Le** — *Length of expected response.* An optional single byte indicating the maximum number of bytes expected in the data field of the response to the command.

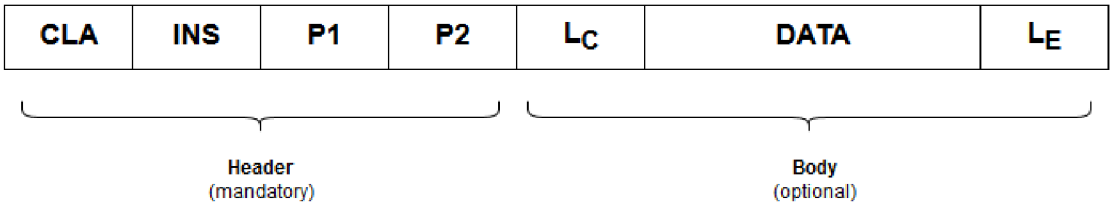

Fig. 2.6: CAPDU structure

RAPDU consists of a variable-length conditional body and a two-byte mandatory trailer:

- Data field. An optional sequence of bytes received in the data field of the response.
- SW1 and SW2 — Status words. Two mandatory single-byte values that denote the processing state in the card.

Applications reply to each APDU command with a status word indicating the result of the operation, and optionally with data. To activate an application on

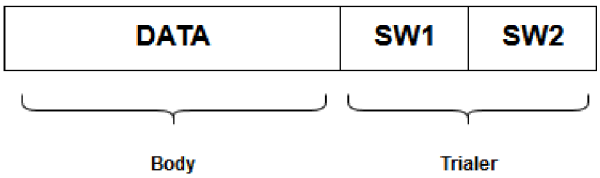

Fig. 2.7: RAPDU structure

Tab. 2.1: MULTOS status words with a description

Status Word	Description
90 00	Successful processing
61 xx	Successful processing where xx bytes of unexpected data are returned
6C xx	Successful processing, but Le value and La value are different. Send the command again with an Le value of xx.
6A 82	File not found
6A 83	Record not found
6B 00	Wrong parameter P1P2
6D 00	Instruction not supported
6E 00	Class not supported
6F 00	Unknown error

a Multos smart card is necessary to insert SELECT command. After receiving SELECT command follows the process of searching for the application whose AID matches the one specified in the command. If there is a match, the application is getting prepared to be selected (in a case the application is already selected, Multos deselects it). When an application is selected, Multos forwards all subsequent APDU commands (including further SELECT commands) to the application. In the main method, the application interprets each incoming command APDU and performs whatever is requested by the command. For each command APDU sent, the application responds by sending back a response APDU.CITE The command-and-response dialogue continues until a new application is selected or the card is removed from the reader. Once the application is deselected, it becomes inactive until it is selected the next time.

Technical specification:

- Power $3 \sim 5\ V$
- External clock $1 \sim 10\ MHz$
- Application Header: 48K Bytes
- Total temporary space per protected ALU: 16 Bytes

[4] [**?**]

3 MULTOS

MULTOS is a multi-application smart card operating system, that enables a smart card to carry a variety of applications, from chip and pin application for payment to on-card biometric matching for secure ID and ePassport. MULTOS is an open standard whose development is overseen by the MULTOS Consortium – a body composed of companies which have an interest in the development of the OS and includes smart card and silicon manufacturers, payment card schemes, chip data preparation, card management and personalization system providers, and smart card solution providers. There are more than 30 leading companies involved in the consortium.

One of the key differences of MULTOS with respect to other types of smart card OS, is that it implements a patented public key cryptography-based mechanism by which the manufacture, issuance and dynamic updates of MULTOS smartcards in the field is entirely under an issuer's control using digital certificates rather than symmetric key sharing. This control is enabled through the use of a Key Management Authority (KMA), a special kind of certification authority. The KMA provides card issuers with cryptographic information required to bind the card to the issuer, initialize the card for use, and generate permission certificates for the loading and deleting of applications under the control of the issuer.

Application providers can retrieve and verify the public key certificate of an individual issuer's card, and encrypt their proprietary application code and confidential personalisation data using that card's unique public key. This payload is digitally signed using the private key of the application provider. The KMA, on request from the card issuer, signs the application provider's public key and application code has and creates a digital certificate (the Application Load Certificate) that authorises the application to be loaded to an issuer's card or group of cards. Applications are therefore protected for integrity and confidentiality and loaded to a card without any party sharing symmetric keys and therefore needing to trust any other party sharing the card platform – including the card issuer. Both the Application Provider and Card Issuer know that only specific, authorised applications from authorised parties can be loaded to any specific card.

Hundreds of millions of MULTOS smart cards have been issued by banks and governments all around the world, for projects ranging from contactless payment, Internet authentication and loyalty, to national identity with digital signature, ePassport with biometrics, health care and military base and network access control.

[9]

3.1 Memory structure

While developing an application using MULTOS OS, it is fundamental to get understand how memory on ID card works and what is the approach to it. Every application on a card has its own memory space, which is secured by a firewall – it does not approach directly to physical memory, but through AAM (Application Abstract Machine). It allows to ensure the application on a card cannot approach to other application's memory space. A card's memory space is divided into two independent parts – **code space** and **data space**.

3.1.1 Code space

Code Space refers to the memory space occupied by the application's code, which can not be accessed to read or write, but rather can only be interpreted by the AAM. Physically the code space is a block of memory that consists of up to 64 KB of contiguous, non-volatile memory.

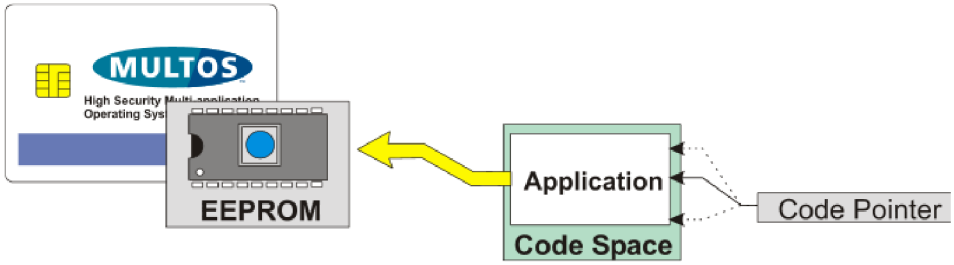

Fig. 3.1: Application code space[6]

Segment addresses within the code space are always relative to the application. So, the starting offset is always zero and, furthermore, application execution always starts from the first byte. Now, when an instruction is being executed the Code Pointer register contains the code address of the next instruction to be executed. The value that is held in this register is affected when using program flow instructions such as Jump, Branch, Call and Return instructions. However, the code pointer value is not available for manipulation by an application. [6]

3.1.2 Data space

Data Space contains all of the data that is addressable by the application and consists of three distinct memory areas. Those areas are:

1. non-volatile **Static memory**,
2. RAM based **Public memory**,

3. RAM based **Dynamic data**.

The latter can be composed of application specific session data and the stack. Data Space can be no more than 64 KB in size and is addressed from 0 using tagged addresses.

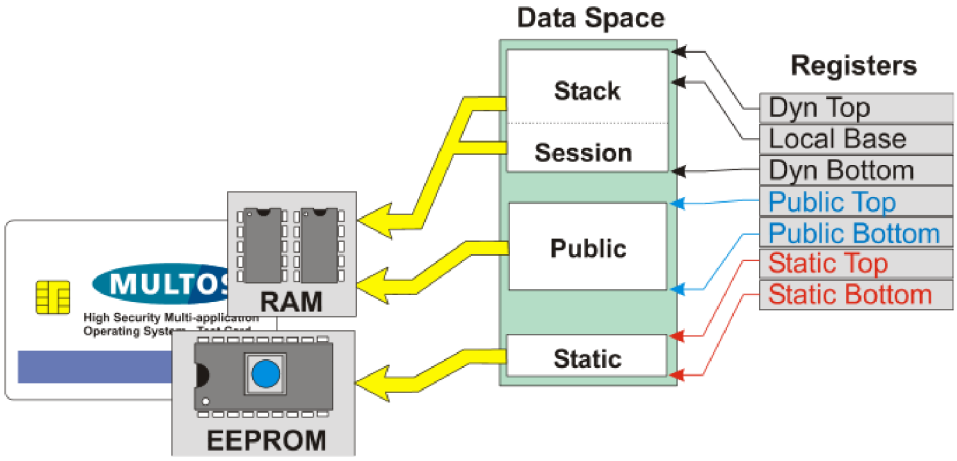

Fig. 3.2: Application data space[6]

Static memory

Static data is the non-volatile memory of the MULTOS Card. Static memory is private to the application and cannot be accessed by the terminal or other applications. Therefore it is used as a storage of private data of applications such as encryption keys. Static memory is usually implemented by EEPROM memory. It is important to avoid corrupting static memory. There is a limited amount of space for applications and corrupted memory can render an application useless. There are two OS supplied mechanisms that eliminate this problem: data item protection and transaction protection. Data Item Protection is always used. MULTOS guarantees that in executing an instruction that writes data to static memory it will either be completely updated or not updated at all. Transaction Protection is controlled using the Set Transaction Protection primitive – this is an OS supplied mechanism for caching a number of writes to memory. They can then be committed or discarded. MULTOS guarantees that once a commit has started then all affected writes are all made. This behaviour holds even if there is a loss of power during the writing of the data. [6]

Public memory

The Public memory area is the RAM resident input / output buffer for applications. Incoming APDU are held in Public and any outgoing status word (SW), L_A and data are placed here. This buffer is also used to pass information from one application to another when delegation is used. As an I/O buffer it is visible to IFD. During the command-response dialogue MULTOS passes an APDU to an application the APDU is written into Public memory. The APDU Header appears at the top of Public, and command data appears at the bottom of public. When a MULTOS Application wishes to pass a response back to the terminal then the response APDU is written into Public and MULTOS sends the response to the terminal. MULTOS guarantees that data in Public remains private to the application until it exits or delegates to another application. So, public may be used as temporary workspace. MULTOS will automatically clean up the public area if the application terminates abnormally, but will not do so otherwise. This means that any data held in Public that an application does not wish to reveal after exiting should be explicitly erased. [6]

Dynamic memory

Dynamic data is volatile and held in the RAM memory of the MULTOS Card. Like the other areas, dynamic memory is behind a firewall and private to the application. Unlike the other areas, this memory can consist of two parts: session data and the stack. Session Data is RAM based application variables, which are available to any function used in the application. The size of the session data area is fixed when an application is loaded onto a MULTOS Card and will always appear at the bottom of the dynamic area. Session Data, however, is not mandatory and if none is used, then none will be present. The stack is an application's work area. A MULTOS chip is a stack machine, which means that this memory area is used to perform many functions. For example, most primitives and many instructions use stack-based values as input.

3.2 SmartDeck

SmartDeck is an application development system for Multos, which enables to code applications using C and assembler programming languages. These applications run on the Multos smart card operating system under high security. SmartDeck, as a programming extension, runs strictly on Microsoft Windows operating system and is based around the Eclipse IDE. Programs which are prepared on these host

machines using the tools in the SmartDeck package are not executed on a host, but on a Multos smart card.

Credibility of an implementation of SmartDeck in *C* programming language is guaranteed by the ANSI and ISO standards. Basically, Multos SmartDeck environment brings enhancements and improvements to extend authentic *C* programming language. SmartDeck also contains assembly language. As well as providing a cross-compilation technology, SmartDeck provides a PC-based simulation which is interfaced with the Eclipse IDE allowing to debug application quickly. A set of tools for generating standard application load units and facility for loading and deleting applications on Multos smart cards provide the final stage of the software development life-cycle. [5]

3.2.1 SmartDeck components

SmartDeck includes a several programs which are separated but at the end work together through proprietary files and create a chain for developing and testing the applications for smart cards. Multos uses various file formats.

File formats used by SmartDeck:

- `.hzo` – *Object files* – contain compiled program code
- `.hza` – *Library files* – collections of object files
- `.hzx` – *Executable files* – fully linked executable program
- `.alc` – *ALC files* – contain Application Load Certificate
- `.adc` – *ADC files* – contain Application Delete Certificate
- `.alu` – *ALU files* – contain Application Load Unit

Object files can be created:

- From assembly language source code using the has program
- From C source code using the hcc program

Library files are created using the *har* program. Object and library files are linking together using the *hld* program to create an executable file. Executable files can be loaded and debugged on the MULTOS debugger/simulator programs *mdb* and *hsim*. Executable files can also be loaded onto MULTOS cards using the *hterm* program. MULTOS application load units can be generated from executable files using the *halugen* program. The use of the C compiler, assembler, linker, and even the archiver is coordinated by the compiler driver, hcl.exe so you won't need to use these programs directly.

Tab. 3.1: The supported formats used by MULTOS

hcl.exe	**Compiler driver:** provides a useful way to get files compiled, assembled and linked
hcc.exe	**C compiler:** compiles modules written in C
mdb.exe	**Eclipse gdb/mi debugger:** Provides the debugging interface between Eclipse and the MULTOS simulator.
has.exe	**MULTOS Assembler:** assembles modules written in assembly language
hld.exe	**Linker:** Required for linking compiled and assembled files
hsim.exe	**MULTOS Simulator:** used in conjunction with mdb but can also be used stand-alone.
hterm.exe	**Loader:** Used to load and delete application from MULTOS cards
har.exe	**Archiver:** Consolidates multiple object files into a single, object code library
hls.exe	**Object file lister:** displays useful information held in unlinked files and linked executables.
hkeygen.exe	**RSA key pair generator:** creates a private and public RSA key pair
halugen.exe	**ALU generator:** creates a standard MULTOS application load unit.
melcertgen.exe	**ALC/ADC generator:** creates load and delete certificates for developer cards.
meldump.exe	**MULTOS file list:** outputs contents of standard MULTOS files.
hex.exe	**Extractor utility:** used to prepare images in various formats

3.2.2 MUtil

The MULTOS Utility application – **MUtil** provides a single application that handles the different functions needed when working with MULTOS cards. The latest release (version 2.9.1) includes the following features:

- Loading and deleting of applications for developer cards
- Loading and deleting of applications for live cards
- Creating an MCD ID list for enrollment data/MSM Control Data requests
- Enable cards using KMA supplied MSM Control Data
- Exchanging APDUs
- Running of scripts from the command line
- MULTOS 4.4 / 4.5 support
- Optional output of a trace file
- INI file for settings

During a development of the application for smart card, Mutil was used mainly for loading the applications on the smart card and testing APDU commands (see the screenshots in the pictures 3.3 and 3.4).

MUtil supports developing cards as well as "live cards" and is able to load and delete applications using ALC and ADC certificates. It is recommended to use just one tool for uploading of applications – either `hterm` as a part of SmartDeck or MUtil. Whereas it may come to inconsistency and accidentally the same application would be loaded twice, `hterm` is not then able to delete that application or to load another one with the same AID. A solution of this issue is to search for all loaded applications on a card using "FindFirst" and "FindNext" functions on "Delete Test" tab in MUtil. Since the application is found in the way of comparing all possible AIDs, it is possible to delete the selected application.

[10]

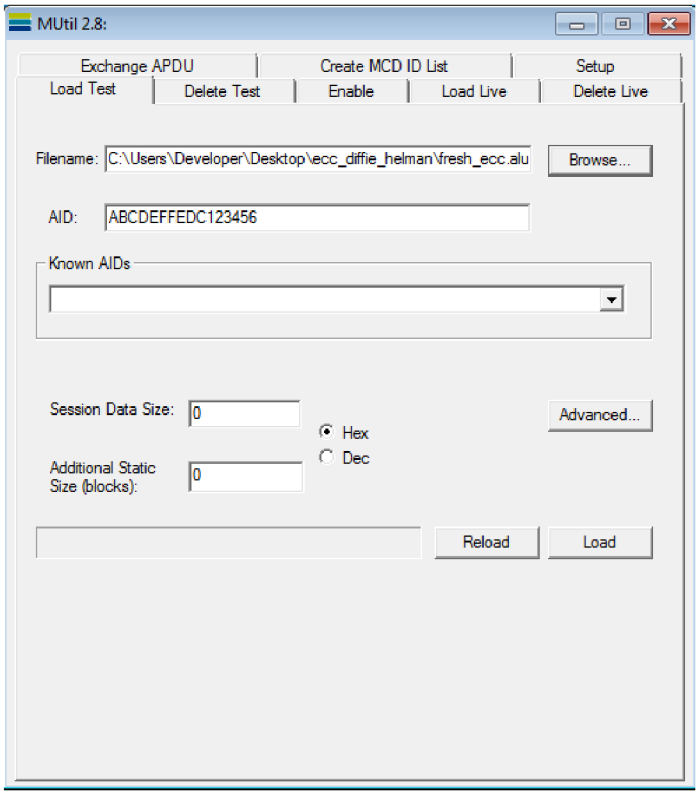

Fig. 3.3: A feature of Mutil for loading the applications on the smart card

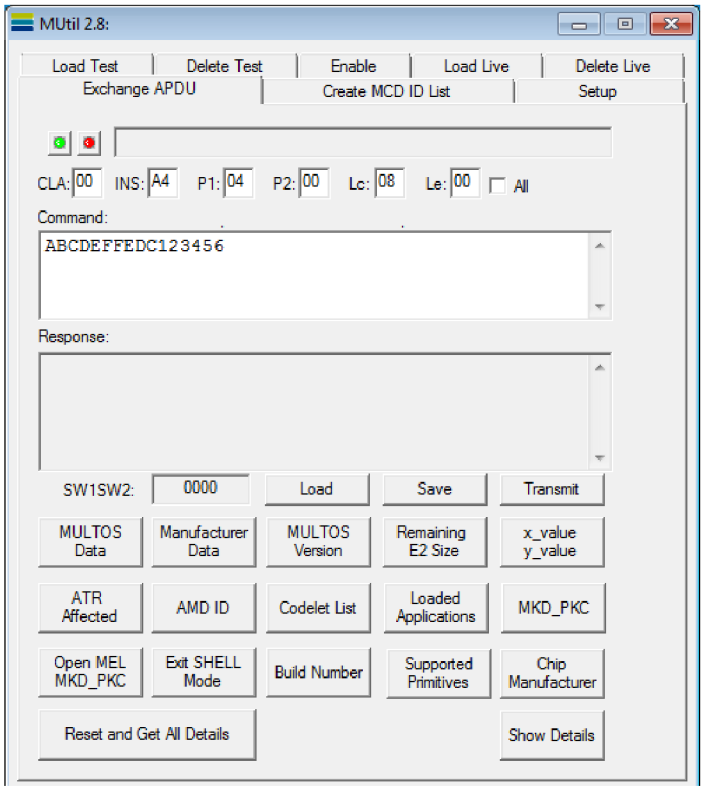

Fig. 3.4: A feature of Mutil for loading the applications on the smart card

4 PRACTICAL RESULTS

This chapter is focused on practical application and implementation of the cryptographic protocol based on ECC – Diffie-Hellman. Hereby is demonstrated, that this algorithm is confident to establish a reciprocal secret key between two users. For these purposes, the applications in language C and language Java had to be built.

Fig. 4.1: The smart card UBM21-Z48 and the terminal.

For physical connection of the card to a PC, the card reader for chip smart cards was used (picture 4.1). The connection of the reader to PC is through USB cable with standard USB 3.0 connector. Thanks to this, the reader is powered from the PC.

4.1 Card application

The application running on the smart card is written in language C and uses some MULTOS functions from header file `multos.h` as displayed in C code definition part:

```
#include <multos.h>
#include <string.h>
```

As stated in section 1.3.1, it is obvious that principle of Diffie-Hellman algorithm is demonstrated by using mathematical operations based on modular arithmetic. However, Diffie-Hellman algorithm is based on ECC, therefore during the implementation and development of the application in programming language C it uses scalar multiplication for its operations as it is showed in the picture 4.2 It was men-

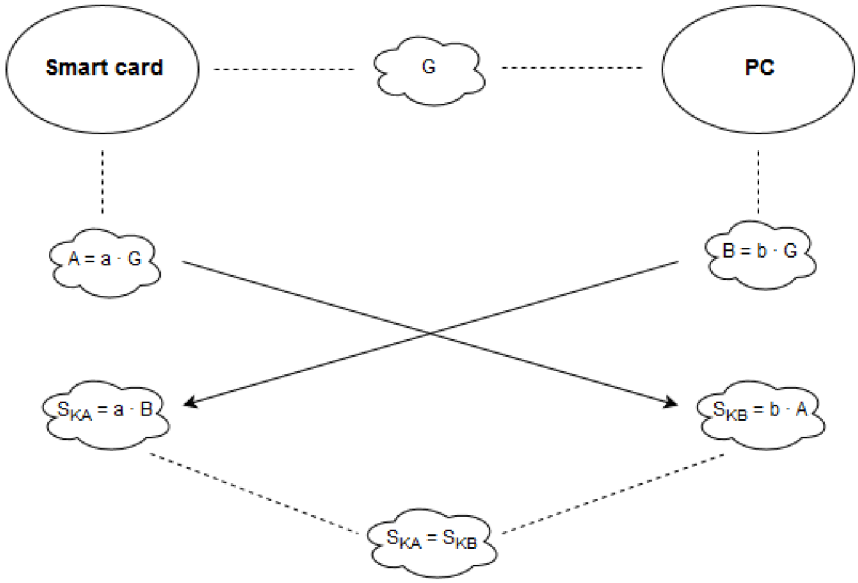

Fig. 4.2: Diffie-Hellman operation scheme

tioned in the section 2.1.3 that every single application running on the card must be represented by its name – AID. The AID is implemented in the beginning of the code as well as set up of the log in DIR:

```
#pragma attribute("aid", "ABCDEFFEDC123456")
#pragma attribute("dir", "61 1c 4f 04 AB CD EF FE DC 12 34 56 50 14
43 52 33 31 32 20 45 43 43 20 50 72 69 6d 69 74 69 76 65 73")
```

4.1.1 Parameters of elliptic curve

The definition of the elliptic curve created for an application running on the smart card is characterized by these parameters, which are firmly fixed:

```
0xAC, 0x75, 0xCF, 0x35, 0x99, 0x88, 0x5A, 0x6A, 0x26, 0xB2,
0x0F, 0x52, 0x71, 0xAB, 0x95, 0xA3, 0xF0, 0xD2, 0x4B, 0x74,
0x37, 0x21, 0x46, 0xCC, 0xDB, 0xA0, 0x5F, 0xA9,  // P
0x93, 0x62, 0xE8, 0xF2, 0x7B, 0xDC, 0xA9, 0x6F, 0x81, 0xE6,
0xBF, 0xA6, 0x79, 0x5E, 0x10, 0x60, 0xA9, 0x69, 0xD2, 0x0D,
0x9F, 0x88, 0x2E, 0xB4, 0xD8, 0xE8, 0xD4, 0x20,  // A
```

```
0x89, 0xD8, 0x66, 0x9D, 0x59, 0x20, 0x5C, 0xB4, 0xA3, 0x6E,
0xEC, 0x01, 0x22, 0xC6, 0x49, 0x1C, 0x92, 0xB6, 0x18, 0xB8,
0xFC, 0x09, 0xB6, 0xD6, 0xF3, 0x24, 0xAA, 0xCA,  // B
0x2E, 0xDA, 0x6A, 0x9C, 0xE8, 0x53, 0x3B, 0xBC, 0xB8, 0x1D,
0x49, 0xF4, 0x69, 0xB5, 0x43, 0x95, 0xD3, 0x1A, 0x64, 0xB8,
0x14, 0x8B, 0x92, 0xB3, 0x6B, 0xC0, 0x23, 0x00,  // Gx
0x84, 0xDA, 0x69, 0x9D, 0xF7, 0x56, 0xBF, 0x58, 0xC9, 0x50,
0x76, 0x7A, 0xD7, 0xF8, 0x84, 0x62, 0x1E, 0x2F, 0x5C, 0xFC,
0x28, 0x25, 0x97, 0x99, 0x14, 0x05, 0xB2, 0x4D,  // Gy
0x0F, 0xAD, 0x9E, 0x79, 0x3C, 0x80, 0xC2, 0x66, 0xBD, 0xB3,
0x18, 0xAA, 0x67, 0x6C, 0x9E, 0xDB, 0x4F, 0xB6, 0x53, 0xCF,
0x4F, 0x67, 0x92, 0x37, 0x13, 0x37, 0x56, 0xA1,  // N
```

The stated parameters are:

- **P** – the characteristics of a prime number
- **A** – the characteristics of a point A lying on the curve
- **B** – the characteristics of a point B lying on the curve
- $\mathbf{G}_x$ – the characteristics of x axis of a base
- $\mathbf{G}_y$ – the characteristics of y axis of a base
- **N** – the characteristics of an order of the curve

The length of the curve is 28 bytes and every byte is characterized in hexadecimal numeral system (according low-level programming in plain C language and data representation on the smart card). A strength of the cryptographic functions and primitives depends on the length of curve. However this length is preferred as a good compromise between the strength and application size.

The parameters P, A, B and N are used on background, other parameters are are actively used. According Diffie-Hellman operation scheme of scalar multiplication (section 4.2), it is conspicuous an addition parameter is required – **multiplier**. The multiplier presents the private key in the Diffie-Hellman operation scheme and is generated randomly and must be smaller than order N:

```
GenerateRandomNumberLessThan(order, (BYTE *)private_key);
EccMultiplyPointByScalar(domain_params, generator_point,
                         /*multiplier=*/private_key,
                         /*result=*/public_key);
ReturnResponse(2*ECC_LEN+1, public_key, ERR_OK);
      break;
```

The multiplier could be possibly firmly fixed in the code as well, but regarding the tendency to set up the highest achievable security of the private key, implemented solution is preferred.

4.1.2 Functions

To obtain desired results from the smart card, some MULTOS functions had to be used. To generate the secret key on side of the card, the `GENERATE_SHARED_SECRET_INS` is called:

```
case GENERATE_SHARED_SECRET_INS:
   if (!CheckCase(3)) multosExitSW(ERR_BAD_INS);
     GenerateSharedSecret(apdu_data);
     ReturnResponse(0, NULL, ERR_OK);
     break;
```

This functions uses scalar multiplication to multiply required parameters with private or public key:

```
void GenerateSharedSecret(BYTE *other_public_key) {
  EccMultiplyPointByScalar(domain_params, other_public_key,
                           private_key, shared_secret);
```

After the secret key is established, this key is supposed to use and therefore it must be kept in private and non-readable for others. For this purpose mathematical operation XOR was used as a demonstration of such a security. This has to be implemented on the application card as well:

```
case ECHO_INPUT_INS:
      if (!CheckCase(4)) multosExitSW(ERR_BAD_INS);
      EncryptSymmetrically(apdu_data, Lc, result);
  ReturnResponse(Lc, result, ERR_OK);
      break;
```

The operation XOR is implemented very simple hereby:

```
void XorMessage(BYTE *msg, WORD len, BYTE *shared_secret,
                WORD shared_secret_len, BYTE *result) {
  int i;
  for (i = 0; i < len; ++i) {
    result[i] = msg[i] ^ shared_secret[i % shared_secret_len];
  }
}
```

4.1.3 Supported primitives

This master thesis deals with the card UBM21-Z48, which supports specific primitives and because of that it was necessary to adjust the code in SmartDeck. The

primitives listed in attachment A of this thesis are those that are included in the target specification. As it is obvious, the card UBM21-Z48 does not support the Multos *GenEccKeyPair* primitive and therefore it was crucial to generate the private and public key.

4.2 Client application

The application for the communication with the card is written in Java programming language. This application physically connect to the smart card and exchange the APDU messages between a client and smart card. This application demonstrates all implemented functions on the smart card.
The source file of client application contains of three Java files:

1. *DiffieHellmanMain.java* – main code
2. *ApduSender.java* – code for sending and exchanging APDU messages
3. *RawDataSender.java* –code for sending raw data

The client application consists of several methods and functions, its main role is to connect to the card, communicate with the card using APDU messages and receive and display the answer from the application loaded on the card.
This application written in Java language shows the result of the code after it was run in the console as it is stated in section below.

4.2.1 Result

After the Java code is debugged and built, the console displays this result: In

```
ESTABLISHING CONNECTION WITH THE CARD
Terminals: [PC/SC terminal Gemplus USB SmartCard Reader 0]
Selected Terminal: PC/SC terminal Gemplus USB SmartCard Reader 0
ATR: 3B6F00008031E06B8421020700555555555555

Card info: PC/SC card in Gemplus USB SmartCard Reader 0, protocol T=0, state OK
APDU >>>: 00A4040008ABCDEFFEDC123456
APDU <<<: 9000 SW =9000
It took 112117322 nanoseconds to execute the APDU.
GENERATING PRIVATE KEY
GENERATING PRIVATE/PUBLIC KEY PAIR ON CARD
APDU >>>: 8001000000
APDU <<<: 04130ABC4FAB86C12FB42A59DAFD317AD838976E07091AEFB7A5F2D1AF7C52707BDF5433F8A32CB950429DDE05743E99A0CECFC50E9B827F159000 SW =9000
It took 682298728 nanoseconds to execute the APDU.
GENERATING LOCAL SHARED SECRET
Sending command to card to generate shared secret
APDU >>>: 8002000039041755827760 27D8CCF3A09355F060D422A5D50DB3D41C49D1B48339DF083589EDCF1984CADFA9F9F5AC5D1CC8D25AC7250857DCBE4758BB19
APDU <<<: 9000 SW =9000
It took 172942651 nanoseconds to execute the APDU.
Verifying shared secret using random string.
Sending a random string for encryption.
APDU >>>: 800300001474C93E80ADE0158A806BB267E858D2C8C46F06BD
APDU <<<: 708E304D3C14CCA66E76F5CE5FE4A709DD1118169000 SW =9000
It took 449784540 nanoseconds to execute the APDU.
SUCCESS: Returned encrypted string after decryption equals the original one
```

Fig. 4.3: Output of the client application

the very first step, client sends the C-APDU message with AID of the application.

Naturally, client must know in advance which application wants to use and therefore this information is mandatory to input. After the application with given APDU is found, client receives R-APDU message from the smart card with SW 9000. This means application was found and the communication is established.
Then, to calculate shared secret key, it is necessary to generate the private key. As mentioned above, the private key is generated randomly and to obtain it from the smart card, client sends C-APDU 80 01 00 00 00 and receives generated private key. After the private key is generated on both sides (card and client), a counting of the shared secret key can be launched. This is happening again on both sides. When the client receives R-APDU 9000 from the client, then the shared secret keys are compared. A value of the secret key is not displayed in console purposely, because this information is considered as a top secret. Generated secret keys must be compared and therefore the XOR operation is used to secure the transfer of these keys between both sides. It is recommended to use symmetric cryptography to send established shared secret key to make sure it will remain secret after it was created.[1]
The final output from console is: **Returned encrypted string after decryption equals the original one** and proves, that the common secret key is the same on both sides.

4.3 Measurement

To evaluate the application of the implemented code on the smart card, the speed testing of cryptographic protocol Diffie-Hellman was performed. The measuring was implemented in the client application in Java and several values are captured:
GEN_NEW_KEYPAIR_INS: 640ms, 529ms, 701ms, 599ms, 640ms
GENERATE_SHARED_SECRET_INS: 568ms, 530ms, 580ms, 610ms, 530ms
ECHO_INPUT_INS: 272ms, 268ms, 240ms 290ms, 288ms
ENCRYPT_INS: 288ms, 248ms, 233ms, 266ms, 217ms
The measuring was focused on end-to-end communication, so it means that measured values involves the latency of the data transfer and other factors. The client application measures the time of these primitives after every run, because of that this measurement may differ and stated data represents values measured five times in a row.

4.4 Problems and issues

During the development of the applications, some issues appeared:

[1] For instance DES or SHA cipher.

1. MULTOS SmartDeck is compatible just with Windows 7 32-bit OS, therefore it was necessary to install a virtual machine and run this version of MS Windows.
2. After the smart card is plugged in the card terminal, an installation of drivers starts in MS automatically. The problem occurred when the PC wanted to install the driver of the smart card – unsuccessfully. The specific driver is not publish publicly on internet and it looked like there will be a problem with the communication. However Mutil does not require the driver for the communication with the card, so at the end it did not influence the development of the application.
3. It is very important to study the technical parameters of the smart card, especially memory size and supported primitives. If a memory size is exceeded or a unsupported primitive is used, it might be difficult to find where the problem is, because MULTOS does not inform a developer about all errors and usually returns just very general status words (in some cases it even does not return any error status word).
4. By the SmartDeck manual, the application should be developed in the Eclipse and to generate *.hzx* file developer is supposed to debug the code all the time. It takes too much time and it is not so practical for development of the application, which has to loaded using external software. Therefore, the much better solution is to use command line programs from MULTOS and create scripts. Then a developer does not have to debug the code and can use some other software for writting the code, for instance Notepad++.

Nonetheless, all mistakes and issues have been resolved and development was successful.

5 CONCLUSION

The role of this master thesis was to implement chosen cryptographic protocol on the smart card and its further usage for electronic ID cards. As preferable protocol was used Diffie-Hellman, which was theoretically described and characterized in detail as well as was graphically displayed its arithmetic principle. Diffie-Hellman is based on ECC, the very first chapter covers also this problematic together with an overview of asymmetric cryptography.

The goal was to implement such a cryptographic protocol on a specific smart card based on MULTOS OS. For better understanding how does this OS influence the development of the application for smart cards, the second chapter describes smart cards and the third chapter shows the MULTOS principle.

Practical part of this thesis is focused on the implementation of Diffie-Hellman protocol based on ECC and running on MULTOS OS. The output of this implementation is a software, specifically two codes from two different programming environments – Java (client application) and C (card application). The implementation was successful in the final and was tested in a console application of Java. All source files are included to this project.

At the end, testing of a time of partial primitives used in this thesis was provided, processed and analyzed. Therefore, all targets of this master thesis are considered as schieved.

BIBLIOGRAPHY

[1] MENEZES, A., C. VAN OORSCHOT P., A. VANSTONE S. *Handbook of applied cryptography.* Boca Raton: CRC Press, c1997. Discrete mathematics and its applications. ISBN 0-8493-8523-7.

[2] ROUSE, M. *Asymmetric cryptography (Public Key Cryptography)* [online]. 2001, poslední aktualizace 11. 11. 2004 [cit. 17. 2. 2005]. Online: <http://searchsecurity.techtarget.com/definition/asymmetric-cryptography>.

[3] Encryption and Decryption.com *Asymmetric Algorithms* [online]. 2001, poslední aktualizace 11. 11. 2004 [cit. 17. 2. 2005]. Online: <http://www.encryptionanddecryption.com/algorithms/asymmetric_algorithms.html#Diffie-Hellman>.

[4] Multos *UBM21-Z Family – External Characteristics.* [online]. Online: <https://www.multos.com/products/approved_platforms/MIR/ubivelox/umb21>.

[5] Multos *SmartDeck Developer's Reference Manual.* Version 3.0.1 [online]. <multos.com/downloads/technical/smartdeck-manual.pdf>.

[6] Multos *MULTOS Developer's Guide.* [online]. <http://www.multos.com/uploads/MDG.pdf>.

[7] Smart Card Basics *Types of Smart Card.* [online]. <http://www.smartcardbasics.com/smart-card-types.html>.

[8] TechTarget *Cryptography.* [online]. <http://searchsoftwarequality.techtarget.com/definition/cryptography>.

[9] MULTOS *Further MULTOS information.* [online]. <http://www.multos.com/multos_explained.htm>.

[10] MULTOS *MULTOS Utility Manual.* [online]. <http://www.multos.com/uploads/MUM.pdf>.

[11] Cryptography – Individual Contributors *Asymmetric algorithms.* [online]. <https://cryptography.io/en/latest/hazmat/primitives/asymmetric/>.

[12] Digicert *Behind the Scenes of SSL Cryptography.* [online]. <https://www.digicert.com/ssl-cryptography.htm>.

[13] TechTarget *Elliptical curve cryptography (ECC).* [online]. <http://searchsecurity.techtarget.com/definition/elliptical-curve-cryptography>.

[14] Math and cryptography *Discrete Logarithm Problem.* [online]. <http://wiki.c2.com/?DiscreteLogarithmProblem>.

LIST OF SYMBOLS, PHYSICAL CONSTANTS AND ABBREVIATIONS

AAM	Application Abstract Machine
ADC	Application Delete Certificate
AID	Application identifier
ALC	Application Load Certificate
ANSI	American National Standards Institute
APDU	Application Protocol Data Unit
BLE	Bluetooth Low Energy
ECC	Elliptic Curve Cryptography
EEPROM	Electrically Erasable Programmable Read-Only Memory
ID card	Identification Card
IFD	Interface Device
ISO	International Organization for Standardization
I/O	Input/Output
kB	kilo Bytes
L_A	Length of actual data
MS	Microsoft
NFC	Near Field Communication
OS	Operation System
PC	Personal Computer
RAM	Random Access Memory
XOR	eXclusive OR
ZIP	newer compression algorithms

LIST OF APPENDICES

A ATTACHMENTS

The following tables displays the supported primitives of used smart card based on Multos OS:

Primitive	Supported	Optional / Mandatory
AddBCDN	Yes	
BitManipulateByte	Yes	
BitManipulateWord	Yes	
CallCardBlock	Yes	
CallCodelet	Yes	
CheckCase	Yes	
Checksum	Yes	
ControlAutoResetWWT	Yes	
Delegate	Yes	
DESECBDecipher	Yes	
DESECBEncipher	Yes	
DivideN	Yes	
EccAdd	Yes	
EccConvert	Yes	
EccEqual	Yes	
EccInv	Yes	
EccMult	Yes	
EccVerify	Yes	
ExchangeData	No	
GenerateAsymmetricHashGeneral	Yes	
GenerateAsymmetricSignatureGeneral	Yes	

GenerateDESCBCSignature	Yes	
GenerateRandomPrime	Yes	
GenerateTripleDESCBCSignature	Yes	
GetData	Yes	
GetDelegatorAID	Yes	
GetDIRFileRecord	Yes	
GetFileControlInfo	Yes	
GetManufacturerData	Yes	
GetMemoryReliability	Yes	
GetMULTOSData	Yes	
GetPurseType	Yes	
GetRandomNumber	Yes	
LoadCCR	Yes	
Lookup	Yes	
Memory Compare	Yes	
Memory Compare Fixed Length	Yes	
Memory Copy	Yes	
Memory Copy Fixed Length	Yes	
MemoryCopy	Yes	
Modular Exponentiation	Yes	
Modular Exponentiation CRT	Yes	
Modular Inverse	Yes	

ModularMultiplication	Yes	
ModularReduction	Yes	
MultiplyN	Yes	
Query0, Query1, Query2, Query3	Yes	
QueryChannel	No	
QueryCodelet	Yes	
QueryInterfaceType	Yes	
ResetSessionData	Yes	
ResetWWT	Yes	
ReturnFromCodelet	Yes	
SEEDECBDecipher	Yes	
SEEDECBEncipher	Yes	
SetAFI	No	
SetATRFileRecord	Yes	
SetATRHistChars	Yes	
SetATSHistChars	No	
SetFCIRecord	Yes	
Set Select SW	Yes	
Set Transaction Protection	Yes	
SHA-1	Yes	
ShiftLeft	Yes	
ShiftRight	Yes	
StoreCCR	Yes	
SubtractBCDN	Yes	

B CONTAIN OF ATTACHED CD

The final applications for the smart card and client are saved on the attached CD and are situated in *diplomka_final* folder. This folder contains two sub folders – *client* and *card*. The client folder contains one folder SmartCardTerminal and one ZIP file with the same name[1] The card folder contains the source c code – *ecc_diffie_hellman.c*, the header file – *multos.h*, generated hzx and alu files and script, which enables to generate such files – *build.bat*. Recommended software to open the C code is Eclipse IDE with SmartDeck platform and NetBeans to open and run the Java code.

[1]Because some environments support importing of Java project in ZIP format, for example NetBeans, the client folder contains both ziped and unziped forms.