DESIGN AND IMPLEMENTATION OF AN AUTHENTICATION PROTOCOL ASSISTING SECURE COMMUNICATIONS BETWEEN EMBEDDED SYSTEMS UTILIZING SMARTCARDS

by

SALOUSTROS MARIOS

A THESIS

Submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF INFORMATICS ENGINNERING

SCHOOL OF APPLIED TECHNOLOGY

TECHNOLOGICAL EDUCATIONAL INSTITUTE OF CRETE

February 2015

Approved by:

Supervisors
Ass. Prof. Harry Manifavas
Ass. Prof. Konstantinos Rantos

Abstract

In recent years the smart card technology has been grown, offering smart cards with more memory, faster processors and hardware accelerated cryptographic processes facilitating their spread as a mean of providing security in many fields.

At the same time the Java Card platform, which allows the execution of applications based on Java to be executed in smart cards has been established in the area as the most common platform for use in smart cards since it provides the object orientation and safety of Java and having now reached maturity, provides a remarkable in terms of security algorithms API.

That is the reason why this combination of smart card with the Java Card platform has been selected for secure communication between two embedded systems nodes by the European nShield program using symmetric encryption of data and providing confidentiality, integrity and authentication of the origin of the message.

As part of this work, are referenced and explained the concepts of smart card technology and that of Java Card platform, also is referred the topology and the way of communication between the nodes and finally an implementation of a Java Card applet is presented that enables secure communication between those nodes using symmetric encryption and the message authentication code algorithm.

Σύνοψη

Τα τελευταία χρόνια η τεχνολογία έξυπνων καρτών έχει αναπτυχθεί, προσφέροντας έξυπνες κάρτες με μεγαλύτερη μνήμη, ταχύτερους επεξεργαστές καθώς και υλικό που επιταχύνει τις κρυπτογραφικές διαδικασίες διευκολύνοντας την διάδοση τους σαν μέσα παροχής ασφάλειας σε πολλούς τομείς.

Παράλληλα η πλατφόρμα Java Card, η οποία επιτρέπει την εκτέλεση εφαρμογών που βασίζονται στην Java να εκτελούνται σε έξυπνες κάρτες έχει καθιερωθεί στον χώρο σαν η πιο διαδεδομένη πλατφόρμα για χρήση σε έξυπνες κάρτες καθώς συμπεριλαμβανει την αντικειμενοστρέφεια και την ασφάλεια της Java και έχοντας πλέον ωριμάσει, παρέχει ένα αξιόλογο από πλευράς αλγόριθμων ασφάλειας API.

Αυτός είναι ο λόγος για τον οποίο επιλέχθηκε ο συνδυασμός έξυπνης κάρτας με την πλατφόρμα Java Card για την ασφαλή επικοινωνία μεταξύ 2 κόμβων ενσωματωμένων συστημάτων από το ευρωπαϊκό πρόγραμμα nShield χρησιμοποιώντας συμμετρική κρυπτογράφηση των δεδομένων και παρέχοντας εμπιστευτικότητα, ακεραιότητα και αυθεντικοποίηση της προέλευσης του μηνύματος.

Στα πλαίσια της εργασίας αυτής, γίνεται αναφορά και επεξηγούνται οι έννοιες έξυπνη κάρτα καθώς και η πλατφόρμα Java Card, επίσης αναφέρεται η τοπολογία και ο τρόπος επικοινωνίας μεταξύ των κόμβων καθώς και επίσης παρουσιάζεται η υλοποίηση μιας εφαρμογής Java Card η οποία επιτρέπει την ασφαλή επικοινωνία των κόμβων αυτών χρησιμοποιώντας συμμετρική κρυπτογράφηση και αλγόριθμο ελέγχου γνησιότητας μηνύματος.

Acknowledgements

I would like to express my deep gratitude to Professor Harry Manifavas, my supervisor, for his patient guidance, enthusiastic encouragement and useful critiques on this work. I would also like to express my very great appreciation to Professor Kostas Rantos for his valuable and constructive suggestions during the planning and development of the experimental and theoretical part of the current thesis. His willingness to give his time so generously has been very much appreciated.

Last but not least, I would like to thank my wife, Maria, for her love, patience and support she has shown during the past two years it has taken me to finalize this thesis.

Contents

List of Tables

List of figures

Chapter 1: Smart Cards

1.1 Introduction

Smart card is in fact a small computer. Has the size and shape of a credit card but what makes it "smart" is that it incorporates a microprocessor and a memory that gives it the ability to process and store data. The Central Processing Unit of a smart card is usually an 8-bit microprocessor, although the latest developments in technology allow for use of 16-bit or even 32-bit processors. In order to communicate a computer with a smart card, the card must be placed in or near (depending on the type of card being used) to a reader (smart card reader) which is connected with a computer.

The data storage capacity of a smartcard is much greater than that of a credit card. While a credit card can store a maximum of 125 bytes (persistent storing), the corresponding possibility of a smart card extends between 1 and 64 Kbyte or even more.

Obviously, the ability to store large amounts of data is one of the advantages of using smart cards, but it is also important that the stored data can be protected from unauthorized access and tampering attempt. Because of this high level of security, smart cards are often used by users of cryptographic systems for secure storage of critical information, such as the private key in a Public Key Infrastructure (PKI) system.

The price of a smart card is relatively small and depends mainly on the amount of memory and the software used for its operation. This software runs on the card (on-board software) and depending on the type of card can range from a simple Operating System with encryption capabilities, access control, and file system to more complex environments that support the use of high level programming languages (such as C and Java) for adding new applications on the card even after been purchased by the end user.

A prerequisite for the operation of a smart card is the existence of an appropriate reader. However, most computer hardware manufacturing companies do not include such a device in the equipment of their base products, although their costs have fallen significantly in recent years. The addition of a smart card reader in a computer becomes even more difficult when one considers that, in the absence of a common standard for compatibility between card and device; each card requires a specific card reader to cooperate [1]. To address this problem, companies have built special devices called USB tokens. These devices are designed to work just like smart cards with the difference that there are not inserted into a specific reader, but to a USB port that is present in all modern computers.

In any case, the smart card technology is promising and grows rapidly. More and more people worldwide enjoy the benefits of using a smart card in their daily lives, while research has shown that for cards in general (including or not a chip) the global market ranged at 34.2 Billion Cards in 2013 reaching 16.1 billion dollars.

1.2 Smart cards' History

The mass spreading of plastic cards in the USA starts at early 1950s. The low price of plastic used made the production of robust, durable plastic cards possible which were much more suitable for daily use than that of a paper or cardboard cards previously in use, not able to withstand mechanical stresses and climatic effects.

The first completely made of plastic payment card for general use was issued by the Diners Club in 1950 [2].

This card was aiming an exclusive class of individuals, and thus also served as a status symbol, giving the ability to holder to pay with his or her 'good name' instead of cash. Initially, only the more select businesses accepted these cards, so this card was known as a 'travel and entertainment' card.

Visa and MasterCard entered into the field leading to a rapid spread of 'plastic money' in the form of credit cards. This started firstly in the USA, with Europe and the rest of the world following a few years later.

Today, credit cards allow travelers to make shopping without needing real money everywhere in the world. A card owner is never at a loss in means of payment, yet he or she avoids exposure to the risk of loss due to theft or other unpredictable hazards, particularly while traveling. Another advantage of using credit cards is that eliminates the need of exchanging currency when traveling abroad. These important advantages were the reason why the credit cards spread throughout the world. Today billions of cards are produced and issued annually.

At the beginning, the functions implementing by these cards were quite simple. They were mainly used as data storage device offering security against forgery and tampering. General data, such as the card issuer's name, was printed on the surface, while personal data, such as the cardholder's name and the card number, was embossed. In many cards a signature panel was present for the cardholder to sign his name. These first generation cards, was able to protect against forgery provided by these visual features such as security printing and the signature panel. It is logical to presume that the system's security depended on the experience of the employees working for the card-accepting organization. With the spreading of the cards use continually increasing, these rather simple functions and security provided were no longer enough, particularly since threats from organized criminals were growing fast.

The cards handling by merchants and banks created increasing costs that made a machine-readable card necessary, while at the same time, increasing losses for card issuers as a result of bad customer service and fraud that were growing. It became evident that the security against fraud and manipulation, as well as the basic functions of the card, had to be expanded and improved.

The first improvement for solving those issues was the use of a magnetic stripe on the back of the card, this stripe allowed digital data to be stored on the card in a format that a machine could read as a supplement to the visual information. This made it possible to minimize the use of paper receipts, which were essential before, although the customer's signature on a paper receipt was still required in traditional credit card applications as a form of personal identification. However, new technology that could eliminate entirely the use of paper receipts could also be invented. These made it possible to finally achieve the objective of replacing paper-based transactions by electronic data processing. This required changed in the method used for user identification, which previously used the user's signature. The method the industry came up with and led into widespread general use was a secret Personal Identification Number (PIN) which is compared with a number given from a terminal or a background system. For most people is well known this method when using bank cards in automated teller machines. Embossed cards including magnetic stripe and a PIN code are until today the most commonly used type of payment card.

However, cards offering magnetic stripe technology have a strong weakness, which is that if someone with access to a compatible magnetic card reader/writer may read, delete and rewrite the data stored on the stripe at will. It is thus not suitable for storing confidential data. Complementary techniques must be used to achieve data confidentiality and prevent data manipulation.

In the 1970s, there was a rapid progress in microelectronics industry that made it possible the integration of the non volatile memory and the processing logic on a single chip measuring a few square millimeters. Jurgen Dethloff and Helmut Grotrupp, two German inventors at 1968 had the idea of embedding an integrated circuit into an identification card and are contained in a patent application field. Following that, in 1970 a similar patent application by Kunitaka Arimura in Japan approved. However, real progress in smart cards' development began by Roland Moreno who registered his smart card patents in France in 1974. His patents made possible for the semiconductor industry to supply the integrated circuits needed at reasonable prices. However, there were still many technical problems that had to be solved before the first prototypes, some of them containing enough integrated circuit chips and were able to be transformed into reliable products that could be manufactured in large numbers with sufficient quality at a reasonable cost.

Germany and France were the critical contributors in smart card technology, so it is not surprising that these countries lead the development and marketing of smart cards.

An important step was made in 1984, when the French PTT (postal and telecommunication services authority) successfully conducted a field trial using telephone cards. It is then that smart cards proved the fact that can provide high reliability and manipulation protection. The most important is that this smart cards' success was not in a field that already existed traditional cards, but in a new application field. Introducing a new technology in a new application offers the great advantage that compatibility with existing systems is not needed, so the capabilities of the new technology can be fully exploited.

At the same period as PTT was conducting its field trial, in Germany a pilot project was conducted, using telephone cards based on several technologies. Magnetic stripe cards, optical storage cards and smart cards were used in these tests.

Smart cards succeeded in this pilot study. Offering a high degree of security against manipulation and reliability, smart card technology has shown that offers the greatest degree of flexibility for future applications. Although the less expensive but older EPROM technology was used in the French telephone card chips, Germany used the newer EEPROM technology chips for their telephone cards. EEPROM technology does not need an external programming voltage. In France alone by 1986, several million telephone smart cards were in use. This number in 1990 was almost 60 million, and in 1997 in worldwide scale, hundreds of millions.

Germany had almost the same progress, with a delay of about three years. Following the success of Germany and France this technology spread throughout the world market introducing the smart card technology in the field of telephone cards. Chips embedded telephone cards are currently used in more than 50 countries. However, the use of telephone cards in their inventor countries (France and Germany), as well as in highly industrialized countries, has declined dramatically in the last decade due to the widespread availability of inexpensive mobile telecommunication networks and the general use of mobile telephones.

Telephone cards use integrated circuits which are relatively small, simple and inexpensive memory chips applying specific security logic that allows the card to reduce its cash balance while protecting it against manipulation. Chips that offer a microprocessor are larger and more complex. These chips were first used in large numbers in telecommunication applications.

The German Post Office in 1988 introduced a processor card using EEPROM technology as an authorization card for the analog mobile telephone network called C-Netz. The trigger for making such a move was the increasing cases of fraud caused with the magnetic cards used up to that time. For technical reasons, the analog mobile telephone network was limited to a relatively small number of subscribers (around one million), so it was not a mass availability market for processor cards. However, this move gained a positive experience from using smart cards in the analog mobile telephone system and defined the introduction of smart cards in the digital GSM [3] network. This network began its service in 1991 firstly in Europe and spread over the entire world, with more than three billion subscribers.

In the bank field the progress was significantly slower, mainly due to the more strict security and higher complexity of bank field compared with telephone field. It is worth mentioning that the development of modern cryptography has been just as crucial for the spread of bank cards as developments in semiconductor technology.

The widespread use of electronic data processing in the 1960s, the advances in cryptography experienced a sort of quantum leap. Modern hardware and software offering high-performance made possible the implementation of complex and sophisticated security algorithms in single-chip processors, allowing previously unparalleled levels of security to be achieved. Furthermore, this new technology was available to everyone, As opposed to the past where cryptography was a covert science in the private reserve of the military and secret services.

With the help of the modern cryptographic algorithms, was possible to mathematically calculate the strength of the security mechanisms in electronic data processing systems. It is no longer necessary to rely on a highly subjective assessment of conventional techniques, whose security is based on the secrecy of the methods used.

The smart card has proven to be an ideal medium. Create a high level of security available to all, since it could be safely store secret keys and execute cryptographic algorithms. In addition, smart cards are small and easy for the operator. Can be transported and used everywhere by everyone in daily life. It was a natural to use these new security features in bank cards in order to address safety risks arising from the increasing use of cards with magnetic tape.

French banks were the first to introduce this exciting technology in 1984, after the completion of the pilot program with 6000 cards for the period 1982-1983. It took another 10 years before all the French bank cards embedding chip. In Germany, the first tests were conducted in 1984-85, using a multifunctional payment card with an integrated chip. However, the Zentrale Kreditausschuss, which is the steering committee of leading German banks failed to issue a tender for multifunctional Eurocheque cards with integrated chips until 1996. In 1997, all German savings associations and many banks had issued new smart cards. In 1998, multifunctional smart cards enabled POS, an electronic wallet and optional value-added services were issued throughout Austria. This made Austria the first country in the world that has a nationwide electronic purse system.

In 1997 all German banks were using smart cards for their transactions. In US, smart card technology was not adopted as fast as in Europe. To familiarize the world with this new technology, Visa issued at Olympic Games in Atlanta in 1996 over 1.5 million smart cards VISACash [4]. That same year, Visa and MasterCard was financing research to solve the problem of compatibility of the cards with programming environments. The result was the creation of JavaCard, supported by Visa and MultOS (Multi-application Operating System), sponsored by MasterCard.

An important milestone for the future of the global use of smart cards to make payments was the adoption of EMV, a product of the joint efforts of Europay, MasterCard and Visa. The first edition was published in 1994. It provides a detailed description of the operation of credit cards that integrate processors, and ensures global compatibility of smart cards from the three major credit card agencies. Hundreds of millions of EMV cards currently in use worldwide.

With a delay of about ten years compared to the classic contact smart cards, contactless technology of smart cards has been developed at the maturity of the market. With contactless cards, an electromagnetic field is used to power the card and exchange data with the terminal, without electrical contacts. The majority of currently issued EMV cards use this technology to enable fast, easy payment for small purchases.

In the 1990s, it was expected that electronic purses, which store the money on a card and can be used for offline payment, will prove to be another factor for the international spread of smart cards for payment transactions. The first such system, called Danmont, was launched in Denmark in 1992. There are currently more than twenty national systems used in Europe alone, many of which are based on the European standard EN 1546. The use of such systems is also increasing outside Europe. Payment via Internet offers a new and promising area of application for electronic wallets. However, a satisfactory solution to the problems involving the use of the Internet to make payments safely, but anonymously around the world, including small payments, has not yet been found. Smart cards could play a key role in such a solution.

The expected success of electronic purses failed to materialize so far. Most installed systems remain far below the initial highly optimistic expectations, which, among others, can be attributed to the fact that pay for online transactions have decreased dramatically, resulting in one of the main advantages of electronic systems - saving cost of offline transactions - has largely disappeared. Today, e-wallet function is often included as a supplementary application in multifunctional smart cards for payment transactions

Another potentially important application for smart cards are the personal security devices for electronic signatures, which are gradually implemented in several European countries after the creation of a legal basis for their use in 1999, when the European Parliament approved an EU directive on digital signatures.

Another application resulted in the issuance of smart cards in almost all citizens in various countries. These smart cards are used as health insurance cards issued to members and contribute to cost savings in the pricing of insurance organizations services. In most cases, the first cards issued were simple memory cards containing only personal data of the insured, necessary for identification, but the patient cards that are currently in common use contain complex security microcontrollers also enables recipes and patient records storing, and electronic signatures usage to enable secure access to centrally stored data via the Internet.

The smart cards flexibility, allows programs for new applications to be added to a card already in use, has opened up completely new application areas, extending beyond the boundaries of traditional card uses.

As already mentioned, the technology of contactless smart cards has reached a level of maturity that enables the possibility of economic mass production. For this reason, contactless smart cards used as electronic tickets for local public transport in many cities around the world. Additionally, this technology has come in the area of electronic passports. Although electronic passports do not have the same size or shape to a credit card which has been standardized as ID-1 card, under the cover have the same circuit with a contactless smart card, which consists of a security microcontroller connected to an antenna for contactless data exchange. Efforts are currently underway at European level to achieve the standardization of electronic contactless cards that will be issued to all citizens, which will have a form of ID 1 (the same as a credit card), for use as a personal identification card, among others.

Although the history of smart cards and their applications starts more than 25 years ago, a steady stream of promising new applications is still under development. The growing and almost ubiquitous networking of our world creates significant problems in terms of security, confidentiality, and anonymity of personal data. Smart cards and personal security devices, with the ability to encode and store data safely, can significantly contribute to solving these problems.

1.3 Smart Card Applications.

With the rapid development of Internet technology and e-commerce, smart cards are accepted in global market as prepaid cards and data storage cards. However, their application fields are not limited there. They can be used for controlling access to specific areas, such as universities and businesses, as identity authentication of the holder, as well as electronic payment. Smart cards are also used in the transport sector and in the health sector. Also with the help of smart card technology a lot of "sensitive" data may be stored, such as passwords so that user does not need to remember a great number of passwords. In this section we describe some recent applications of smart cards. These applications can be classified into six main categories: Electronic Payment, Security and Authentication, Transportation, Telecommunications, Customer support services (loyalty cards) and Health and Insurance Applications.

1.3.1 e-Payment.

Although the primary use of smart cards was telecommunications field in recent years is widely used in a different market segment, called electronic payments system. In future electronic money application is expected to replace conventional payment and allow transactions through global networks (e-commerce) and pay TV (pay-per-view television). There are three basic models of electronic payments using smart cards: (a) Credit cards, where the payment is made after the delivery of the service (pay after), (b) debit cards, where payment is made on delivery of the service (pay now) and (c) electronic wallet (e-Wallet), where the payment is made before the delivery of the service (payment in advance). The application developed most in recent years is that of the electronic wallet which mentioned bellow.

1.3.1.1 E-Wallet

With e-wallet electronic money are "loaded" in the card before making any payment. When a transaction is completed, the card balance is reduced by the amount of the payment, and at the same time the amount is transferred into the wallet of the second party of the transaction (usually the merchant).

Examples of uses are controlled parking lots, tolls, fare payment in public transport (metro, trains, buses), soft drinks from automatic vending machines located in public places, automatic payment photocopying in public libraries and consumer goods payment in all kinds of shop.

Money transactions using the electronic wallet do not usually require the use of a PIN code. This way, the transaction may be faster but creates a problem in case the card is lost. Usually the transactions amount that takes place in an e-wallet payment is small, so a potential loss is not catastrophic. The widespread use of electronic money transfer may reduce the cost of handling large amounts of money for banks and merchants. To achieve this, the banking institutions have established several e-money running programs such as the Proton by BankSys, the VisaCash by Visa International [5] and the Mondex by MasterCard [6].

1.3.2 Security and authentication

Because of the high level of security offered by smart cards, has been widely accepted by applications needed user's security and authentication. Some of these applications are mentioned bellow.

1.3.2.1 Digital signature and digital certificates

Digital signature is a technology designed to provide security of data transferred within a Public Key Infrastructure network and to authenticate the owner. The keys required to implement such security application (public and private), can either be created by the card (on board key generation) or outside of it (off card key generation). The smart card is used to store the user's private key but also for creating a digital signature. In the simplest case, the digital signature can be used for secure transmission of electronic mail and electronic payment such as a digital signature of a transfer.

Also, a smart card can be used to store digital certificates to control user access rights in computers and network systems. This security applies for user authentication, but also for creation of virtual private networks (VPN) for accessing corporate systems from public networks such as the Internet.

1.3.2.2 Access Control

The most well known tools providing access to protected places where important data and information are kept are keys and magnetic cards. Their main disadvantage is that they can be easily stolen and forged providing access to unauthorized persons. This disadvantage come to meet smart cards because, not only is difficult to forge, but can also store personal user information such as access rights for specific areas. The movement data of a smart card holder can be stored in it, proving, if necessary, his location for a given time. Access control cards for enterprises and universities can give users access to specific data, parts and equipment, depending on their job position. A pilot study for implementing smart card technology for access control to protect physical and intellectual property; and transactions at photocopying centre without physical money has been made by N. Raza. [7].

1.3.3 Transportation

Smart cards may help transportation sector for acting as tickets for public transport and for payment of tolls and parking spaces. Usually prepaid contactless cards are used; the money amount representing shall be reduced by the corresponding price of service each time the cards are passed by the reading machine. Their use in the transport sector has advantages such as easy and fast service without blocking traffic, especially in the case of tolls. In order to attract more customers, transport companies using smart cards; usually reward their users with special offers the more they use the card.

Many countries use or are preparing to use smart card technology for their transportation, in 2016 Athen's commuters will be able to use smart cards in order to move around the city using public transport [8].

1.3.4 Telecommunications

Telecommunications is one of the largest markets for electronic card applications. Over 70% of smart cards issued are prepaid phone cards and this market will continue to grow for at least the next three years. Smart cards have become an integral part of mobile telecommunications. The smart card is present in every cell phone called SIM (Subscriber Identity Module) and has great potential in developing applications and memory. Inside the card contains information about the subscriber and the mobile network. Such information is the subscriber number, cost information, dialed numbers and SMS messages. As the use of mobile phones increases, the use of SIM cards increases too. Today it is estimated that the total number of SIM cards globally produced, exceeds 25 billion units and the industry continues to produce over 4.5 billion SIM cards each year. [9].

1.3.5 Customer Service Applications (loyalty cards)

Retail businesses are able to use smart cards to serve more effectively their customers and keep them loyal. For example they can offer bonus to their customers with some points with each purchase and reward them by giving gifts to redeem these points when they reach a certain level of points.

The fact that the points stored in the integrated circuit of a card provides two key benefits:

1. There is no need for network between the branches to update a central database with the points of the customer.
2. The customer is immediately rewarded by reaching a point limit, giving him extra motivation for further purchase.

This way business keeps loyal their customers, while getting information about their consumption habits, valuable data for both marketing and sales strategy providing more effective customer service.

1.3.6 Health and insurance Applications

Medical smart cards offer a new perspective on health applications. Inside the card can be stored information about personal patient data, such as insurance company, medical history and information useful at an emergency. This information may include the patient's name, which person should be contacted in case of need and potential sensitivities and reactions to this medication. In this way, the information needed is fast and reliable available to patients and physicians, allowing the free movement of patients who can travel abroad by bringing with them their insurance and their medical file. Many hospitals in France and Germany have started to implement this smart card application.

Furthermore, smart cards in healthcare are used in patient identification and healthcare professionals (doctors, nurses etc.), Electronic signature applications for the integrity and authenticity of medical data, in data encryption applications to ensure confidentiality (health professional cards) and secure access applications in health networks.

1.3.7 Other Applications

Other applications of smart cards are their use in set-top boxes, Internet access provisioning, and driver's license (ideal for storing points depending on the violations of the driver and immediate removal of the driving license) etc.

Table 1-1 shows a summary classification of known and possible future applications of smart cards by sector and type of card.

	Prepaid Card	**Records and information storage card**	**Access Control & Security Card**	**Membership card**
Bank sector	• E-Wallet • Bank transfer • E-Payment • Insurance application		• Access to specific account • Security using Internet from home	• Credit Cards • Debit Cards
Telecommunications	• Prepaid phone card		• SIM cards	
Public sector	• Accounts management (pensions, allowances, etc)		• Passport • Identity • Driving license	
Transportation	• E-Tickets • Automatic toll payment		• Boarding pass	
Healthcare	• Insurance payments • Medical payments	• Store / retrieve medical history • Store donor information		• Healthcare cards
Other	• Hotel reservations • Staff salaries payment	• Information / history of staff • Academic information • Personal information storing	• Access in airport departure halls • Hotel room keys • Access in buildings • Access in networks • Car renting keys	• Customer service programs

		• Car renting files • Personal profile		

Table 1-1: Categorization of smart cards applications

1.3.8 Advantages of using smart cards

As seen in the previous section, smart cards have a very wide range of applications. The feasibility of implementing multiple applications with security provides a number of advantages relating not only to businesses but also to end users of a service. Some of the advantages of smart cards presented below:

- Can detect and respond to malicious operation (tamper-proof) by embedding tamper-resistant microprocessors to prevent an attacker from retrieving or modifying the information, all integrated circuitry available in a smart card are epoxied (or glued) together such that there is no easy way to take the card apart. The only feasible way to communicate with the microprocessor is through its electronic interface and can be accessed only by the embedded software, which should contain the appropriate security measures [10].
- As smart cards are composed entirely by integrated circuit relating the smart part of the card, means that integrate the rapid developments in semiconductor technology, enabling the continuous improvement of their characteristics and as semiconductor technology evolves, smart cards evolves too.
- Ability of computing and calculating makes smart card a small type of computer capable of executing code, mathematic operations and cryptographic algorithms rendering it a valuable tool in a large scope of applications.
- Process data and store information since it contains a processor able to process data received RAM memory for temporary storing processing data and an EEPROM memory capable for long term storing.
- Able to compare and manage complex information with the help of its powerful for its size processor and the tools provided by the compatible operating systems.
- Typically used for high security applications due to its hardware specifications providing tamper-resistance but also most of the time embedding a cryptographic coprocessor for faster cryptographic operations. Smart cards operating systems are also designed with security in mind, offering plethora of security tools through their APIs.

- Able to choose which card data is accessible from different applications, a characteristic provided by the operating system of a smart card that provides firewall rules in the ability of an applet to access another applet's data stored. Firewall boundaries are created around every package of applets or even every single applet, so a rouge applet can only access the data of its own or of its package contained. This security feature, if needed can be skipped with the declaration of a variable as public.
- Allow high security, and this is due to the cryptographic techniques used for coding and decoding the information transferred from the smart card into the reader, and vice versa.

1.4 Smart Cards global market trends

This section takes advantage of the latest research made in the field of Smart Cards by leader companies in this area [11][12] to present the current scenario as well as the future market potential for cards in general and for secure smart cards globally.

The smart card market is expected to show a continued growth in the coming years, this is mainly due to increased demand for safe and reliable payment transactions. Moreover, large companies of online payments such as Visa, MasterCard and Europay have become the driving force for technological developments related with smart cards. The decrease in prices of Subscriber Identification Module (SIM) cards and mobile services use costs will result in an increase in the number of mobile subscribers, and thus further contributing to the development of the smart card market. However, factors such as price pressures and technological challenges, together with the cost of the transition to compatible smart card readers affect the adoption of smart cards. By contrast, developments in technology, lower prices and the introduction of high-end SIM cards offer great potential for the development of smart cards in the market.

1.4.1 Smart Cards market segmentation

The market for smart cards has been segmented based on four major parameters:

- card type
- components
- applications
- geography

Smart Cards Market, by Type

- Contact Smart Card
- Contactless Smart Card
- Hybrid Smart Card

- Dual-interface Smart Cards

Smart Cards Market, by Components

- Memory Cards
- Microcontroller Cards

Smart Cards Market, by Applications

- High-End Applications
    - Government Sector
    - Healthcare Sector
    - Transportation Sector
    - Telecommunications Sector
    - Pay TV Sector
    - Financial Services, Retail and Loyalty Sector
- Low-End Applications

Geographically, the market for global smart cards has been segmented into four regions:

- North America
- Europe
- Asia Pacific
- Rest of the World

1.4.2 Smart Cards growth until present

Eurosmart estimates, that 7.2 billion Smart Secure Devices were shipped in 2013 [13], while over 7.7 billion units to be shipped in 2014, this is a growth by 7%, the worldwide continued demand for high-end technological solutions combining convenience and security is strong.

For cards in general (including or not a chip) the global market ranged at 34.2 Billion Cards in 2013 where at 2012 was at 33.6 Billion Cards, this implies to a 1.9% increase for all types of cards manufactured [14]. Figure 1-1 Shows for each part of the world the corresponding number of cards manufactured in millions of units.

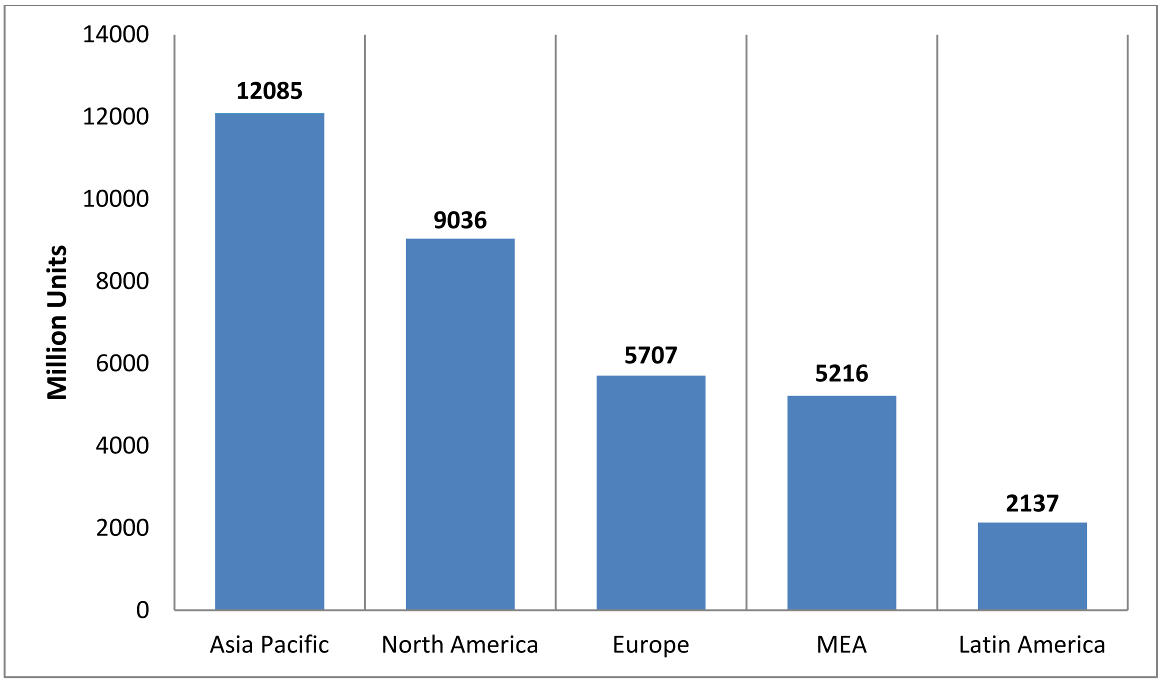


Figure 1-1: 2013 Cards Manufactured in Millions of Units

The market share for these cards manufactured benefited for the same years ranged at 16.1 Billion dollars for the year 2013 while at 2012 ranged at 15.1 Billion dollars, which translates into a percentage increase in market share of 6.5%. Figure 1-2 shows how this market share has been distributed between the regional markets.

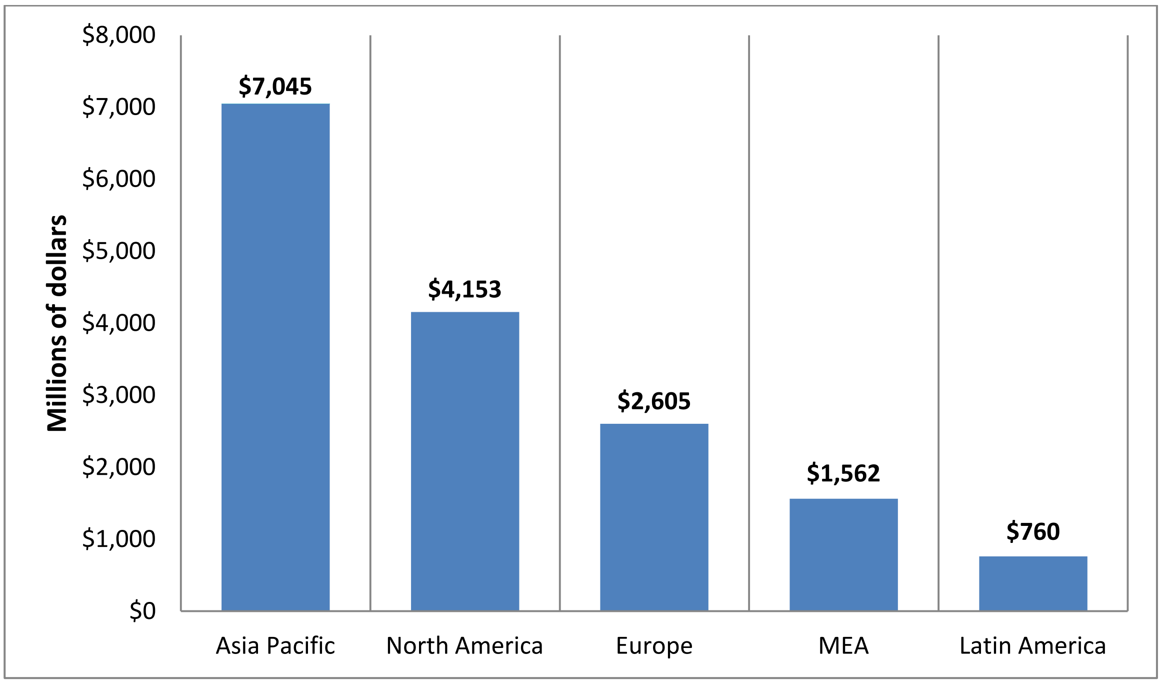


Figure 1-2: 2013 Cards Manufactured earnings in Millions of Dollars

The upward trend is noticeable in the advancement of contactless technologies such as Near Field Communication (NFC) and dual interface cards as contactless payment cards are expected to have very high demand in the coming years, thanks to a comfortable and secure user experience that provide. A past research [15] was indicating that every third payment card sent in 2013 will be a dual interface card. Combining the contact and contactless technology these cards offer multiple uses, for example, may be used both for the payment in a contactless terminal and for ordinary withdraws at ATMs.

An increasing growing part of electronic identification cards currently include contactless operation, an example is the electronic driving licence or electronic passports providing easy and automated border gate control. It is estimated that during 2013, every second secure electronic document in the Government segment is contactless. Transit cards, like the London Oyster Card or similar cards used in public transport, are all contactless, as are many physical and logical access cards like company access badges. Therefore, past researches account this market of 930 million contactless cards were estimated to be shipped in 2013, an increase of 41% over 2012. And this is without counting the more than 250 million units of NFC secure elements to be shipped in 2013.

Regarding smart cards with security features, in Figure 1-3 we may observe the increase in units shipped over the years divided for each application field, while Table 1-2 shows the percentage increase estimation until the current year.

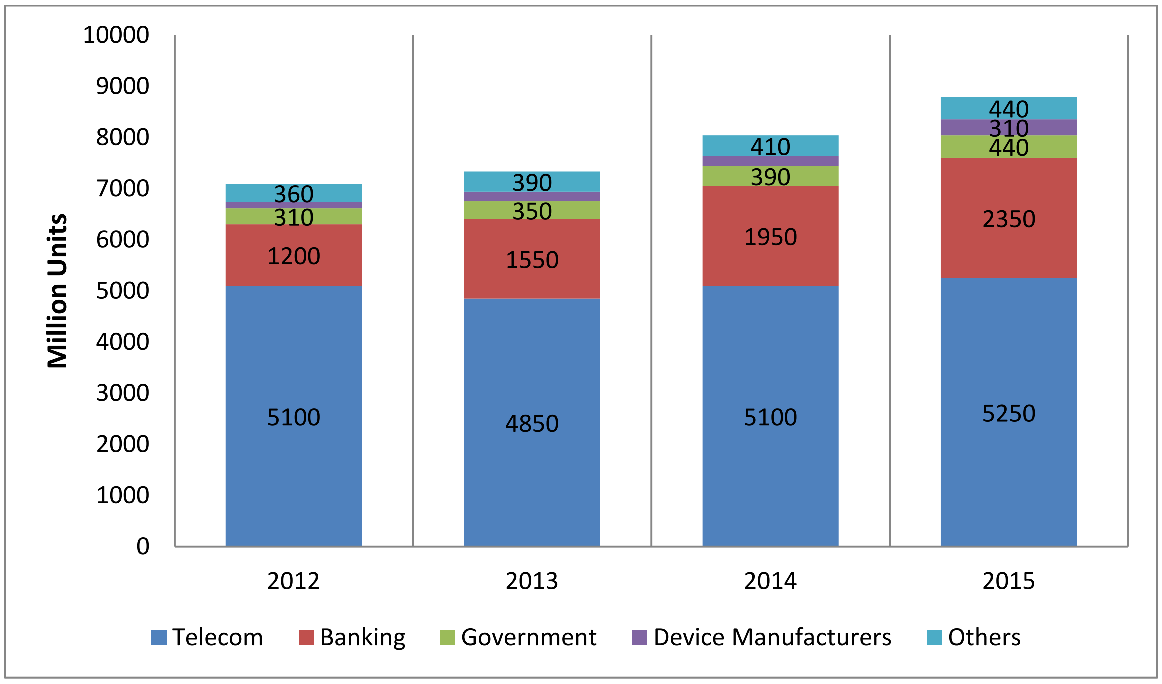


Figure 1-3: Secure Elements Shipments Estimates

Global SE shipments forecast	2013	2014f	2015f	2014 vs 2013 % growth	2015 vs 2014 % growth
Telecom	4 850	5 100	5 250	5%	3%
Banking	1 550	1 950	2 350	26%	21%
Government	350	390	440	11%	13%
Device manufacturers	190	190	310	0%	63%
Others	390	410	440	5%	7%
Total	7 330	8 040	8 790	10%	9%

Table 1-2: Global Smart Secure Elements Shipment

1.4.3 Secure Smart Cards market trends forecast

Both digitalization and globalisation of our world, of trade and services, is the hallmark of the 21st century. Smart devices are getting even smaller, user convenience is improving rapidly, and people can access digital services from even the remotest corners of the world. The vision of a world where everything and everyone is connected at all times, independent of time and location seems on the verge of finally becoming reality. The Word Economic Forum (WEF) calls this phenomenon "hyperconnectivity" [16] and estimates that by 2020 "50 billion networked devices will underpin our societies and economies". With technological innovations such as SIM cards, secure payment and electronic ID cards as well as associated solutions and services, the smart security industry is one of the major players at the heart of this development, both driving its evolution as well as safeguarding it against fraud and violation of privacy.

Eurosmart presented at the CARTES [17] Secure Connexions 2014, its forecast regarding smart cards with secure elements named *Vision 2020 paper* [18], an estimation of 12 billion secure elements is expected to be shipped in 2020 proving that the demand of the market for secure element based solutions that can provide both convenience and security is increased. Technological innovations regarding smart cards with security features like SIM cards, secure payment cards and electronic ID documents as well as associated solutions and services, the Smart Card Security Industry is proved to be one of the major players at the heart of this development, both driving its evolution as well as safeguarding it against fraud and violation of privacy.

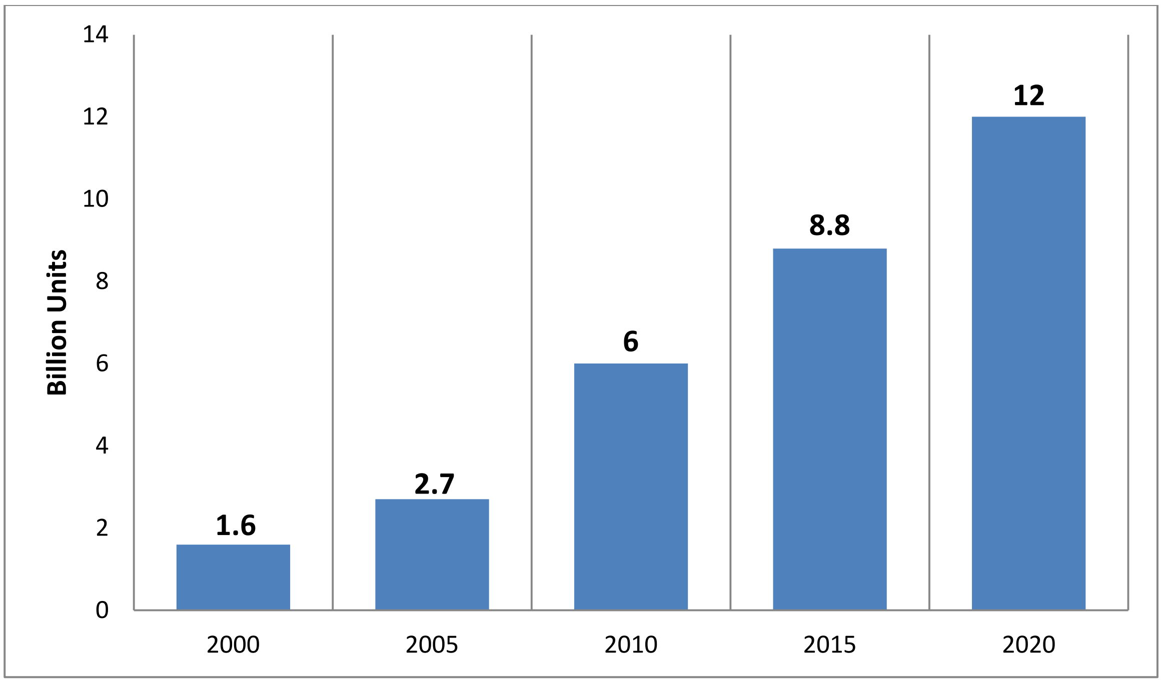


Figure 1-4: Secure Smart Cards Shipments Estimates until 2020

Figure 1-4 shows the estimated Secure Smart Cards shipment until 2020 according to Eurosmart.

Chapter 2: Smart Cards' characteristics

2.1 Modeling smartcards

A smart card is just one piece of a larger complex system. This means that the communication environment between the card and the rest of the system should be precisely defined which is achieved with the definition of national and international standards. These standards are essentially a set of rules that are independent of the applications implemented on the cards, thus ensuring interoperability. Interoperability should be guaranteed at 4 different levels [19]:

1. At the card itself
2. At the card reader
3. At the network
4. At cards developer system.

The following are organizations supporting the smart card technology and the standards they have proposed.

2.1.1 ISO- International Standard Organization

ISO 7816 is an international standard for smart cards with electrical contacts on them, and cards that communicate with terminals and readers without contacts, but with radio frequency technology (RF / Contactless cards). The first three parts of the standard ISO 7816 focus on the physical characteristics of the card and the type of integrated circuit used. The other parts identify all the mechanisms and properties of applications and operating systems for smart cards.

2.1.1.1 ISO 7816-1: Physical characteristics.

The first part of ISO 7816 specifies the physical dimensions of a smart card with contact and its resistance to static electricity, electromagnetic radiation and mechanical stress. It also describes the physical location of the magnetic stripe and the chip where a card combines both technologies.

2.1.1.2 ISO 7816-2: Dimensions and contacts position.

ISO 7816-2 defines the position, size and electrical characteristics of metal contacts of a smart card. It also describes the method to be used to measure the distances of the contacts in the card.

2.1.1.3 ISO 7816-3: Electronic Signals and Communication Protocol.

This is the most important standard regarding the electrical parameters of the microprocessor of a smart card. Standardize all the basic electrical characteristics as the supply voltage (3V-5V), stopping the clock and reset signal. A large part of the standard refers to data transmission mode in the physical layer and provides two transport protocols (T = 0 and T = 1).

2.1.1.4 ISO 7816-4: Interactive commands for data exchange.

The fourth part of the standard ISO 7816 defines a set of commands for the card's CPU providing access, security and data transfer to and from the card. Also basic mechanisms described for using smart cards in industrial applications.

2.1.1.5 ISO 7816 -5: Numbering System and Registration Procedure for Application Identifiers

Specifies the numeric format used to uniquely identify the applications that run on a smart card. This scheme is called AID and consists of two parts. The first one is 5 bytes long, called RID (Registered Application Provider Identifier) and identifies the application's constructor. The second part can be from 0 to 11 bytes long, called PIX (Proprietary Identifier Extension) and specify the application.

2.1.2 FIPS (Federal Information Processing Standards)

Created by the IT department of the NIST (National Institute of Standards and Technology), FIPS standards have been designed to protect federal resources, including computer and telecommunication systems. The standards listed below apply smart card technology and references to digital signature standards, encryption and security requirements of cryptographic models.

2.1.2.1 FIPS 140 (1-3):

Safety requirements in FIPS 140 (1-3) refer to fields related to security design and implementation of a cryptographic model. Specifically referring to physical security, operating environments, key management, electromagnetic compatibility, design guarantee and reduce of attacks. The content of this model is very practical and refers to detailed technical applications such as criteria for the quality of random number generators.

2.1.3 EMV

Europay, MasterCard and Visa have founded the company EMV [20] and drew up a set of rules entitled "Integrated Circuit Card Specifications for Payment Systems". The content of these rules associates with ISO 7816 standard and creates a common technical platform for cards and systems used in financial transactions

2.1.4 PC/SC

This is a Microsoft's proposal in the field of standardization for cards and card readers and applies only for cards with integrated CPU. The standard PC / SC (Personal Computer / Smart Card) runs on computers with Windows operating system. Allows cards to run any application, regardless of the programming language since it supports the most widely used languages such as C, C ++ and Java. The only prerequisite is a suitable driver for the terminal and the card used to be PC / SC compliant.

2.1.5 OCF

In 1997 more than 10 companies active in the field of smart cards and computers founded the Open Card Initiative. Their purpose was to create an interface between the smart card and the computer, independent of the operating system (Windows, Linux). The result was the Open Card Framework (OCF) [21], an interface based on Java that allows applications "running" on a computer to gain access to smart card applications. The OCF has been standardized and is widely used in environments where Java is used.

2.2 Smart cards types and technical characteristics

In this section refers at the types and technical features of smart cards.

2.2.1 Smart card types

Smart cards can be classified according to how data is read from and written in them and the type and capabilities of the integrated circuit containing. Figure 2-1 shows a diagram of classification of smart cards according to the type of integrated circuit used and the data transmission method.

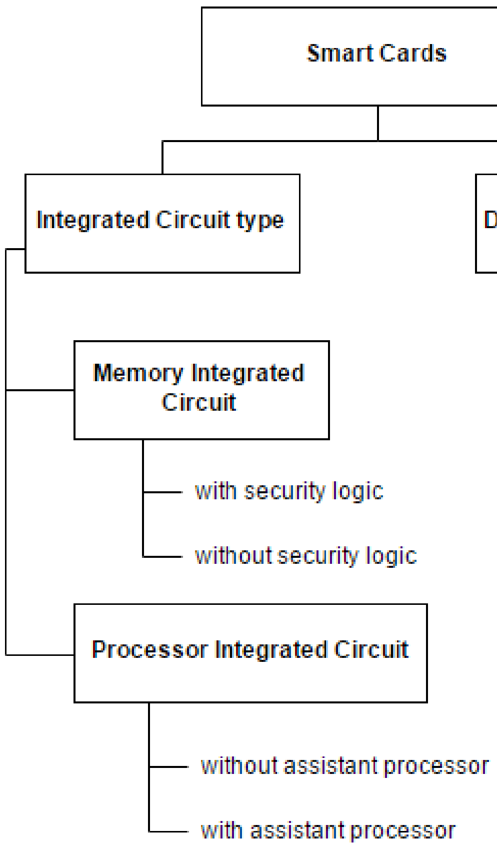


Figure 2-1: Smart cards Categorization depending on the type of integrated circuit and the data transmission method

2.2.1.1 Memory cards

The first smart cards produced were memory cards and this type of card is still today most used. A memory card does not integrate an embedded microprocessor and can be used only for data storage. The memory used is called EEPROM (Electrically Erasable Programmable Read-Only Memory) and because the lack of a microprocessor, memory access is achieved via a synchronous mechanism defined in the third part of the standard ISO 7816. In fact, the communication channel between the user and the card is always under the direct control of the reading device and the card's circuit responds immediately (synchronous) to reader low-level commands for accessing specific memory areas and reading from or writing to them [21]. The data transferred to and from the card through Input/output port. Memory cards lack a safety mechanism, resulting in data exposed to unauthorized access. However, in more recent memory cards, access to data is controlled by a security logic, which in the simplest case consists of a protection mechanism for memory (or areas) writes, or deletes.

Figure 2-2 shows an architectural block diagram of a memory card.

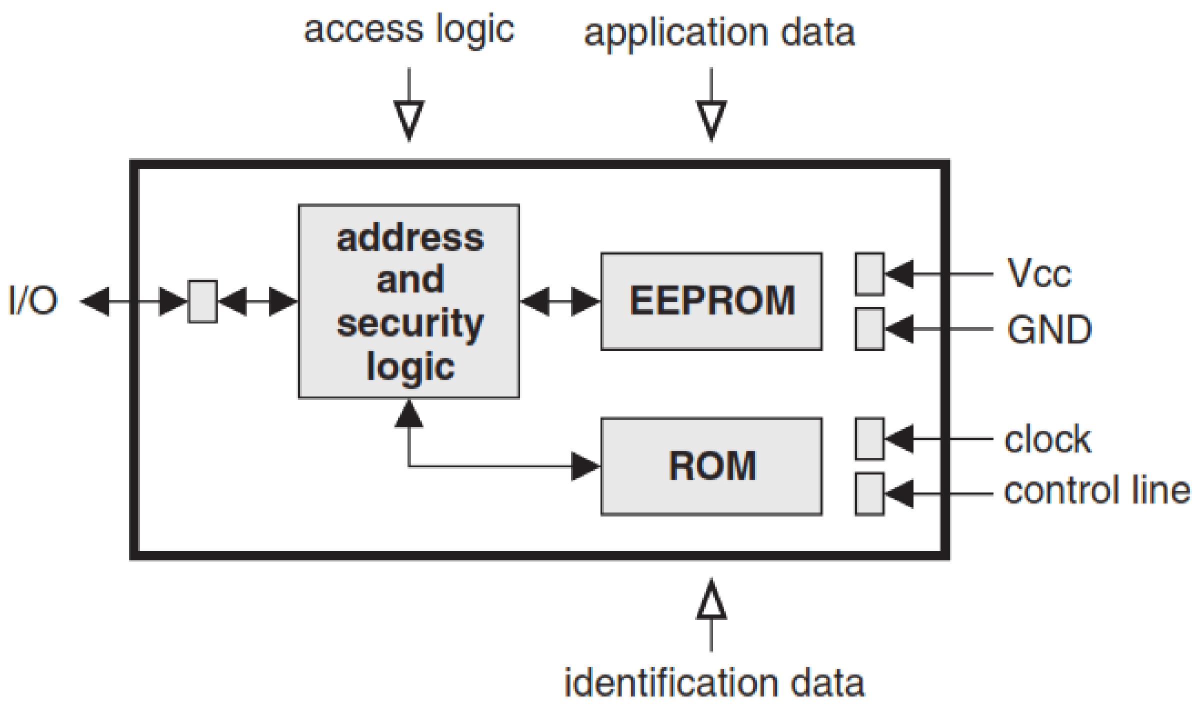


Figure 2-2: architecture of a contact-type memory card with safety mechanism

2.2.1.2 Microprocessor cards

The heart of integrated circuit of a microprocessor card consists of a microprocessor that communicates with the available types of memory ROM, EEPROM, RAM and an I/O port. Figure 2-3 shows the architecture of such card.

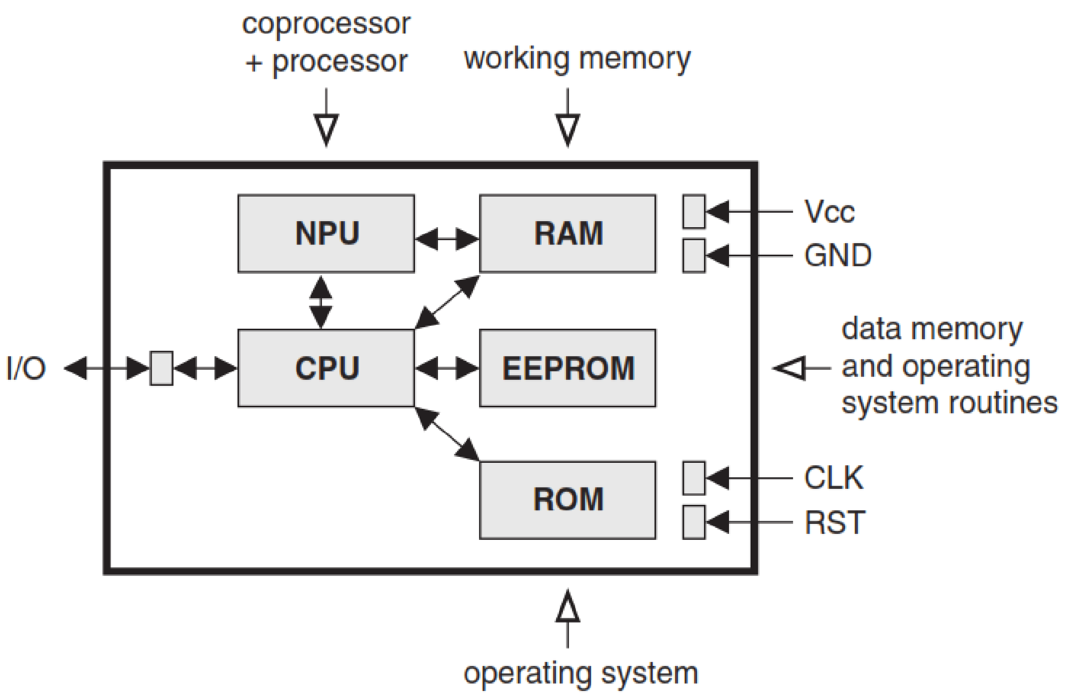


Figure 2-3: Typical architecture of a microprocessor card-type contact with an auxiliary processor (NPU)

ROM memory contains the operating system of the card, which executes the commands given from the terminal and returns the requested results. The content of this memory cannot be changed during the life of the card. The data and code programs are stored in EEPROM memory. The RAM is used as auxiliary memory of the microprocessor and data stored in it are lost every time the card is turned off. Through the I/O serial port data is transferred per bit to and from the card. In cases where microprocessor cards are used for large numerical computations, such as key pair creation in a PKI application, an auxiliary processor NPU (Numeric Processor Unit) is used for fast calculation of the required algorithms.

In the simplest case, microprocessor cards contain a program suitable for one and only application so that can only be used for this application. However modern functional smart card systems allow the implementation of many different applications. In this case the memory ROM contains only the essential parts of the operating system while the other more specialized parts, are loaded in EEPROM after manufacturing of the card.

One of the key features of microprocessor cards is security. If a user can not successfully verify its authenticity, he will not be able to process the data stored in it.

Because of the fact that in today's information society, the need for secure data processing is growing, the market share for microprocessor cards increasingly grows. It is now available specialized microprocessors with high processing and storage capabilities, to meet exactly these needs.

Figure 2-4 shows a digital card with microprocessor and electrical contacts.

Figure 2-5 shows, recent years production trends of memory cards and microcontroller cards.

Figure 2-4: Smart card with microprocessor and electrical contacts

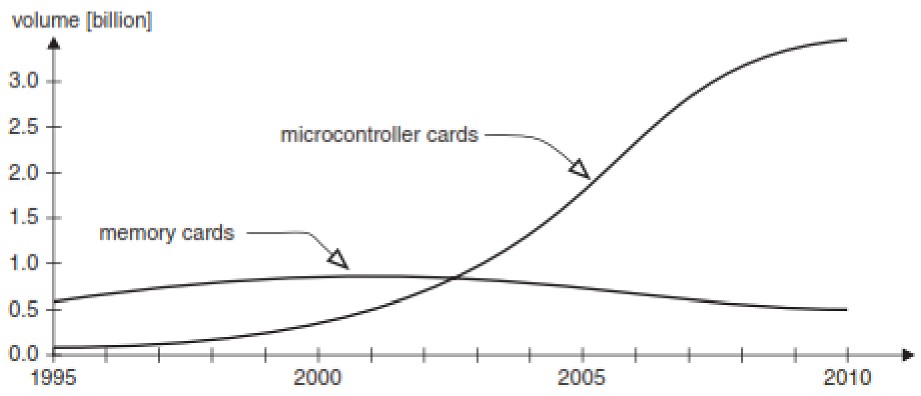


Figure 2-5: Worldwide production for memory and processor cards

2.2.1.3 Cards with electrical contacts

A smart card with contacts requires to be inserted into a reader and direct connection between the conductive portions of the card. These cards are the most common and cost-effective, but not suitable for all applications.

2.2.1.4 Contactless cards

Contactless cards communicate with card reader through radio frequency (RFID) [22]without the need to place the card into a reader. These cards do not have electrical contacts and communicate with a reader via an antenna which is embedded in the card and the reader. The energy required for card's chip to operate is transmitted through a microwave frequency from the reader to the card. Depending on the frequency electromagnetic waves are transmitted the distance between the card and reader varies from a few centimeters (for transmission at high frequencies) to 1 meter (for transmission at low frequencies). Contactless cards applications include the transport sector (tolls, public transport) where the service of large number of people required in the minimum possible time. Figure 2-6 shows the components of a smart card without metal contacts.

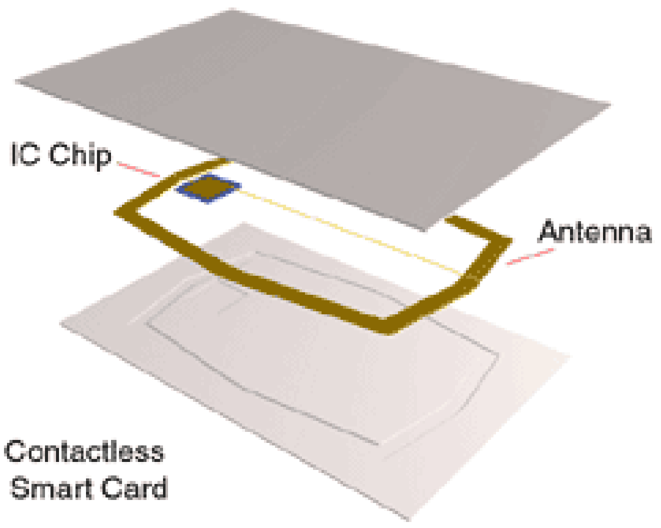


Figure 2-6: Components of a card without metal contacts

2.2.1.5 Hybrid and combi cards

From contact and contactless card technology emerged two other types of smart cards, hybrid cards and combi cards [23]. A hybrid card (Figure 2-7) contains two integrated circuits, one for communication via contact and one for wireless communication. These two circuits typically do not communicate between each other and are already used in many applications.

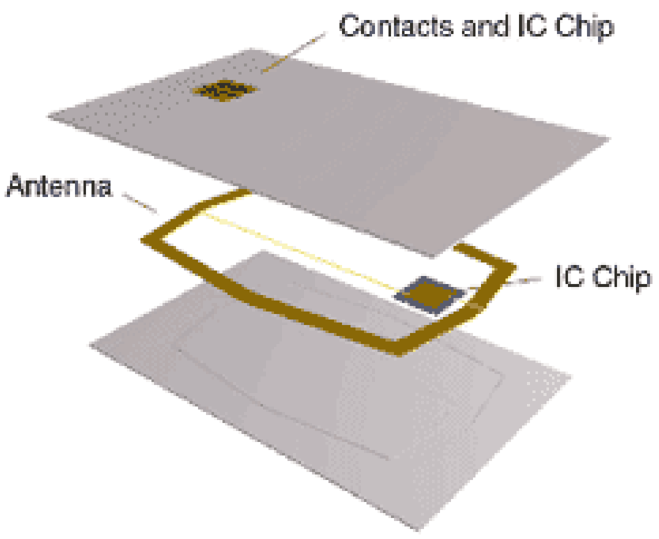


Figure 2-7: Components of a hybrid card

A combi card has an integrated circuit which can be used either for communication with contact or contactless communication, enabling its use in a plurality of different type applications. A combi card is cheaper than a hybrid, so banks and public transport are expected to be the first to adopt this kind of technology. Figure 2-8 shows the parts that make up a combined card.

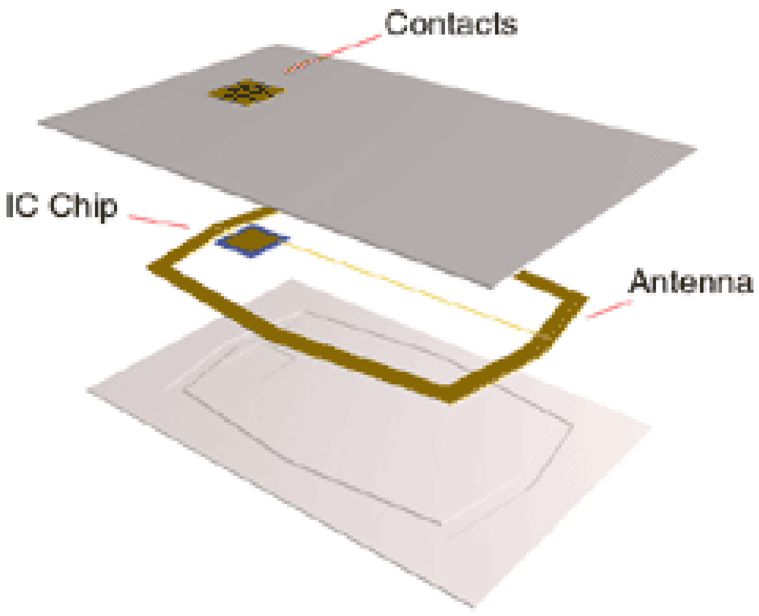


Figure 2-8: Components of a combi card

2.2.1.6 Optical smart cards

Optical smart card technology is similar to that of compact discs (CD-ROM). An optical smart card contains an optical recording medium between two transparent protective layers. The information is stored digitally in the form of 0’s and 1’s and data written in memory of card cannot be deleted. Despite their large storage capability, the usage of optical smart card is very limited because of the high cost of the read and writes equipment.

2.2.2 Smart cards technical characteristics

2.2.2.1 Dimension

Different cards applications require different card dimensions. To meet these different requirements have been standardized three different shapes with names ID-1, ID-000 and ID-00.

ID-1

An ID-1 card has 85.6mm length, 54mm width and is 0.76mm thick with divergence of 0.08mm and angular radius of 3.18mm with divergence 0.30mm.

ID-000

An ID-000 card has the shape of cards used in mobile phones. Its dimensions are 25mm length, 15mm width, 0.76mm thick with divergence of 0.08mm and angular radius of 3mm with divergence 0,03mm.

ID-00

An ID-00 card is 66mm long with 33mm width and it is 0.76mm thick with 0.08mm divergence and 3.18mm angular radius with 0.30mm divergence.

Figure 2-9 shows the relative sizes of each card according to its type.

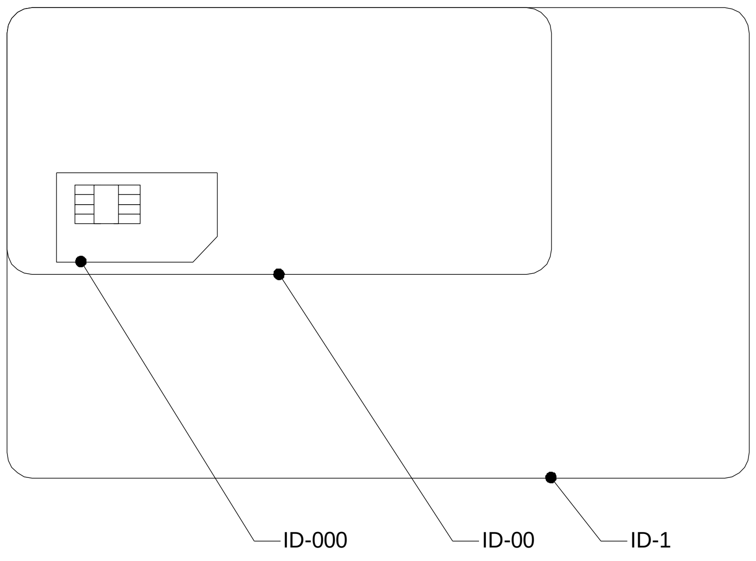


Figure 2-9: Example sizes for ID-1, ID-00 and ID-000 standards

2.2.2.2 Types of electric contacts

The electrical contacts of a smart card are the metal part located on the surface. Protect sensitive integrated circuit card from exposure to the outside environment, provides power and allow communication with the card reader. This metal outer casing may have 6 or 8 contacts depending on the size of the integrated circuit used. For circuits that have a size of up to 5mm^2 metal plate six contacts are used while for larger circuit eight. The maximum size of an integrated smart card circuit is approximately 32mm^2 [24]. The function of each contact is determined by the standard ISO 7816-2. Figure 2-10 shows the positions of the electrical contacts in correspondence with the operation of integrated circuits with 6 or 8 contacts. Next is an explanation of the function of each contact [25].

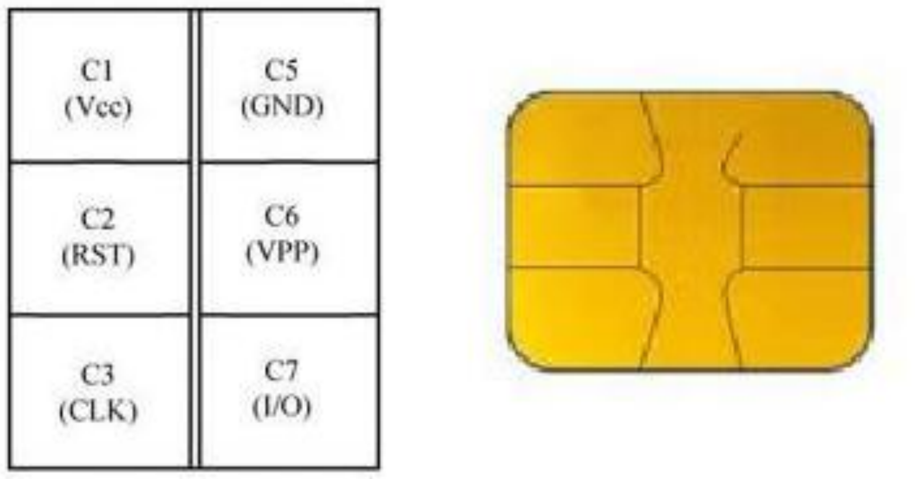


Figure 2-10: Electrical contacts numbering and corresponding function

- V_{cc} (C1): the supply voltage of the smart card. According to the ISO 7816-3 standard, the supply voltage is 5V with an acceptable deviation of 10%.
- RST (C2): input for card reset signal
- CLK (C3): clock input
- GND (C5): ground
- V_{pp} (C6): programming voltage (no longer used)
- I/O (C7): Input / Output Smart cards Operating Systems

Each smart card contains an operating system which provides the basic functions such as secure access to stored data, certification authentication and cryptography. The operating system of each card is "burned" into the ROM of the card at the time of construction and consists of a series of commands that are independent from the individual applications implemented in the card, but are used by them. The operating system of each card provides a means of communication between the application that is implemented and the computer terminal. Naturally, many smart card operating systems have been developed by manufacturers over the years. The creation of a single operating system that will run on all cards is the ultimate goal. But taking into account the different requirements of applications in security, computing power and memory, it is understood that it is very difficult to develop a single smart card operating system, at least for the foreseeable future.

2.2.3 Historical development of smart card operating systems

Since the introduction of smart cards in the early 80s until today, the software of smart cards has not stopped evolving. The operating systems of smart cards can be categorized chronologically into four generations. For better understanding of the evolution of operating systems for smart cards is initially presented the parties involved in the production process of a smart card (Figure 2-11). Each of these parties has an important role in the evolution of operating systems for smart cards.

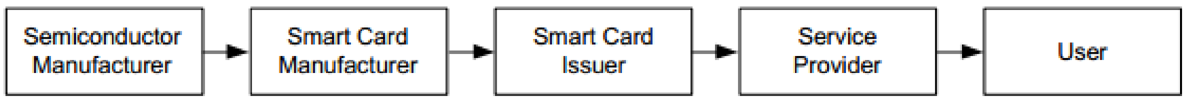


Figure 2-11: Parties involved in the production of a smart card

Semiconductor manufacturers are responsible for the design and production of integrated circuits. The smart card manufacturers are associated with the creation of the card's software and they must meet the requirements of smart card issuers. The latter are usually enterprises that should develop and manage applications based on smart cards. Service providers design and develop together with issuers services for smart cards and end users are those who benefit from these services.

First generation Operating Systems smart card software architecture is a monolith provided to semiconductor manufacturer by smart card manufacturers that meets client requirements and chip specifications. Smart card manufacturers (associated with software producers) could easily realize that, as chips benefited from market standardization, more pieces of code (mainly related to hardware management) could simply be copied. Leading to a very important software reuse, that could save time and greatly reduce the critical time-to-market; these observations marked the beginning of the second generation characterized by a three-part monolith:

Second generation operating systems comprise of three major parts:

- A set of rules concerning the hardware management modules
- A set of applications, common for most cards, such as the PIN code application.
- Application specific code card would implement and not covered by previous applications.

These three pieces were store in card's memory at the manufacturer of the integrated circuit level. The first two generations of operating systems no change could be made from the time the card was leaving the production site.

Third generation software emerged considering cost and not time. Smart card manufacturers realized the applications they designed could be almost entirely derived from limited platforms such as file systems or database systems. In this case, the question software producers ask is no more "which modules can be reused" but "which platform best fits the application requirements". This is the point where smart card software divided into parts and became more adaptable. Standard platforms are embedded, and may become more specialized by smart card manufacturers combining additional components called filters (which are functions that have not yet been incorporated into the standard platform, for example special PIN code management). The application is then stored by defining data structures and filling them with data. This final step can also take place at service provider's level but usually is left in the hands of smart card manufacturers.

Fourth generation is for open platforms. For this generation is not important how software is produced, but how the card is used. Modern open platforms allow code to be loaded on card even after it finds its owner. To achieve this goal, smart card manufacturers have introduced an application frame, which allows designing and loading applications wherever and whenever. The fourth-generation platforms are usually based on a virtual machine for both applications portability (one application can be loaded in many different cards) and security [26]. Figure 2-12 shows the development of software architecture focusing on the places where the software is managed.

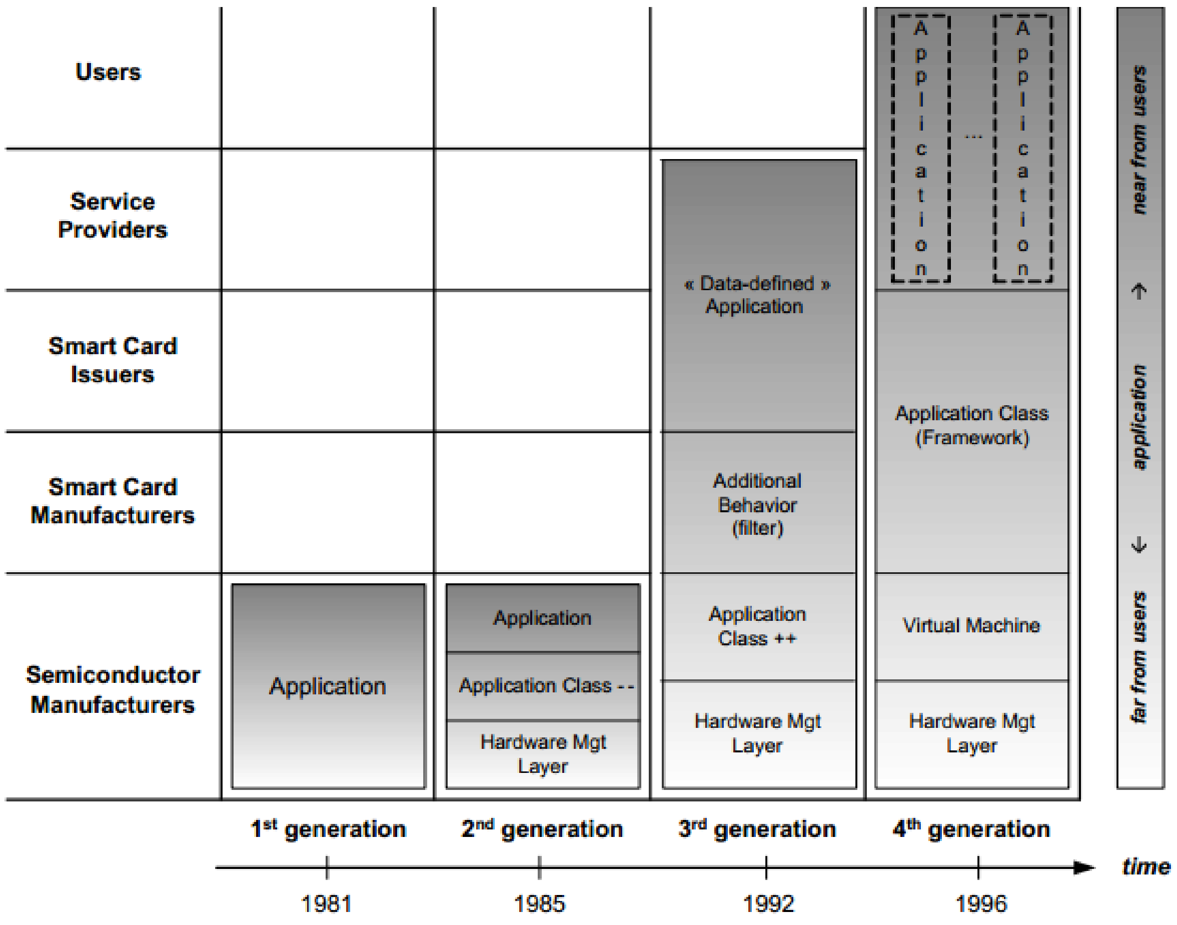


Figure 2-12: Smart card software architecture evolution

Observing smart card operating systems evolution, one may see that from monolithic model to open platforms model, software changes characterized by a gradual separation of operating system from applications implemented by placing the latter closer to the users. The most important fourth generation operating systems are JavaCard and MultOS while recent efforts are being made to develop a Linux operating system suitable for smart cards.

2.2.4 Fundamental knowledge for smart card operating systems

Smart card operating systems, unlike known computer operating systems do not provide a communication environment with user or accessing external media. Their functionality is completely different. Program execution security and protected access in card’s data have the highest priority. Because of memory limitations, smart card operating systems contain a very small piece of code between 3-250 Kbytes [21].

The operating system code is “burned” in a ROM type memory. This explains why it is not possible to make absolutely no change since the microprocessor programmed and constructed. A mistake’s correction is costly process and can take from 10 to 12 weeks [21].

These operating systems, therefore, must not only have zero mistakes but also to be stable, reliable and safe. Hidden trap doors often found in "large" operating systems, but regarding smart cards operating systems this should be avoided at all costs. The probability someone to bypass the operating system and gain, for example, unauthorized access to a file should be minimized as far as possible. Another aspect that should not be overlooked is the required processing capacity. The operating system should be able to perform in short time cryptographic operations, such as creating a pair of public-private key or a digital signature.

In summary, a smart card operating system should be able to:

- Transmit data to and from the smart card.
- Control the commands execution.
- Manage files.
- Manage and perform cryptographic algorithms.
- Manage and run programs.

2.2.5 Smart Card File System

All modern smart cards have a complete, hierarchical file management system specifically designed for them. The file system of smart cards is defined in the standard ISO 7816-4 and applies to EEPROM memory of the card. So for smart card are defined three types of files, the Master Files (MF), the lists of files (Dedicated Files-DF) and data files (Elementary Files-EF). Let us examine each file type individually.

2.2.5.1 MF Features

Each smart card file system has only one Master File (MF). The MF file is at the root of the hierarchical structure of smart card’s file system. An MF file can be compared with a directory which can contain other Directory Files (DF) or Elementary Files (EF).

2.2.5.2 DF Features

A Directory File can, like the MF file, be compared with a directory. Essentially a DF file forms a subdirectory under the root of the hierarchy which is an MF file. A DF file may contain from one to many EF files.

2.2.5.3 EF Features

The EF files are the last level of the hierarchy for a smart card file system. An EF file contains only data. There are two types of EF files: Internal EF, which are used by the applications on the card and Working EF, which are used for storing information coming from out of card applications.

Figure 2-13 shows the hierarchical structure of smart card file system.

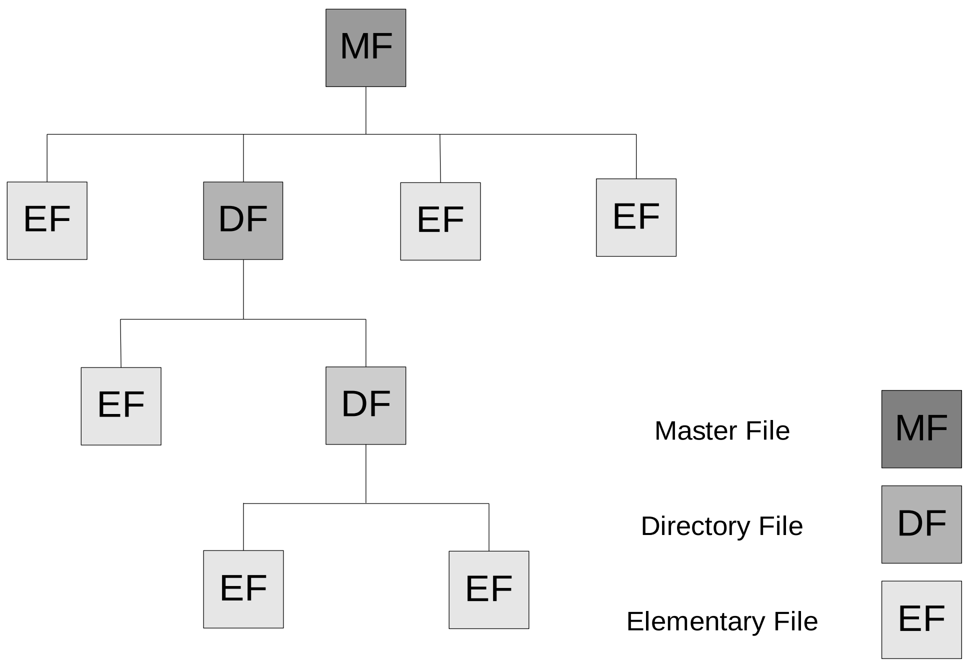


Figure 2-13: Smart card file system hierarchy

Elementary Files (EF) are divided into four types:

- *Transparent*: a transparent file can be considered as a sequence of bytes. When a read or write command is given to a transparent file it is necessary for an offset value to be provided, indicating the byte from which to begin the read or write operation and a value indicating the length of the information.
- *Linear fixed-length*: a linear fixed-length file is composed from fixed length. Each record must have a value of size 1 to 254Bytes.

- *Linear variable-length*: a linear variable-length file consists of variable length records, with size that ranges from 1 to 254Bytes. These files are characterized by an extremely short read and write and conserve memory space.
- *Cyclic*: a cyclic file may be considered as a ring of fixed length records. Characteristic of this structure is the existence of an index that always shows the record last modified. This record is numbered with number 1, the previous modified record with number 2 and the record modified before all records is numbered with number n, where n is the number of records.

Figure 2-14 shows the four types of data formats.

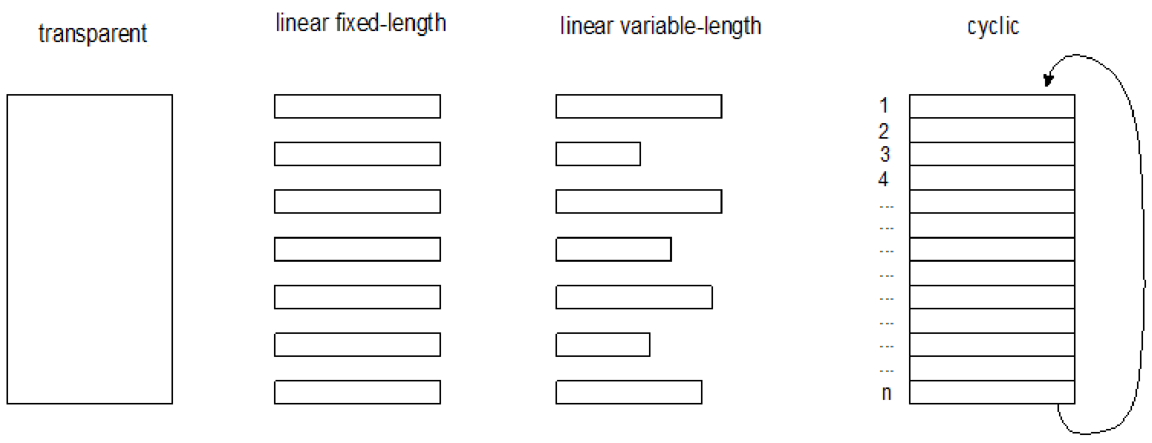


Figure 2-14: Smart card Data Formats

2.2.6 File access commands

ISO 7816-4 standard defines a set of commands for functions such as selecting, reading and writing a file. This set of commands is described below [27]:

Select File: this command gives essentially a pointer to the specified file in the card's file system. Once a file is selected with this command, any work (writing or reading) is done to the file indicated by this index.

Read Binary: this command is used for binary files whenever an application needs to read a particular piece of EF File. This command's parameters are an offset value corresponding to the first byte from which to start reading and a number of bytes (length) to be read and returned as result.

Write Binary: this command is used for binary files and is used by applications to write information on a particular piece of EF File. Depending on application requirements, this command may set the desired bytes at value 1, value 0 or write an array of new bytes in the transparent file.

Update Binary: this command manages binary files and may be used by an application for immediate deletion and store of information on a particular piece of EF File. Internal parameters of this command include an offset value from which to begin the deletion and a byte counter with the number of bytes to be written.

Erase Binary: this command applies to binary files and can be used by an application for deletion of an array of bytes (i.e. convert to 0 a number of bytes) to a particular piece of EF File. As in previous commands, parameters include an offset value corresponding to the first byte from which to start erasing and a counter byte with the number of bytes to be erased.

Read Record: this command applies to linear files used by an application to read and return one or more records of an EF file. Depending on the parameters, it reads and returns one record, part of the records or all records of a file.

Write Record: a command which may be used by an application for registration of a record in a file EF. This command handles linear files.

Append record: this command is used by an application for the addition of a record at the end of a linear file or for storing the first record in a circular file.

Update Record: this is the command used to directly delete a record from a linear file and writing a new one, the content of which is defined in the body of the command.

2.2.7 Open Platforms

With the increasing use of Java Card and MultOS, the term “Open Platforms” is increasingly used. The term “Open Platforms” refers to smart card operating systems that allow third party entities to load applications and programs on smart cards without involving the operating system developer. Most open platforms are publicly available and have been created by the cooperation of many companies, such as the Java Card Forum [28]. Then we will look at some of the most important smart card open platforms.

2.2.7.1 Java Card

Java Card [29] is Oracle’s proposal [30] for programming smart cards. Applications written in Java Card language are suitable for implementation on smart cards and other devices with limited memory and computing power. Java Card consists of three main components, the Java Card Virtual Machine (JCVM), Java Card Run-time Environment (JCRE) and the Java Card Applications Programming Interface (Java Card API) each of which will be discussed in detail in chapter Java Card technology. Applications written in Java Card language called Java Card Applets or Cardlets and can run on any smart card that supports Java Card.

Java Card technology is compatible with existing smart card technology. Complies with ISO 7816 standard and supports memory architecture, communication protocol and smart card application execution model. A smart card that supports Java Card communicates with the terminal through

APDU (Application Protocol Data Unit) messages in order to be compatible with existing technology terminals. Java Card technology will be studied in detail in chapter Java Card technology.

2.2.7.2 MultOS

MultOS (Multi-application Operating System) [31] is a smart card Operating System which supports the implementation of multiple applications and created to satisfy the requirements of electronic payment systems. Developed and supported by MasterCard and Mondex.

As in Java Card, the core of MultOS is a compiler that allows the creation of applications regardless of the card on which they run. Thus, MultOS applications can run on any card that supports this operating system. Because of the use of firewalls in its architecture, MultOS provide the ability to support multiple applications safely (Figure 2-15).

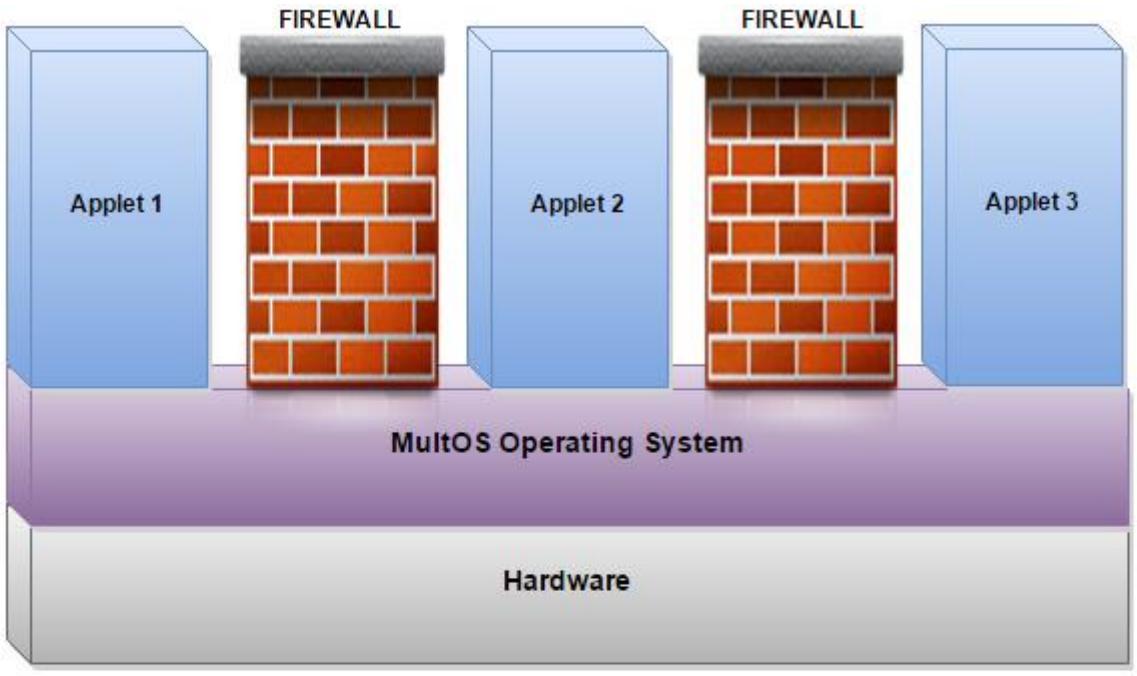


Figure 2-15: MultOS Architecture

Programming on MultOS compatible cards is done with C programming language and then translated into MultOS Executable Language (MEL) by a special translator. MEL language is executed by MultOS‘s virtual machine called Application Abstract Machine (AAM). MultOS also supports cryptographic algorithms such as DES and RSA. In order for an application to be loaded in a MultOS smart card must be digitally signed by an authorized certification authority. The biggest advantage of MultOS is the high level of security provided since different applications implemented in card are separated by firewalls.

2.2.7.3 Linux

From the moment it made its appearance in the mid-90s, the open operating system Linux has changed a lot the software industry. Although until recently aimed only PCs, with much greater computational capabilities than that of a smart card, recent attempts have been made for introducing it in low power microprocessors field. This requires a 32-bit microprocessors and several Kbytes of ROM and RAM memory. The current smart card technology cannot meet these requirements. But it is expected that over time, Linux will manage to ease these requirements. Meanwhile, the performance of microprocessors is increasing from generation to generation and it is very likely that Linux will be available for smart cards in the near future.

2.3 Transferring data to smart cards

Data is transferred from smart card to terminal reader and vice versa via the I/O contact, as we have seen, at C7 contact of the metal plate. Due to the limitation that there is only one transmission line, when one side transmits data, the other side acts as a receiver and waits its turn to transmit. This one-way form of communication called half-duplex. The communication with the card always starts from the terminal, which then waits for the response of the card. So the relationship between the terminal and the card is master-slave, where the terminal acts as a master and a card as a slave. Once the card is put in the card reader its contacts abuts the reader’s respective contacts and the communication begins. The first thing card do is to execute a command power-on-reset and then sends an Answer to Reset (ATR) signal to the terminal. The ATR is a sequence of bytes returned by the card in the terminal and indicates a successful power-up procedure. The maximum length of the ATR is 33 bytes but usually consist of just a few bytes. The terminal evaluates the ATR, which contains a number of parameters associated with the card and data transmission, and then may send the first command. The card receives the command, generates a reply, send it back to the terminal, and so on.

2.3.1 Data Transfer Protocols

There are many ways in which communication can be achieved with a smart card. There are also many methods for communication recovery if for some reason it stopped. The exact implementation of commands, responses and procedures followed if an error occurs define a transport protocol. There are a total of 15 transport protocols identified by the symbol 'T =' followed by a number based on the serial numbers of the Protocol. Table 1-1 summarizes a smart card’s transport protocols as defined in standard ISO 7816-3.

Protocols	Definition
T = 0	Asynchronous, half-duplex, communication per byte
T = 1	Asynchronous, half-duplex, communication per block
T = 2	Asynchronous, full-duplex, communication per byte
T = 3	Reserved for future full-duplex protocol
T = 4	Reserved as extension for T=0
T = 5…T = 13	Reserved for future use
T = 14	Reserved for protocols defined by the cards

	manufacturer, not ISO standardized
T = 15	Reserved for future use

Table 2-1: ISO 7816-3 Transport protocols

From protocols listed in the above table the most commonly used are T = 0 and T = 1 protocols. These protocols transfer data in the form of APDU messages, which will be presented in detail below. Besides the above asynchronous protocols, there is a set of very simple modern protocols, but without implementing error correction mechanisms and are used by memory cards in applications such as phone cards and health insurance cards.

2.3.1.1 T = 0 Transport Protocol

The protocol T=0 defined in the ISO 7816-3 standard. It is an asynchronous protocol, which means that there is no strict timing between a command sent from the reader to the card and the next command is sent from the card to the reader. When the card receives a command from the reader, performs the requested operation and sends a response. Then the reader is free to send the next command to the card when necessary.

For this protocol, the data are transmitted per byte (byte-oriented) through a communication channel between the card and the reader. The error inspection is made by a control bit (parity bit). If this control bit differs from the control bit of the data transmitted then an error has occurred and the message is retransmitted. The communication between card and reader is made via data structures called Application Protocol Data Units (APDUs). In T = 0 protocol there are two kinds of APDUs messages, the APDU command sent from the reader to the card and the APDU response sent from the card to the reader. Figure 2-16 shows the structure of an APDU command.

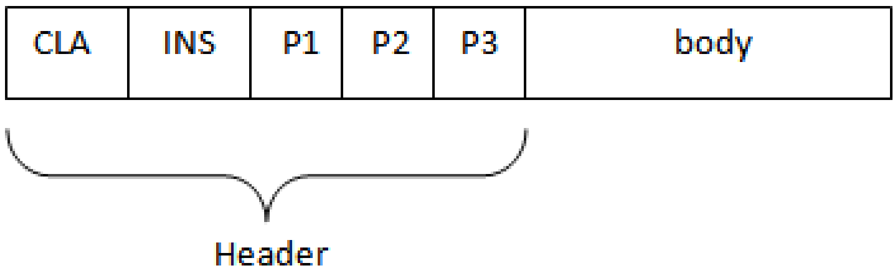


Figure 2-16: APDU command structure

The header of an APDU command consists of five fields, each having a length of 1Byte:

- CLA: defines a specific class of commands.
- INS: specifies the particular command to be executed from a class of commands.
- P1, P2: define the commands parameters.
- P3: specifies the number of bytes of data to be transferred.

For each APDU command sent from the reader, the card responds with a response APDU. The answer consists of three mandatory fields and one optional, all having length of 1Byte:

- ACK: indicates that the card has received the APDU command
- NULL: used for flow control in card's I/O channel. Informs the reader that the card processes the command and must wait before sending another command.
- SW1: card's reply for a particular APDU command
- SW2: card's reply for a particular APDU command (optional)

2.3.2 APDU message structure

APDU messages are used for exchanging data between the card and the terminal. The APDU is an internationally standardized data structure for the application layer. In smart cards this layer is located just above the transport layer in which data are transferred with APDU messages studied above. There are two types of APDU messages: the Command APDU (C-APDU), which are commands sent to the card and the response APDU(R-APDU), meaning the card's reply to the command received.

APDU commands are compatible with ISO 7816-4 standard and independent from the transport layer. Therefore, the content and form of an APDU should not be affected when using a different transport protocol.

2.3.2.1 APDU Command structure

An APDU command consists of a header and a body. The body may have variable length or may be absent if the data field is empty. Figure 2-17 shows an APDU command structure.

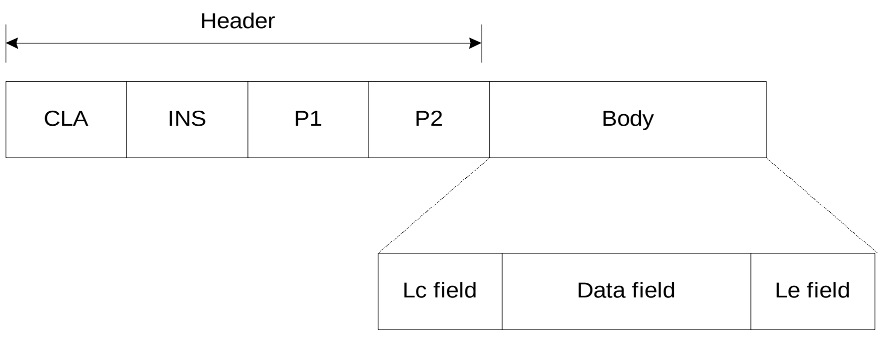


Figure 2-17: APDU Command structure

The header consists of four elements, which are, the class of commands (class of instruction-CLA) byte, the command byte (instruction code-INS) and two parameters bytes (Instruction Parameter-P1, P2).

- CLA: defines a class of commands. According to the standard ISO 7816-4 smart card commands, and hence the CLA byte, start with the byte 0x.
- INS: specifies the exact command. This byte's value should always be even.
- P1 and P2: further specify the command that sets the INS byte.

The rest of the command that follows the header is the body that is possible to be absent. Lc field contains the number of bytes in the data field of a particular command. The data field contains the data to be sent to the card and Le field contains the maximum number of bytes expected as a response data field. Both Lc and Le have 1 byte length.

2.3.2.2 Response APDU structure

A response APDU consists of an optional body and a mandatory tail (trailer), as shown in Figure 2-18.

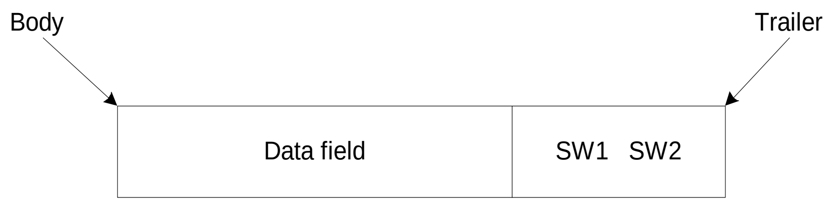


Figure 2-18: Response APDU structure

The body consists of the data field whose length is determined by the Le byte of a particular APDU command. Regardless of the value of Le, the data field of an APDU response may be zero if the card stops processing the command because of an error or incorrect configuration. The tail in a response APDU is mandatory. The two bytes, SW1 and SW2, report the response to the command. Depending on their content, indicate whether the command was executed successfully or if its processing stopped unexpectedly because of an error. Successful execution of the command is indicated by code '9000' (SW1 = 90, SW2 = 00). Any other code indicates unsuccessful processing of the command.

Chapter 3: Smart Cards' Security

3.1 Introduction

Smart Cards are considered a more secure option relative to other data storage devices, such as magnetic stripe cards and memory sticks. Their main advantage is that they provide secure storage of confidential data. A prerequisite for this functionality is the existence of a chip, specially designed to serve this purpose, together with appropriate cryptographic algorithms for protecting confidential data. Unfortunately, security does not only depend on just special microcontroller hardware and cryptographic algorithms implemented. A smart card application security and design principles used in the system development process are also of crucial. This chapter provides a synopsis of the essential principles, methods and strategies for producing secure smart cards and secure smart card applications.

The most important characteristic of a smart card is that it is able to provide a secure mechanism for storing data and programs. If the amount of effort needed to maliciously access protected data from a smart card were not so large, it would essentially be comparable with a memory stick with a different interface.

It is understandable that it is practically impossible to implement a perfect system, or even a smart card, to provide perfect security capable of defending against any attack under any scenario. If enough time and expense is devoted to an attack, every system can be breached or manipulated. However, every potential attacker either consciously or unconsciously performs a sort of cost/benefit analysis and the reward of a successful attack must be worth the time, money and effort necessary to achieve this objective. If the result is not worth the effort, nobody will invest much effort in breaking a system or a smart card.

3.2 Security in smart cards

One of the most important factors in maintaining the integrity and privacy of data stored in a smart card is to protect them from unauthorized access. It is then understandable that the implementation of applications is required, able to check and verify the identity of the person requesting access to the data in the card. The best known and most applicable techniques for authenticating a smart card user are those that request a PIN code (Personal Identification Number) and those controlling biometric characteristics.

A PIN code is usually a four-digit number, which is stored on the smart card. Each time a user tries to access data in the card, the card asks him to enter a PIN code and compares it with the four-digit number stored. If these numbers match then the user gets access to the data. In case of error, the card forbids access but request the code again. If at *n* fails to provide the correct PIN code the card blocks and prohibits access to the data. The reliability of the PIN code request method is based on the fact that the only person that knows the correct PIN code that "unlocks" the card is the owner of the card. If a third person discovers the PIN, then immediately the security of the card is collapsing, since the card is not able to know which physical entity requests access to the data. Also this security method has the disadvantage that an entity that may own many cards is likely to forget the password of a card. In general, however, the implementation of this technique requires no security complex cryptographic computations from the card, since the only requirement is to compare is the input and the stored PIN code.

But things are not so simple when the security is based on biometric characteristics [32]. By biometric characteristics we mean all those biological characteristics that characterize uniquely each person. Biological characteristics can be divided into two categories, the characteristics of physiology in which face, iris, retina of the eye and fingerprints belongs and behavior characteristics such as voice and handwritten signature. These features cannot be transferred to another person and so in fact is verified the identity of the person and not someone that has access to a password. By using of a particular technology is measured a biological characteristic of the person. As in PIN code's case so here the reference data, i.e. the data which will be used for the comparison, are stored in the card, but the comparison with the incoming data cannot be usually handled by the card. The use of biometrics for authentication of a person's identity finds nowadays increasingly more applications. The development of increasingly powerful microprocessors and the possibility of adding biometric data collection sensors on the cards may allow the application of smart cards in many new areas.

3.3 Classification of attacks and attackers

In this section we will make a classification of possible attacks and attackers in smart card systems. This classification helps to assess potential attacks, in order to take appropriate protection measures [21][33][34].

3.3.1 Attacks classification

Generally, attacks on smart cards can be analyzed into these different types: social, logical, physical, side channel and others [35]. However, in practice there may be combinational attacks.

3.3.1.1 Social level attacks

Social level attacks mainly concerns attacks against people working with smart cards. Such people may be microprocessor designers working in semiconductor companies, software designers or even mere card owners. These attacks can be treated with both technical and organizational measures. For example, for one managing to secretly see a PIN typing process can be avoided by using virtual screens left and right of the input keyboard (technical measures). Attacks on social level against smart card programmers are getting meaningless by making public the procedures used, and by assigning to a third party the evaluation of the program code used. In this case the safety depends solely on secret keys and so the knowledge of developers is useless for the attackers (organizational measures).

3.3.1.2 Logical attacks

Smart cards use a serial interface for transferring commands to them for processing. This interface supports a number of command options that are transmitted from a single communication channel for exchanging data with a smart card reader. Confidentiality of data and undesirable changes of the data is critical for the logical attacks.

- Hidden Commands: smart card operating systems have a series of commands that can be abused giving the attacker the ability to retrieve data from or modify data within the card. These commands can remain active in the installation phase or the execution of the previous application.
- Parameter Poisoning and Buffer Overflow: if these parameters of commands misinterpreted could lead to surprising results.
- File Access: smartcard file systems have detailed permissions on files and directories. The command access permissions determine the security procedures for accessing a file. Access permissions can be confusing to smartcard operating systems with complex interactions.
- Malicious Applets: smartcard security can be compromised with rouge applets.
- Communication Protocol: exchange of information between the smart card and the card reader is defined by a communication protocol that handles the data flow control and error recovery
- Crypto-Protocol, Design and Implementation: encryption protocols handle cryptographic operations required for the execution of transactions. These protocols should be designed carefully to avoid fallbacks with transactions.

3.3.1.3 Physical Attacks

Physical attacks refer to the examination and/or interference with the integrated circuitry of smartcards. The functions encapsulated on a card's integrated circuit can be reversed engineered, although, may require high-end lab equipment.

- Chemical Solvents, Etching and Staining Materials: smartcards can be de-layered and de-capsulated by etching materials. Etching dissolves the metal and silicon layers of the chip. Staining is an advanced etching technique that uses differences in etching speed to reveal subtle material differences that define the ones and zeroes in some ROM memories.
- Microscopes: optical and Scanning Electron Microscopes are possible to be used for optical analysis and reverse engineering. A functional chip capable to perform its electronic functions can be analyzed revealing active chip sections and potentially even running code or data values.
- Power analysis: allows the use of tine oscilloscope probe needles to be positioned on the circuit of a naked chip in order to monitor the differences in voltage and therefore data exchanged. All data exchange between the CPU and the memories can be monitored and it is possible to retrieve full running program code and program data including keys. Figure 3-1 shows this particular operation while in Figure 3-2 we may observe the results obtained using this technique.
- Focused Ion beam (FIB): shoots ions that can make changes within the circuitry. Blown fuses of test circuits can be reconnected, or hidden internal signals can be forwarded to external wires.

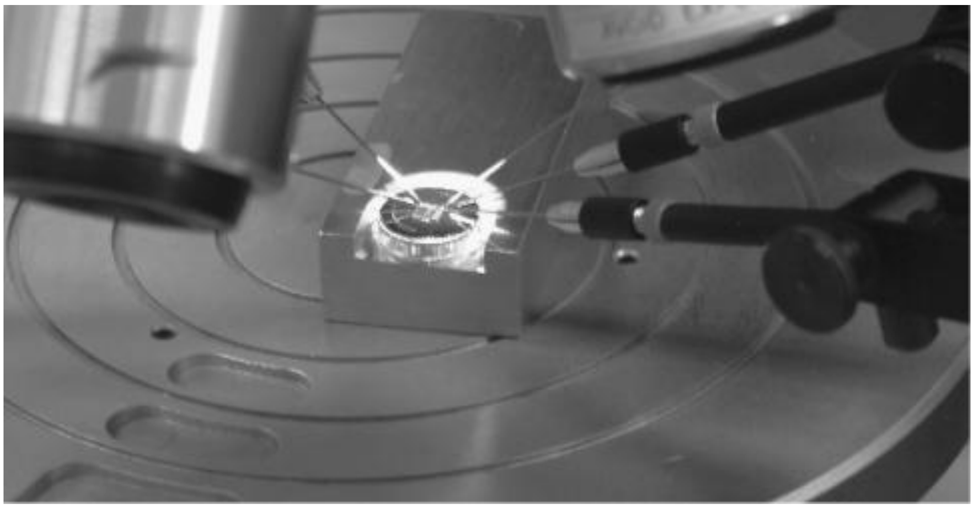


Figure 3-1: Using needle probes for examining an exposed microcontroller

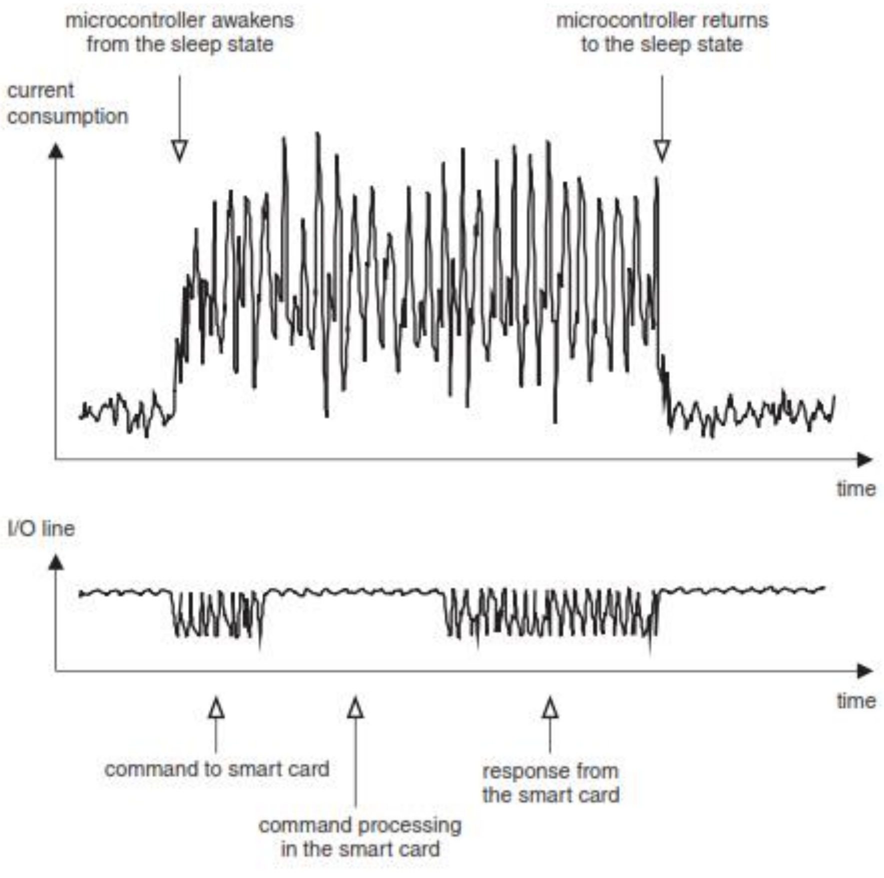

microcontroller awakens from the sleep state
microcontroller returns to the sleep state
current consumption
time
I/O line
time
command to smart card
command processing in the smart card
response from the smart card

Figure 3-2: Variations of current consumption during operation of a microcontroller

3.3.1.4 Side Channel Attacks

Smartcards are designed with integrated circuits that are composed of semiconductors sensitive to basic physical phenomena like electric power and radiation. Side Channel Attacks uses physical phenomena to analyze or manipulate the behavior of a smartcard chip. These attacks offer the great advantage that can be made without physically opening the card and therefore damaging it.

- Differential Power Analysis [36]: a statistical attack against a cryptographic algorithm that compares a hypothesis with a known result and is often capable of extracting an encryption key from a smart card or other computing device.
- Power glitching: Microprocessors are designed to operate at a constant voltage, where fluctuations or interruptions in power supply is likely to cause collapse of applications running or reset the circuit. A power failure will affect both the saved and the threshold values. Different internal capacities will influence values in different ways, could lead to misinterpretation of the actual value.

3.3.1.5 Other Attacks

Some other threats are identified as:

- Eavesdropping: easy ways to intercept and alter data being transmitted over the air.
- Interruption of operation: interrupting the communication between the reader and the card at any time without notice in order to produce abnormal card operation.
- Denial of service: cards can be emptied or destroyed remotely using suitable radiation.
- Covert transactions: fake transactions triggered by fraudulent merchants using fake readers.
- Communication links and dual modes: dual mode chip cards tend to share the underlying chip so that the only difference is the way the data is transmitted to the I/O buffer of the chip card.

3.3.2 Attackers classification

Typically, attackers have two main motivations. One is simply greed, while the other may be the desire of "prestige" and glory in a particular "field". But the consequences for the system administrator are different depending on the motive. An attacker seeking economic benefits, potentially risking to be the card publisher or attempting to blackmail the system administrator. In any case if the details of an attack made public, it is certain that the reputation of the smart card system will be harmed and if many card owners lose money because of an attack that maximize the damage dealt.

But it is possible for the system be damaged in other ways, often through scientific research, rather than using criminal acts. Here as an attacker may be considered any scientist / programmer discovered a “flaw” in the card security system and wants to make his achievement known. In that case the attacker has no fear of criminal prosecution, as he is considered as a defender of smart card security.

Finally, the conclusion is that there is not significant interest for who the attacker is, because if the attack is really dangerous, usually the economic and social costs are very large. In the worst case, the system must be closed, all cards should be blocked and deliver new ones that will not be affected by the attack. In a system with several million cards in use, such a process may take more than six months.

If we categorize attackers according to their nature, then six types of attackers arise, all equally dangerous to a smart card system, regardless of the different level of capabilities and access they may have. These types are shown in Figure 3-3 and are:

- *Hackers*: People with creative ideas, moderate knowledge about the system, without particularly sophisticated equipment and usually with limited financial means.
- *Insider:* They constitute a very special category, since theoretically have deep knowledge of the system and possible weaknesses, they can gain access to necessary equipment or software, but since they not anonymous as hackers, the source of their attacks can be recognized.
- *Criminals:* Without having special technical knowledge on the subject, are considered a threat because they usually show great zeal when it comes to gain personal benefits, mainly financially through an attack.
- *Academic institutions:* Possible source of attackers, who may be either students or teachers. Without necessary expertise on smart card's microprocessors, but with a wide range of useful general knowledge and information. They can easily use remarkable human resources (students) and adequate technological equipment that may exist in their laboratories.
- *Competitors:* Another special class of attackers, with high technical knowledge and high probability many of them own advanced equipment.
- *Organized crime:* Representing a completely different level of threat, since it has the financial resources to "buy" all the expertise and tools required for a successful attack by lawful or unlawful means.

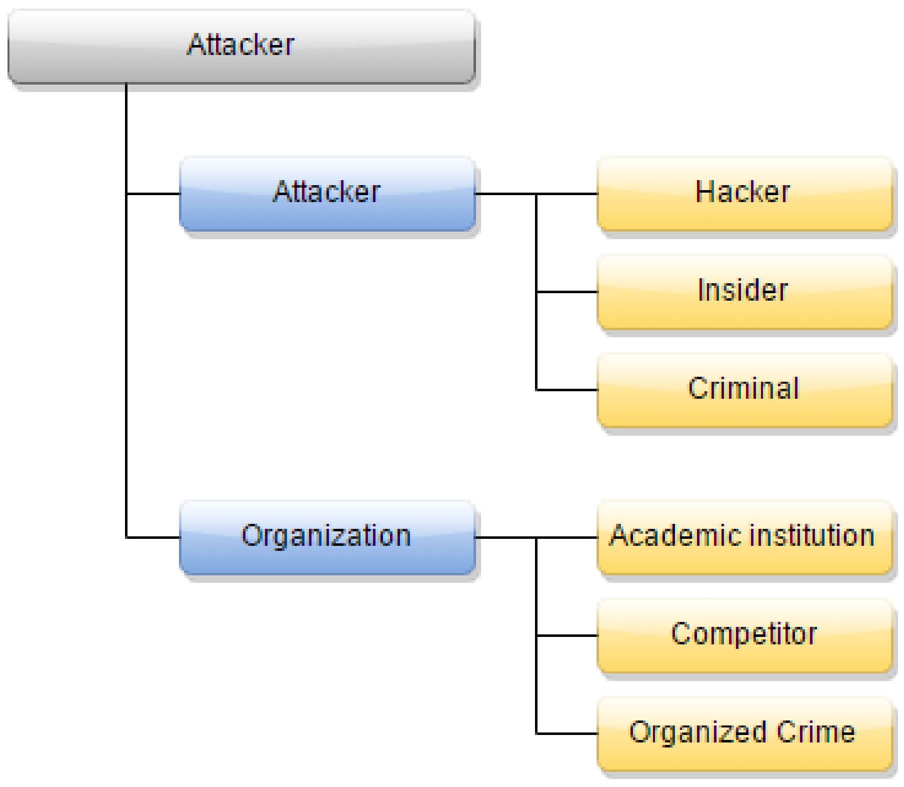


Figure 3-3: Classification of possible types of attackers

3.4 Smart Card’s attacks countermeasures

In this section there are presented some countermeasures for each type of attacks presented in the previous section that could potentially offer some security levels depending of the type of the attack is applied.

3.4.1 Countermeasures for Logical Attacks

This type of attack is related to the complexity of the smart card’s software. Considering the fact that with the size of the software code growing, the number of bugs created is increased. For addressing this issue, the following methods - countermeasures are provided:

- Structured Design: this method creates small functional pieces of software which are more easily understood and validated.
- Formal Verification: by using mathematical models is able to prove the correctness of the functions used.
- Testing: the implementation should be validated in an experimental environment before it’s released.
- Standardization of Interfaces and Applications: the attitude of re-using well-proved software reduces the possibility of errors.

- Convergence to the Java Card Operating System: an object-oriented language which by design was emphasizing on providing security is definitely safer than the older operating systems that do not provide separating applications.
- Popularity of Evaluation Labs: a growing number of card manufacturers and card issuers use evaluation labs to get a report or certificate concerning their implementation.

In short, the increasingly growing software complexity is naturally expected to lead in an error-prone implementation introducing new flaws. Careful design and validation makes it possible to reduce the number of defects of which will be created in the code but also increase the difficulty of exploiting these defects.

3.4.2 Countermeasures for Physical Attacks

Currently there are available in the market highly sophisticated chip designs accompanied with significant improvements regarding physical security.

- Feature Size: the increasingly smaller design of smart card components makes it increasingly difficult the task of using optical microscopes or placing needles of probe stations in order to analyze them.
- Multi-Layering: this manufacturing technique provides the capability of hiding sensitive data lines layers under other less critical layers for multiple layer chips, making the analyzing of the card's integrated circuits even harder.
- Protective Layer: where a protective layer containing an active current grid carrying protective signals (noise signals) is placed on top of all other layers preventing analysis of the real data underneath. A large number of seemingly non-correlated and frequently changing signals can avoid penetration of the protective layer.
- Sensors signals: can be used for measuring outside environment's variables including light, temperature, power supply and clock frequency combined with a logic of disabling the chip in case of harmful conditions are detected, thus, reducing the attacker's possibility to do live data analysis on a prepared chip.
- Bus-Scrambling: sensitive data traveling between blocks of the integrated circuit (like CPU and memories) through data buses, the latest can be scrambled using a sophisticated variable scrambling technique making harder to analyze.
- Glue Logic: this technique aims to confuse the attacker by gluing or mixing all functional blocks, making it harder for an attacker to identify the functional building blocks and analyzing the physical structure of the chip.

3.4.3 Countermeasures for Side Channel Attacks

For providing protection against smart card integrated circuit Power Analysis and in general Side Channel Analysis attacks, there are multiple techniques divided in three levels of defense which are mentioned bellow.

3.4.3.1 Hardware Countermeasures

Hardware countermeasures reduce the sensibility to side channel attacks. By increasing the noise in relation to signal make attacks more difficult to occur.

- Balance the circuits and reduce electromagnetic emissions to lower the power signal.
- It is possible to increase noise levels by performing consecutive random processes.
- Process interrupts and variable clock speeds are used with timing noise to prevent or intercept alignment of traces.

3.4.3.2 Software Countermeasures

Software countermeasures just like hardware's increase the noise achieving reduced leakage of useful information from the side channels.

- Processes are ordered randomly in order to reduce the genuine signal emissions.
- Other techniques used for adding timing noise is the use of random delays and/or using alternative paths that will intercept the alignment of traces, and lower the quality of the differential trace.
- Implement time constant key operations to eliminate time dependencies in key material and intermediate values avoiding basic power analysis attacks by visual inspection of traces.
- By adding random values to be subtracted later to hide intermediate values is used to prevent sensitive data emissions. These are carefully designed to indemnify the divergence caused by random data.

3.4.3.3 Application Level Countermeasures

Some general simple countermeasures that can block the preconditions for side channel analysis are:

- PIN verification mechanism preventing access and blocking the smart card after three unsuccessful login attempts can be a useful protection measure against differential analysis.
- Data visibility feeded to and returned by cryptographic operations should be minimal or restricted in order to avoid a differential power analysis attack.

3.4.3.4 Countermeasures for Power Glitching

The rigid use of sensors for voltage, frequency and temperature sensing is the most common technique against power glitching attacks. But sensor may affect the reliability of the smart card causing malfunctions in some cases (e.g. terminals location or climate).

Software and application countermeasures must be implemented to detect and recover from fault injection. Controlling the important program flow decisions and cryptographic results makes it possible to detect a fault injection. The validity of results should be done by computing the results twice and comparing both results.

3.4.3.5 Countermeasures for Other Attacks

In order to be handle sufficiently the threats, developers should be aware of the properties of Smart Cards. Public Key Cryptography and crypto co-processors needs to be optimized for the specific implementation. Encryption of the data being exchanged and mutual authentication is a necessity.

Chapter 4: Java Card technology

4.1 Introduction

Since 1997, Sun Microsystems introduced the Java Card programming language, opening new horizons for smart cards applications development. Today a lot of smart card technology promotion companies use Java Card as a development platform for applications that run on cards. The wide acceptance of Java Card by programmers and software developers is due to interoperability. The concept of interoperability is not unambiguous. Firstly lets developers write applications (Java Card applets) that can run on cards with different microprocessor technology and operating system, drastically reducing the cost of application development and the time to make them market available. Secondly, enables the coexistence of applications from different service providers, on the same card. Also software developers can choose which smart card technology best suits the needs of applications developed. Another important advantage of Java Card is that programmers do not need to know the communication protocols and memory management techniques used by the card operating system to develop an application. This implies that the development of smart cards applications becomes accessible to a greater number of developers.

In summary, the characteristics of Java Card that make it ideal application development platform in smart cards are the following:

- *Application platform independent:* Java Card technology is independent of the architecture of the card and the operating system running in it. It allows developers to write code in a high level language, which can be implemented anywhere, even in different smart card technologies (write once, run everywhere).
- *Versatile support*: Multiple applications, which may come from different developers, are supported by the same card. The design of small code fragments makes easier the secure implementation of different applications on a single card.
- *Installing applications after the issuance of the card:* installing applications in the card after its issuance, allow companies that have adopted smart cards into their system to respond dynamically to changes in their customers' needs.
- *Flexibility:* the object-oriented methodology of Java Card technology provides flexibility in smart card programming. A code written in Java Card can be easily extended and reused both in the same card and in other cards.
- *Compatibility with existing smart card standards*: *Java Card Applications Programming Interface (Java Card API)* is compatible with the official international standards
- *Easy to programming*: programmers do not have to deal with procedures relating to specific hardware, such as memory management, since these are controlled by the Java Card

environment. This reduces the production time while increasing the reliability of applications.

4.2 Java Card architecture

Smart cards are one of the smallest computing platforms today. The memory capabilities of a smart card may, for example, be of the order of 1K for memory RAM, 16K for EEPROM memory, and 24K for ROM memory. The biggest challenge for Java Card technology design was to match the Java software system on small-scale requirements of a smart card, while maintaining a sufficient memory space for the implementation of applications. The solution adopted was that Java Card supports only a subset of features of the Java model and a different implementation model of the Java virtual machine (Java Virtual Machine).

The Java Card virtual machine is divided into two parts: one that runs “on card” and one that runs “off card “. Many processes such as class loading, byte code verification and optimization are performed by the virtual machine that runs off card which usually features are not limited.

Smart cards differ from a personal computer in various ways. Apart from Java language, Java Card technology provides an execution environment (runtime environment) that supports the smart card memory system, communication, safety and the application execution model. The Java Card runtime environment is compliant with the international standard ISO 7816.

The most important feature of Java Card execution environment is that it provides a clear separation between the system and smart card applications. The virtual environment manages the system complexity and details of the card. Applications request services and system resources through a well-defined high-level interface.

Thus, Java Card technology provides a platform on which, applications written in Java language, can run on smart cards and other devices with limited memory capabilities.

Figure 4-1 shows the Java Card technology architecture.

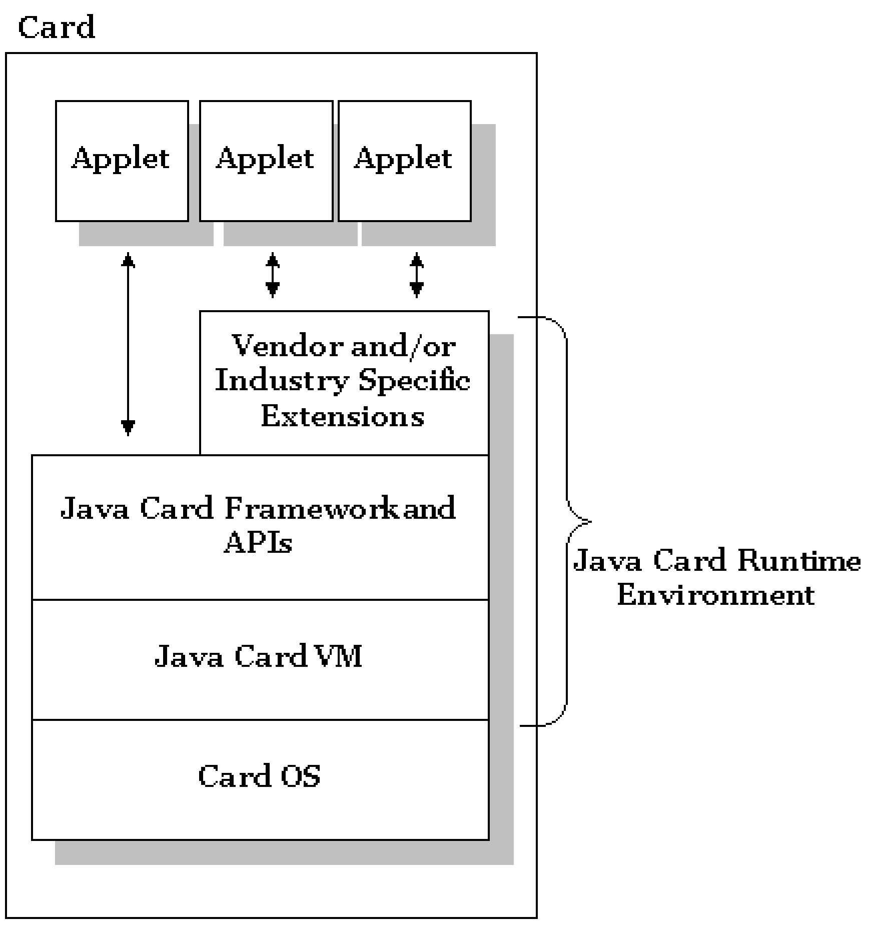

Card
Applet
Applet
Applet
Vendor and/or Industry Specific Extensions
Java Card Framework and APIs
Java Card VM
Card OS
Java Card Runtime Environment

Figure 4-1: Java Card architecture

4.2.1 Java Card Virtual Machine

A key difference between Java Card Virtual Machine (JCVM) and Java Virtual Machine (JVM) is that JCVM implemented in two separate parts, as shown in Figure 4-2

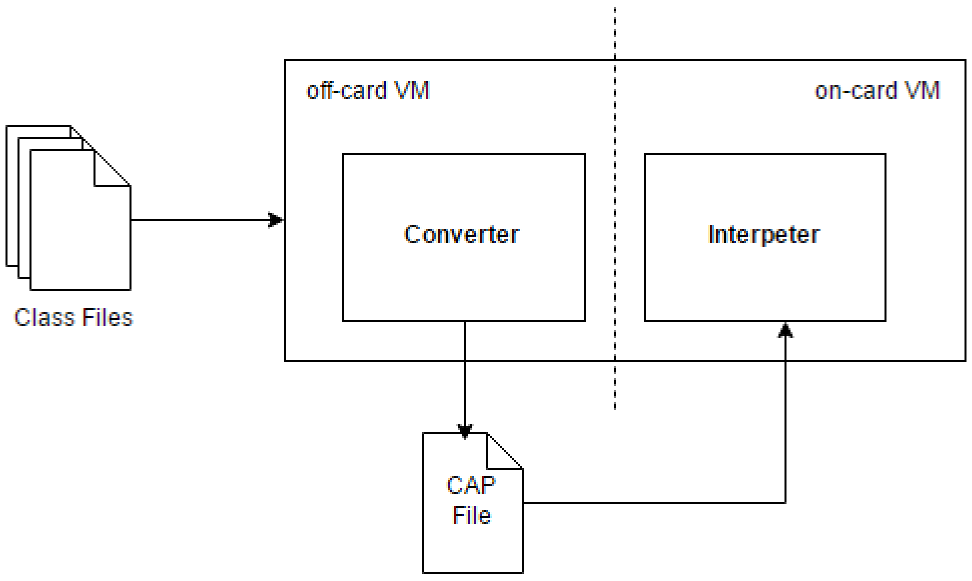


Figure 4-2: Java Card Virtual Machine

The on-card part includes a bytecode interpreter. Converter is executed on a terminal and constitutes the off-card piece of the virtual machine. Together, implementing all virtual machine's functions, load the classes and implement them. Converter loads and processes the files in classes composing a Java Card package and create a CAP (Converted Applet) file. The CAP file is loaded on the smart card and executed by the interpreter. Besides the CAP file, converter generates an export file containing the package API to be converted.

4.2.1.1 CAP files and export files

Java Card technology introduces two new file formats that enhance the development of an independent platform, dispersion and execution of Java Card software. A CAP file contains information about the classes of an application, executable binary code, application connection and verification information. CAP files use compact data structures and define binary code forms appropriately designed for the capabilities of the Java Card. CAP format is the format in which the software is loaded into a smart card that supports the Java Card platform and allows dynamic insertion of Java Card applications even after manufacturing of the card.

Export files are not loaded in smart cards and because of that are not directly used by the interpreter. They are created and processed by the converter for verification and connection purposes. They contain information for the methods of an entire class package, specifying the names of classes, methods and their fields. An export file also contains information for connecting class packages in the card.

Export files do not contain any executable code. Thus an export file can be shared freely by a programmer to application users without disclosing details of the internal implementation of the application.

4.2.1.2 Java Card converter

In Java Card, the converter processes all class files, which Java compiler has created from the source code of a Java package and converts the package to a CAP file.

During the conversion, the converter makes all those actions the Java Virtual Machine would make in a computer environment when loading classes, in particular:

- Verifies that classes are created correctly.
- Checks for violation of the Java subset that defined in Java Card.
- Initializes static variables.
- Converts symbolic references to classes, methods and fields in a more compact format, which is easily manageable by the card.
- Optimizes the binary code, by taking advantage information acquired during the loading of classes and connection.
- Reserves memory and creates virtual machine data structures for representing classes.

The converter accepts as input not only class files but one or more export files. Apart from creating CAP files, the converter generates an export file for the package that converts. Figure 4-3 shows how to convert a package. The converter loads all classes of a Java package. If the package imports classes from other packages, the converter loads the export files of those packages too. The output is a CAP file and an export file of the package converted.

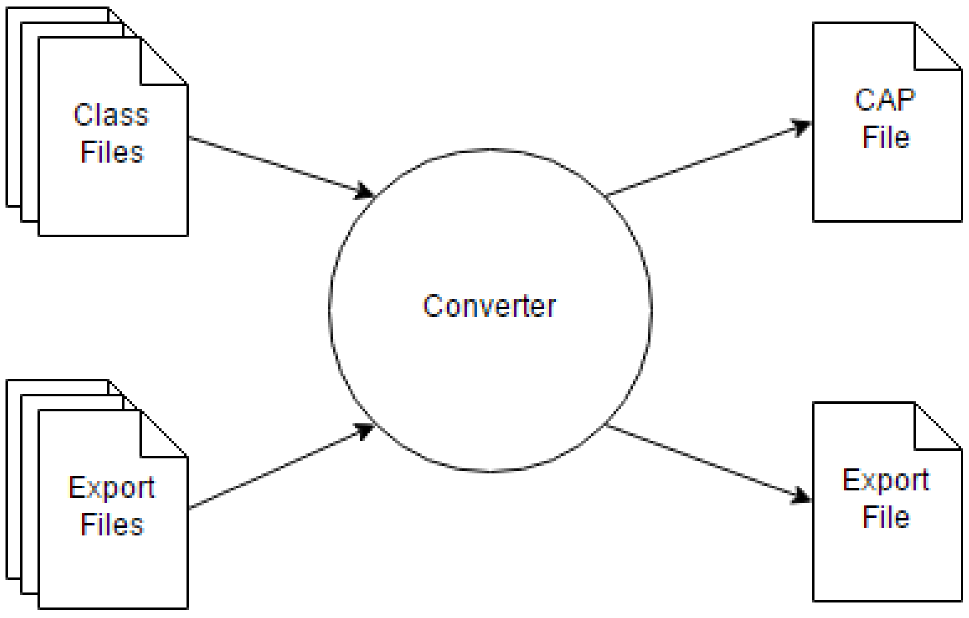


Figure 4-3: Converting a package

4.2.1.3 Java Card interpreter

The Java Card interpreter provides independence of application code from the hardware environment of the card. The interpreter performs the following procedures:

- Executes the binary code and therefore the applications.
- Controls the memory reservation and objects creation.
- Ensures the safety of the application during execution on the card.

4.2.1.4 The Java Card subset

Because of the limited capabilities of smart cards' memory and computational power, Java Card language supports only an appropriately selected subset of Java language. This subset, maintains the object oriented features of the Java language and is suitable for writing programs not only on smart cards but also to other devices with limited memory and computing power. Table 4-1 presents some basic supported or not, characteristics of Java language by Java Card. Note that some advanced smart cards support garbage collection mechanism to implement object deletion.

Supported Java features	Non supported Java features
• Small basic data types: Boolean, byte, short • One dimension arrays • Java packages, classes, interfaces and exceptions • Object oriented Java characteristics: inheritance, abstract methods, overloading and abstract object creation, access scope binding rules. • 32 bit integer support is optional	• Big basic data types: long, double, float • Characters and strings • Multi dimensional arrays • Dynamic class loading • Security management • Garbage collection • Threads • Serialize objects • Clone objects

Table 4-1: Supported and non supported Java features in Java Card

4.2.2 Java Card Runtime Environment

Java Card Runtime Environment (JCRE) consists of elements of Java Card system, which run on smart card. The runtime is responsible for managing the resources of the card, communication networks and the secure operation of applications. As seen in Table 4-1**Error! Reference source not found.**, Java Card Runtime Environment is located above the card's operating system and the native system. JCRE consists of Java Card Virtual Machine (JCVM) (i.e. binary code interpreter) from Java Card Framework and the set of Java Card (API) classes and industry specific extensions. JCRE is responsible for hiding the details of the card hardware from applications, making the programming of Java Card applications easier and applicable on different smart cards architectures.

The lower level of JCRE consists of Java Card Virtual Machine and native methods. Native methods provide support to Java Card Runtime Environment and next level's classes, but also are responsible for handling low-level communication protocols for memory management and cryptographic support.

Java Card Framework defines the Application Programming Interface (API). A business can add extensions to implement new services or enhance existing ones. In this way, the Java Card allows control of the cards from the companies that adopt them and widen the applications scope of a smart card.

4.2.2.1 Life Cycle of Java Card Runtime Environment

The life cycle of the virtual environment of the Java Card is identical to the smart card’s one. It starts from the time of manufacture and testing of the card before reaching the market, and is completed when the card’s operation terminate. Specifically, the initialization of the virtual environment is done only once in the lifetime of the card during the initialization phase. In this phase the virtual machine initializes and creates objects that will provide the services of virtual environment and application management. When applications are installed on the card, the virtual environment creates snapshots of these applications and these applications create objects for data storage.

Most information contained in the card should be maintained even after removing the card from the terminal. For this reason some data stored in a permanent storing memory card (EEPROM). In fact when the card is removed from power the virtual machine does not stop its operation, just turn into idle state. The virtual environment state and the objects created are maintained.

When the card is put back in service, the virtual environment starts the operation of the virtual machine by downloading data from the EEPROM memory. The disadvantage in this case is that the operation of the virtual machine does not start from where it left off when it lost power but starts execution from the beginning.

4.2.3 Java Card Simulators

The Java Card SDK provides the tools for developing and managing applets that can be executed by smart cards. Apart from the necessary tools for converting and uploading an applet in-card, the SDK provides two simulators that simulate the hardware environment of a smart card allowing the execution of a Java Card applet as if it were masked in a smart card’s ROM for developing applications without the need of real hardware present and gives the ability to the developer to debug the applications developed or even collect valuable statistics regarding the resources required from a smart card. These two simulators cover different aspects of the development phase and both have their advantages and disadvantages, therefore both used for the development of this project’s applet and are presented bellow in detail.

4.2.3.1 JCWDE Simulator

The Java Card platform Workstation Development Environment (Java Card Workstation Development Environment or Java Card WDE) tool allows the simulated running of a Java Card applet as if it were masked in ROM. It emulates the card environment.

The JCWDE is not an implementation of the Java Card virtual machine. It uses the Java virtual machine to emulate the Java Card RE. Class files that represent masked packages must be available on the classpath for the JCWDE.

JCWDE comes with some disadvantages which are shown below:

- lack of package installation procedure
- cannot store the state of the simulation for further use
- does not simulate firewall
- transactions are not supported
- transient array cannot be cleared
- does not support object, applet and package deletion

But besides these limitations, JCWDE is considered an invaluable tool, especially in the beginning of the development phase of an applet since it provides the following advantages:

- Provides a good initial test
- Executes applet as though it was masked
- Can be executed with many instances in multiple communication ports
- Able to execute in debugging mode

The JCWDE tool uses the *jcwde.jar*, *api.jar* (with cryptography extensions) and *apduio.jar* files. The main class for JCWDE is *com.sun.javacard.jcwde.Main*. A sample batch and shell script is provided for JCWDE execution.

The applets to be configured in the mask during JCWDE simulation need to be listed in a configuration file that is passed to the JCWDE as a command line argument. Also, the *CLASSPATH* environment variable needs to be set to reflect the location of the class files for the applets to be simulated. In some releases, the sample applets are listed in a configuration file called *jcwde.app*. Each entry in this file contains the name of the applet class, and it's associated AID.

The configuration file contains one line per installed applet. Each line is a white space(s) separated {*CLASS_NAME AID*} pair; where *CLASS NAME* is the fully qualified Java name of the class defining the applet, and *AID* is an Application Identifier for the applet class used to uniquely identify the applet. *AID* may be a string or hexadecimal representation in form: *0xXX*. Where the construct *0xXX* is repeated as many times as necessary.

Should be noted that the *AID* must be in the range of 5 to 16 bytes in length.

For example:

```
edu.teicrete.msc.thesis.Node 0x01:0x02:0x03:0x04:0x05:0x06:0x07:0x08:0x09:0x00:0x01
```

The general format of the command to run the Java Card WDE and emulate the Java Card RE is:

```
jcwde [-help] [-verbose] [-p port] [-t0] [-version] [-nobanner]
      <config-file>
```

Option	Description
<config-file>	The configuration file described above.
-help	Prints a help message.
-nobanner	Suppresses all banner messages.
-p port	Allows you to specify a TCP/IP port other than the default port.
-t0	Runs T=0 single interface only.
-verbose	Prints more verbose output.
-version	Prints the JCWDE version number.

Table 4-2: Command line options for JCWDE simulator

Table 4-2 describes the command line options for JCWDE.

JCWDE starts listening for APDU commands in T=1 as the default communication protocol, unless otherwise specified, on the TCP/IP port specified by the *-p port* parameter for contacted and *port+1* for contactless. The default port is 9025.

4.2.3.2 CREF Simulator

The Java Card reference implementation is written in the C programming language and is called the C-language Java Card Runtime Environment ("C-language Java Card RE"). It is a simulator that can be built with a ROM mask, much like a real Java Card technology-based implementation. It has the ability to simulate persistent memory (EEPROM), and to save and restore the contents of EEPROM to and from disk files. Applets can be installed in the C-language Java Card RE. The C-language Java Card RE performs I/O via a socket interface, using the TLP-224 protocol, simulating a Java Card technology-compliant smart card in a card reader (CAD).

The implementation of the CREF simulator is supplied as prebuilt executables and located in %JC_HOME%/bin/cref.exe where %JC_HOME% is the name of the environment variable locating at Java Card SDK.

The Development Kit Installer, the Java Card virtual machine interpreter, and the Java Card platform framework are built into the installer mask. It can be used as-is to load and run applets. Other than the Installer, it does not contain any applets. The C-language Java Card RE requires no other files to start proper interpretation and execution of the mask image's Java Card bytecode.

The command line usage of CREF simulator is the same for all supported operating systems. The syntax is:

```
cref [options]
```

The output of the simulation is logged to standard output, which can be redirected to any desired file. The output stream can range from nothing, to very verbose, depending on the command line options selected.

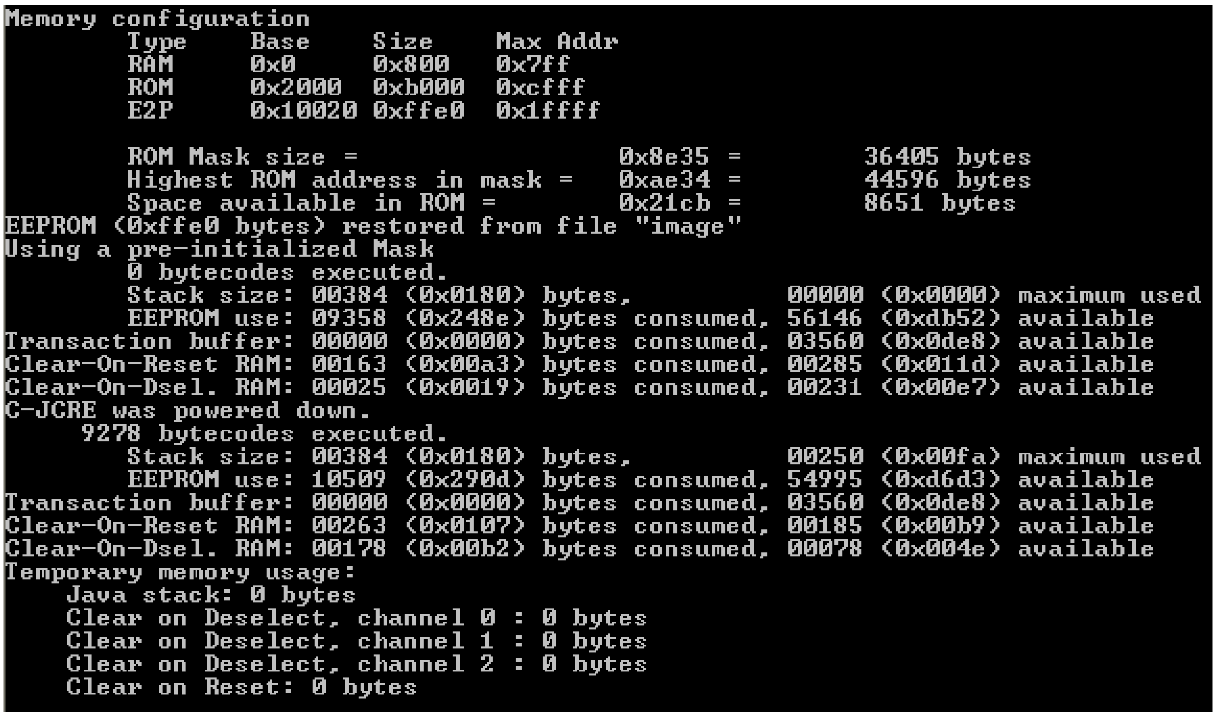
Option	Description
-b	Dumps a Byte Code Histogram at the end of the execution.
-e	Displays the program counter and stack when an exception occurs.
-h, -help	Prints a help screen.
-i <input filename>	Specifies a file to initialize EEPROM. File names must be single part; there can be no spaces in the file name.
-n	Performs a trace display of the native methods that are invoked.
-nobanner	Suppresses the printing of a program banner.
-nomeminfo	Suppresses the printing of memory statistics when execution starts.
-o <output filename>	Saves the EEPROM contents to the named file. File names must be single part; there can be no spaces in the file name.
-p <port number>	Connects to a TCP/IP port using the specified port number.
-s	Suppresses output. Does not create any output unless followed by other flag options.
-t	Performs a line-by-line trace display of the mask's execution.
-version	Prints only the program's version number. Do not execute.
-z	Prints the resource consumption statistics.

Table 4-3: Command line options for CREF simulator

Table 4-3 describes the available command line options when executing the CREF simulator.

The CREF simulator provides a command line option (-z) for printing resource consumption statistics. This option enables the CREF to print statistics regarding memory usage once at startup and once at shutdown. Although memory usage statistics will vary among Java Card RE implementations, this option provides the applet developer with a general idea of the amount of memory needed to install and execute an applet.

The output shown in Figure 4-4 is obtained by executing this project's applet with the -z command line option, thus, CREF provides us with resource statistics of the specified applet.



```
Memory configuration
        Type     Base     Size     Max Addr
        RAM      0x0      0x800    0x7ff
        ROM      0x2000   0xb000   0xcfff
        E2P      0x10020 0xffe0   0x1ffff

        ROM Mask size =                 0x8e35 =        36405 bytes
        Highest ROM address in mask =   0xae34 =        44596 bytes
        Space available in ROM =        0x21cb =        8651 bytes
EEPROM (0xffe0 bytes) restored from file "image"
Using a pre-initialized Mask
        0 bytecodes executed.
        Stack size: 00384 (0x0180) bytes,          00000 (0x0000) maximum used
        EEPROM use: 09358 (0x248e) bytes consumed, 56146 (0xdb52) available
Transaction buffer: 00000 (0x0000) bytes consumed, 03560 (0x0de8) available
Clear-On-Reset RAM: 00163 (0x00a3) bytes consumed, 00285 (0x011d) available
Clear-On-Dsel. RAM: 00025 (0x0019) bytes consumed, 00231 (0x00e7) available
C-JCRE was powered down.
     9278 bytecodes executed.
        Stack size: 00384 (0x0180) bytes,          00250 (0x00fa) maximum used
        EEPROM use: 10509 (0x290d) bytes consumed, 54995 (0xd6d3) available
Transaction buffer: 00000 (0x0000) bytes consumed, 03560 (0x0de8) available
Clear-On-Reset RAM: 00263 (0x0107) bytes consumed, 00185 (0x00b9) available
Clear-On-Dsel. RAM: 00178 (0x00b2) bytes consumed, 00078 (0x004e) available
Temporary memory usage:
    Java stack: 0 bytes
    Clear on Deselect, channel 0 : 0 bytes
    Clear on Deselect, channel 1 : 0 bytes
    Clear on Deselect, channel 2 : 0 bytes
    Clear on Reset: 0 bytes
```

Figure 4-4: CREF simulator usage statistics output

This particular example shows the resources used to upload and install this project's applet and calling the new encrypt session method along with encrypt method and encrypting a byte array. More fine-grained statistics could be obtained by limiting the actions during a single session. For example, using a single session to upload one application would provide information regarding the resources needed to process the application upload. The EEPROM contents at the end of the session could be saved using the -o option, and subsequent sessions could be used to measure resource usage for other actions, such as applet installation and execution.

In addition to the command line option, the Java Card API provides programmatic mechanisms for determining resource usage such as *javacard.framework.JCSystem.getAvailableMemory()* method.

Apparently CREF comes with some disadvantages which are shown below:

- The maximum number of remote references that can be returned during one card session is 8.
- The maximum number of remote objects that can be exported simultaneously is 16.
- The maximum number of parameters of type array that can be used in remote methods is 8.
- The maximum number of Java Card API packages that CREF can support is 32.
- The maximum number of library packages that a Java Card system can support is 32.
- The maximum number of applets that a Java Card system can support is 16.
- Does not have debugging capabilities

Admittedly CREF is considered a more robust simulator than JCWDE but its lack for debugging makes JCWDE simulator a valuable tool especially in the early stage of development where debugging is important. At a later stage CREF offers a more precise simulation environment and valuable usage statistics, a feature desirable at the later stages of development.

Therefore CREF is considered a robust simulator with many advantages, such as:

- Supports objects deletion
- Integer data type
- Executes applet as though it was masked
- Can be executed with many instances in multiple communication ports
- Applets resources needed statistics
- card reset in case of object allocation during an aborted transaction
- Able to store the state of EEPROM and recall for later use

As mentioned above CREF can save the state of EEPROM contents and then load it in a later invocation. To do this, should be specified an EEPROM image or "store" file to save the EEPROM contents.

The use of -i and -o flags manipulate EEPROM image files at the CREF command line:

- The -i flag, followed by a filename, specifies the initial EEPROM image file that will initialize the EEPROM portion of the virtual machine before Java Card virtual machine bytecode execution begins.
- The -o flag, followed by a filename, saves the updated EEPROM portion of the virtual machine to the named file, overwriting any existing file of the same name.

The -i and -o flags do not conflict with the performance of other option flags. File names used with the -i and -o flags flags must not contain spaces.

The commit of EEPROM memory changes during the execution of CREF is not affected by the -o flag. Neither standard nor error output is written to the output file named with the -o option.

4.2.4 Java Card Application Programming Interface (Java Card API)

This is a set of rules that defines the appropriate classes of Java Card for programming smart cards according to the standard ISO 7816. It consists of three basic packages, *java.lang*, *javacard.framework, javacard.security* and one expansion package, the *javacardx.crypto* package.

It is obvious that many classes defined in the Java API are not supported by Java Card's API. For example, classes for graphical user interfaces (GUI) and networks Input / Output are not supported. The reason is that graphical interface is not supported, and that the communication mode of Java Card with the external environment is different. Java Card classes support cryptographic services, while there are special classes designed to support standard ISO 7816.

4.2.4.1 Java.lang package

Java Card's java.lang package is a subset of Java's java.lang package. Supported classes are Object, Throwable and some relevant with virtual machine exception classes (exceptions), as shown in Table 4-4. Many of the methods of the corresponding Java classes are not available. The Object class defines the root of the hierarchy of Java classes and *Throwable* class a common ancestor from which they inherit all exception classes. The exception classes detect errors that violate the definition of a subset of the Java Card.

Object	Throwable	Exception
RuntimeException	ArithmeticException	ArrayIndexOutOfBoundsException
ArraystoreException	ClasscastException	IndexOutOfBoundsException
NullPointerException	SecurityException	NegativeArraySizeException

Table 4-4: Java.lang package classes

4.2.4.2 Javacard.framework package

Javacard.framework package provides the basic classes and interfaces for the functionality of a Java Card application. The package defines the basic class Applet, which is the basic framework for applications and communication with the runtime environment during their lifetime. The class of an application should inherit from the basic class Applet and override methods.

Another important class of this package is the class APDU. APDU messages are used by transport protocols for communicating the card with the terminal. The APDU class is designed to be independent of the various transport protocols, so developers do not worry about the differences between protocols and to handle with ease the APDU messages. *Javacard.framework* package also contains classes with methods that control the execution of an application, resource management and receiving and verification of a PIN code.

4.2.4.3 Javacard.security package

Javacard.security package provides a framework for cryptographic functions supported by Java Card platform. Its design is based on the package *java.security*.

In *javacard.security* package is defined the keyBuilder class and various interfaces for creating cryptographic keys using symmetric (AES, DES, 3DES) or asymmetric (RSA and DSA) algorithms. Also in this package are defined the classes RandomData, Signature and MessageDigest used to generate random data and to produce message hashes and digital signatures.

4.2.4.4 Javacardx.crypto package

Javacardx.crypto package is an extension package. It contains cryptographic classes and interfaces that are subject to United States algorithms export regulations. In this package, class Cipher defined to support encryption and decryption functions.

Javacard.security and *javacardx.crypto* packages define interfaces called by applications when applying cryptographic services but provide no implementation. Usually smart card manufacturers should offer classes that inherit from classes *RandomData*, *Signature*, *MessageDigest* and *Cipher*. For the implementation of cryptographic algorithms that require difficult numerical calculations, another microprocessor is available in card for this purpose.

4.3 Java Card Applets

A Java Card Applet should not be confused with a Java application. A Java Card Applet is essentially a Java application that is subject to some restrictions, to be executed from Java Card runtime environment. A Java Card Applet can be loaded dynamically in the card even after its construction, meaning that is not necessary to be loaded in card's ROM during the construction of the card.

All Java Card applications inherit from the base class Applet from *javacard.framework* package. The Applet class is the super class of all Java Card applications in which are defined all methods of an application. An application running on the card is a snapshot of the application, i.e. an object of class Applet. Like all persistent objects, from the moment of creation, one application remains permanently on the card.

The runtime environment supports the existence of multiple applications on the same card. More than one application can coexist in the same card and an application can have more than one snapshot. For example, in an electronic wallet application can be created a snapshot that supports dollars and another to support British pounds.

4.3.1 Java Card Application Identifier

In Java Card Platform, the snapshot of each application is uniquely identified and selected based on a number sequence named Application Identifier (AID). The AID numbers does not only uniquely identify the Java Card applications but Java Card packages too. If a package is loaded into a smart card, associates with other packages using the AID number.

This numerical scheme is defined in ISO 7816-5 standard. This model indicates that AID numbers are used for unique identification of the card applications and certain types of files in the card file system. This is a sequence of bytes, which can be divided into two sections. The first section has a length of 5 bytes and called Resource Identifier (RID). The second portion may have a variable length, called Proprietary Identifier Extension (PIX). The PIX section may have a length of 0 to 11 bytes, so the total length of an AID ranges from 5 to 16 bytes.

According to ISO 7816 standard, the RID part is assigned to companies: each company has its own unique RID, which is registered to ISO registry. The companies then use PIX to uniquely identify their packages and applications. The AID of a package or application is created by combining the RID of the company with the PIX of the package / application.

AID numbers of a package and every application specified in it, are specified in CAP's file content.

4.3.2 Life cycle of a Java Card Applet

The life cycle of a Java Card applet starts from the time is correctly loaded into card's memory and is ready for execution. The runtime environment starts to communicate with the applet through its public methods *install, select, deselect, and process*.

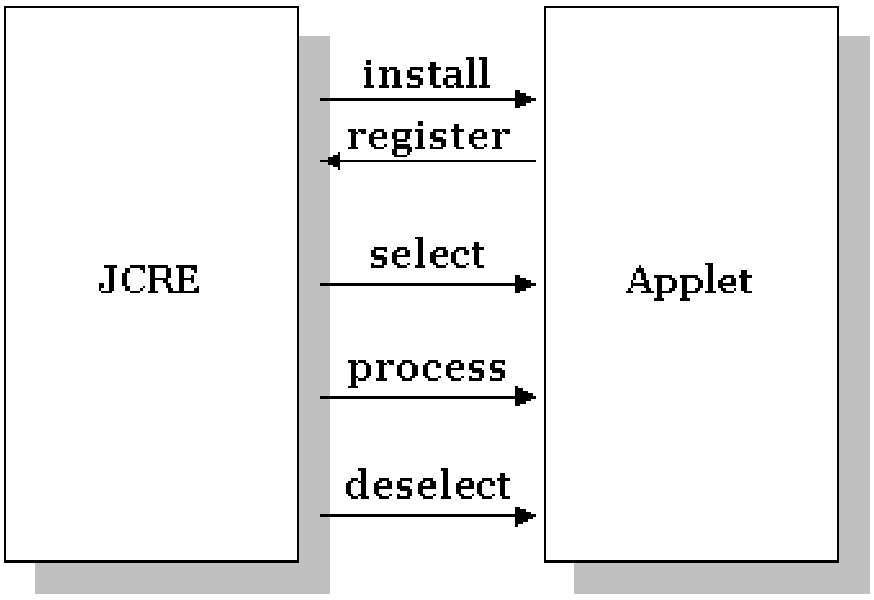


Figure 4-5: Life cycle of a Java Card Applet

4.3.2.1 `Install` method

At the time the method *install* (byte [], short, byte) is called, no application objects exist. The main function of the *install* method within the code of an application is to create an instance of Applet subclass, using its constructor and linking that snapshot to the runtime environment. All other items that may be required during the execution of the application can be created whenever necessary. The same applies for the necessary preparations to be made for selection and processing of the application from terminal.

```
/*
* Installs applet and creates an instance of  MyApplet.
* JCRE calls this method during installation phase.
* @param bArray installation parameters array.
* @param bOffset installation parameters offset.
* @param bLength installation parameters byte length.
* @throw ISOException if method fails
*/

public static void install(byte[] bArray, short bOffset, byte bLength) throws ISOException
```

Figure 4-6: install method

Install method (Figure 4-6) receives the initialization parameters from the body of the incoming APDU command. This method must call directly or indirectly the *register* method. Indirect call to *register* is done by calling the constructor of the applet.

Constructor

Through constructor is allocated the memory required for the needs of the application during its life cycle and make all necessary objects initialization that will be created during application's execution. The constructor is called from *install* method, i.e. only once during the lifecycle of the application. All memory reservation should be made before calling the *register* method.

```
private Wallet (byte[] bArray,short bOffset,byte bLength){

   // It is good programming practice to allocate
   // all the memory that an applet needs during
   // its lifetime inside the constructor
   pin = new OwnerPIN(PIN_TRY_LIMIT,   MAX_PIN_SIZE);

   // lastly constructor may call register function
  // to register this applet instance
   register();
 }
```

Figure 4-7: Applet constructor

Figure 4-7 shows the creation of *javacard.framework.OwnerPIN* object representing a PIN number. This object will be available throughout applet's lifetime.

With *register* method is determined the AID to be used for selecting the applet. The installation is successful when *register* call is completed without causing any exception. The installation is considered unsuccessful if *install* or constructor does not succeed to call *register*, if *register* call is made before *install*, or if calling *register* causes an exception. If installation is not successful the runtime environment will regain control, returning all persistent objects to their previous -from calling install- state. If installation succeeds the runtime environment marks the application as ready for selection.

4.3.2.2 `Select` method

Applets remain in idle state waiting to be selected. Selection occurs when the runtime environment receives a SELECT APDU command, in which the content data matches the AID of the applet. Before selecting an applet runtime environment must deselect all previously selected applets. To achieve this, the applet should call the *deselect* method. The applet may refuse its selection returning false value to the *select* call. If the method returns true, it mean that the applet is ready to process incoming APDU commands using the *process* method.

Figure 4-8 shows an example implementation of conditions controlling inside a *select* method, where the maximum number of PIN tries should not be reached, in order for the method to return true, if that is the case, the *process* method is called waiting for APDU commands.

```
public boolean select() {
  // The applet declines to be selected
  // if the pin is blocked.
  if ( pin.getTriesRemaining() == 0 )
    return false;

  return true;
}
```

Figure 4-8: Select method

4.3.2.3 `Process` method

Once an application is selected, is ready to accept commands from the terminal in the form of APDU messages. Each time the runtime environment receives an APDU command, calls the *process (APDU)* method passing as parameter the APDU command. Then, the method *process (APDU)* is responsible for the following:

1. Checks CLA and INS fields of the APDU command
2. Retrieves P1, P2 and data fields
3. Processes the APDU command data
4. Creates and sends an answer APDU.
5. In case of error, invokes an ISO exception

If *process (APDU)* method is completed correctly, the runtime sends an APDU with SW1 and SW2 fields' values 0x90 and 0x00 respectively. In case of invoking an *ISOException* exception, the runtime sends the fields SW1 and SW2 with the appropriate values of the exception that returned to the terminal. Figure 4-9 shows an example of the *process* method from JavaCard Wallet sample applet.

```
public void process(APDU apdu) {

  // APDU object carries a byte array (buffer) to transfer incoming
  // and outgoing APDU header data bytes between card and reader

  // At this point, only the first header bytes [CLA, INS, P1, P2, P3]
  // are available in APDU buffer.
  // The interface javacard.framework.ISO7816 declares constants
  // to denote the offset of these bytes in the APDU buffer

  byte[] buffer = apdu.getBuffer();

  buffer[ISO7816.OFFSET_CLA] = (byte)(buffer[ISO7816.OFFSET_CLA] & (byte)0xFC);

  if ((buffer[ISO7816.OFFSET_CLA] == 0) && (buffer[ISO7816.OFFSET_INS] == (byte)(0xA4)) )
   return;

  // verify the reset of commands have the correct CLA byte,
  // which specifies the command structure
  if (buffer[ISO7816.OFFSET_CLA] != Wallet_CLA)
    ISOException.throwIt (ISO7816.SW_CLA_NOT_SUPPORTED);

  switch (buffer[ISO7816.OFFSET_INS]) {
   case GET_BALANCE:   getBalance(apdu);
                  return;
   case DEBIT:        debit(apdu);
                  return;
   case CREDIT:       credit(apdu);
                  return;
   case VERIFY:       verify(apdu);
                  return;
   default:      ISOException.throwIt
              (ISO7816.SW_INS_NOT_SUPPORTED);
  }
}
```

Figure 4-9: Process method

4.3.2.4 `Deselect` method

The runtime environment calls *deselect* method to inform the applet that has been deselected. This method resets the elements required for executing another applet.

```
public void deselect() {

  // reset the pin value
  pin.reset();

}
```

Figure 4-10: Deselect method

Figure 4-10 shows an example of a *deselect* method resetting a pin object.

4.3.3 Writing a Java Card applet

The code writing of a Java Card applet is made using the Java Card API, which as we have seen contains all classes, methods and fields needed for this purpose. The process of developing a Java Card applet involves two steps:

1. Definition of command and response APDU messages, based on which the communication of the applet and the terminal is made.
2. Code writing of the Java Card applet.

4.3.3.1 Java Card applet structure

Figure 4-11 shows a typical Java Card applet's structure.

```
import javacard.framework.*
...
public class MyApplet extends Applet {
   // Definition of variables and their values
   // including CLA, INS, P1 and P2 expected values
   ...
   install() {...}              // one time called install method
   MyApplet() {...}             // one time called Constructor

   // Applet's operation basic methods
   select() {...}
   process() {...}
   deselect() {...}

   // specific operations private methods
   ...
}
```

Figure 4-11: Java Card applet structure

A Java Card applet defines the commands associated with the APDU messages, the constructor, its lifecycle methods, and finally all the necessary for its specific operation private methods.

4.3.3.2 Defining APDU messages

Different Java Card applets have different communication needs with the terminal. A credit card applet must support methods for importing and confirmation of a PIN code for credit and debit transactions, to control the balance of the card, etc. A health insurance applet should provide access to data including history patient, insurance coverage limits, the physician, etc. It is obvious that APDU messages vary depending on the application needs to be served [37].

As described in the corresponding section, the *process (APDU)* method processes the APDU commands sent from the terminal to the application and vice versa. The APDU messages have a header that contains the code for the command to be executed by the application and a body that contains data. Typically the method of *process (APDU)* processes the header to determine what command will be executed and then calls auxiliary methods, specified in the application for processing the data [38].

The general form of an APDU command has already been studied. Some basic commands are standardized by ISO 7816-4 standard. For example, the SELECT statement is determined by the value 0x00 for the CLA field and the value 0xA4 for the INS field. Specialized APDU commands values are defined by each application.

According to the Java Card API, the class *javacard.framework.APDU* contains all the methods required to manage APDU commands. This class provides a dynamic and flexible interface for handling APDU commands and is designed in such a way that hides the complexity of the communication protocol.

When Java Card runtime environment receives from the terminal an APDU command, converts the APDU command to an object and sends it as parameter to the *process ()* method of the selected application. The APDU object carries an array of bytes, representing the contents of the APDU command.

The Java Card applet processes the contents of the APDU command by calling the appropriate methods for the object APDU. Generally, the application performs the following steps [39]:

Step 1. Retrieves APDU buffer content

The applet calls the *getBuffer* method to obtain a reference to the APDU buffer, which contains the message command. When the application receives the APDU object, only the first 4 bytes of header APDU is available in the buffer, i.e. bytes CLA, INS, P1 and P2, which determine the structure and type of command.

Step 2. Receive data

If the command contains data to be received and processed by the card, the application should call the *setIncomingAndReceive* method. The data loads in the APDU buffer immediately after the first 4 bytes of header. The last byte of the header (Lc) indicates the length of data that follows. If does not all data fit in the buffer, the applet can process them piece by piece or copy them to an internal buffer. In any case, is possible to repeatedly call the *receiveBytes* method to read the additional data from the buffer.

Step 3: Return data

After processing and execution of an APDU command, the application will most likely need to return data to the terminal with an APDU response. In this case the application should call the *setOutgoing* method which does not send data, but prepares the runtime environment for sending data. This method returns to runtime the Le value, i.e. the expected length of the response data.

Then the application calls the *setOutgoingLength* method to update the terminal how many bytes will be transferred in response. Then, the application must transfer the data you want to send the APDU buffer and call the *sendBytes* method. This method can be called repeatedly if the data to be sent to the terminal does not fit all in the APDU buffer. If the response of the data stored in an internal buffer, the application calls the *sendByteLong* method to send data from the internal buffer. If the response data is of sufficient size to fit into the APDU buffer, the APDU class provides the *setOutgoingAndSend* method. This method is a combination of methods *setOutgoing*, *setOutgoingLength* and *sendBytes*, but can be called only once and no other method of sending data can be called from there.

Then the applet calls the *setOutgoingLength* method to inform the terminal how many bytes will be transferred as response. Then, the application must transfer the data to the APDU buffer and call the *sendBytes* method. This method can be called repeatedly if the data to be sent to the terminal are larger than the APDU buffer. If the response data were stored in an internal buffer, the application calls the *sendByteLong* method to send data from the internal buffer. If the response data has sufficient size to fit into the APDU buffer, the APDU class provides the *setOutgoingAndSend* method. This method is a combination of methods *setOutgoing*, *setOutgoingLength* and *sendBytes*, but can be called only once and no other method of sending data can be called after.

Step 4. Return status words

After a successful *process* method call, the runtime sends the hexadecimal sequence 0x9000 indicating a successful processing. If at any point, the applet detects an error, it may cause an *ISOException* exception calling the class method *ISOException.throwIt*. The status word is defined in the parameter reason. If the applet does not handle the *ISOException*, then it would be invoked by the runtime environment, which determines the reason parameter and passes it to the status word.

To facilitate processing of the APDU messaging interface, javacard.framework.ISO7816 defines a set of constants that can be used for recovering various fields of APDU command contained in the buffer.

4.3.4 Development Process for Java Card applets

The process of developing a Java Card applet starts as any other java program. A programmer writes the source code, which is then compiled by a Java compiler creating one or more class files. Once compiled, optionally, the applet may be run and tested in a simulation environment. The simulator simulates the Java Card Runtime Environment in a computer environment where there is no smart card for implementing the application.

The simulation of Java Card applets can be done using the Java Card Development Kit, which contains all the basic tools needed for developing and testing a Java Card applet. In particular it provides an environment dedicated to the development and implementation of out of card applets called JCWDE (Java Card Workstation Development Environment). During the simulation, applet functionality is controlled, but many features of the Java Card Virtual Machine running on the card, such as the applet firewall and transient persistent objects are not supported. Most Java Card simulators contain a debugger which allows developers to monitor the way in which applet's execution state changes, thereby helping making any improvements.

Next, applet's class records constituting a Java Card package are converted into a CAP file using the Java Card converter. As previously mentioned, Java Card converter takes as input not only the class files to be converted but also one or more export files. When a Java Card package has been converted, the converter generates an export file for this package. A CAP file or export file represents a Java Card package. If an applet uses several packages, then a CAP file and an export file is created for each package.

Finally, an applet can be executed from a smart card after the one or more CAP files representing the applet have been loaded to the card.

4.3.5 Applet installation

During construction of a Java Card smart card, the card's operating system and Java Card Runtime Environment "burned" in card's ROM. The registration process of permanent features in the nonvolatile memory card is called masking and the technology with which it is implemented depends on the smart card manufacturer.

4.3.5.1 ROM applets

The ROM applets are Java Card applets, the classes of which can be written (covered) in the ROM of the card along with the runtime environment and other system components during construction of the smart card. Snapshots of those applets are created in EEPROM by the runtime environment during initialization or at some later stage.

ROM applets are standard applets provided along the card from manufacturers. Because the contents of ROM applets are specified by the smart card manufacturers, the Java Card technology enables these applets to be written in C programming language or symbolic language (assembly code). These applets are not subject to security checks performed by the Java Card Virtual Machine.

4.3.5.2 Applets installation before or after the card's issuance

In Java Card technology, applet classes and related class libraries can be stored to the nonvolatile memory of the card (such as EEPROM) after construction. Such applets can be categorized as applets installed in the card after its issued (post-issuance applets) and applets installed on the card before its issued (pre-issuance applets). The latter, just like ROM applets, are subject to control of the issuer of the smart card.

Contrary to ROM applets and applets installed on the card before its issuance, the applets installed in the card after its issuance, can not contain native methods. This would compromise the safety of the Java Card platform since the runtime environment has no way to verify the contents of these applets.

The installation mechanism for applets loaded in the card before its issuance is usually the same as the applets loaded in the card after its issuance. However, the Java Card technology allows issuers to apply different installation mechanisms. The next section focuses on installing applets that are loaded in the card after its issuance.

4.3.5.3 Applets installation after the card's issuance

The term applet installation is referring to the process of loading the classes in a CAP file, their execution by the runtime environment and the creation of an instance of the applet, so that the applet will be eligible and executable.

In Java Card platform, a CAP file consists of classes that make up a Java Card package. A simple applet consists of a Java Card package with only one class, while, a more complex applet that contains more than one class may be organized into one or more Java Card packages.

To load an application, an installer outside of the card gets the CAP file and converts it into a set of APDU commands, enclosing the content of the CAP file. By exchanging these APDU commands, an installation program on the card writes the contents of the CAP file in card's permanent memory and connects the CAP file classes with other classes on the card. The same installation program creates and initializes any parameters used internally by the runtime environment to support the applet. If the running applet requires multiple packages, then are loaded on the card as much CAP files as those packages required.

In the last step of the installation, the built in card installation program creates an instance of the applet and connects it with the runtime environment. To achieve this, the *install* method is called, which is the entry point for the applet, just as the main method in a Java application, and must be present in every Java Card applet. This method calls the applet's constructor to create and initialize an instance of the applet. The method's parameter *bArray* provides parameters for installing the application during initialization. The setup parameters are sent to the card with the CAP file and their content is determined by the developer of the application.

After initializing and linking the applet with the runtime environment, the applet can be selected and executed. The runtime environment defines the applet for executing using an AID number. The install method can be called multiple times to create multiple instances of the applet. In this case each instance is characterized by a unique AID.

4.3.5.4 Recovering from errors during installation

In case there is a problem during installation of an applet, which may be due for example to a bug, memory insufficiency, or even malfunction of the card, the installer rejects the CAP file and instances created and recovers memory wasted and previous state of the runtime environment.

4.3.5.5 Installation limitations

Installing a Java Card applet is very different from the dynamic loading of classes at runtime supported by the Java Virtual Machine in a computer environment. Installing Java Card applets simply means, loading classes on a card after its construction.

Therefore, a Java Card applet installation process has two sensitive points. Firstly, the execution of an applet on the card can only refer to classes that already exist on the card, since there is no way to load classes on the card during execution of the code of an applet.

Secondly, the loading of packages on the card should be done in such a way that the new packages loaded should refer only to packages that are already on the card. For example, to install a package, *javacard.framework* package should be contained in the card, because all classes of a need to inherit from the class *javacard.framework.Applet*. If the loading package A inherits from the package B, which is not on card, then the installation fails.

4.4 Java Card objects

In a Java Card device, memory is the most important resource. But contrary to Java technology, the Java Card technology does not allow dynamic memory management. The execution of a Java Card applet requires access both to persistent and transient objects. The limited memory and the physical characteristics of a smart card require simple and transparent handling of objects for both types of memory. One of the greatest challenges of Java Card technology is the adaptation of the Java memory model to the limited hardware characteristics of a smart card [21].

4.4.1 Persistent objects

By default objects created in a Java Card applet is permanently stored in card’s EEPROM memory. Permanence refers to the ability of an object to be kept in the memory card even after the communication with the terminal or even after a sudden loss of power supply to the smart card. The object is kept in memory while there is a reference pointing it. If the reference is lost, then the object is not accessible while occupying space in the memory. The solution to this problem would be a records management system with garbage collection, a feature available in the latest Java Card platforms.

4.4.2 Transient objects

For security and performance reasons, applets can create objects in the volatile RAM memory. These objects are called transient objects. Transient objects contain data that are lost after the termination of connection to the terminal. Persistent data can be converted to transient while the reverse process is not feasible.

The term "transient objects" can easily be misunderstood and considered that the objects themselves are temporary. In reality, however, only the data contained in these objects are temporary. As for persistent objects, transient objects exist if there is a reference to them [40].

In Java Card API (`javacard.framework.JCSystem`) are defined three methods for creating transient objects and a fourth that checks whether an object is transient:

- `static byte[] makeTransientByteArray(short length,byte event)`
- `static Object makeTransientObjectArray(short length,byte event)`
- `static short[] makeTransientShortArray(short length,byte event)`
- `static byte isTransient(java.lang.Object theObj)`

A transient array may be created with data types of byte or short, or a temporary *Object* variable. In any case a programmer should take into account the following features of temporary objects' behavior:

- Fields of transient objects should get their default value (zero, false or null) when events occurring such as card restarting or deselection of the applet.
- For security reasons, the fields of a transient object should never be stored in non volatile memory of the card. In today's smart card technology, transient objects can be stored in memory RAM, but never in the EEPROM memory. In this manner the transient objects may be used for storing cryptographic keys.
- Storing in the fields of transient objects does not impact system's performance. Typically, the RAM technology has much faster write cycle from EERROM technology.
- Storing in the fields of transient objects is not affected by transactions.

The above characteristics of transient objects make them ideal for storing small data which are modified frequently and does not need to be maintained during the communication with the terminal.

4.4.3 Atomicity and transactions

The term transaction is used to indicate a logical set of updates for persistent objects. For example, the transfer of a set of funds from one account to another is a banking transaction. It is very important for a transaction to be atomic: either all fields of an object would be updated or none. The runtime environment provides strong support for atomic transactions so that the card's data can be stored in the previous transaction situation if for some reason this is not completed correctly. This mechanism of the runtime environment provides protection in case of a sudden power loss during a transaction and if a program error occurs that can cause data loss, if not all steps of the transaction are completed.

4.4.3.1 Atomicity

Atomicity defines how the card handles the content of a persistent repository after an abrupt interruption, failure or exception during an update of an object's field of a class or an array. If the card suddenly losses power, the programmer may rely on the contents of a field or an array after power is restored.

Java Card platform guarantees that any information in a field of a persistent object or a persistent class will be atomic. Also, the Java Card platform provides atomicity to persistent arrays. This means that if the card losses power during an update of a data item (the scope of an object, class or part of an array) that is persistent in all the connections with the terminal, this will recover the previous value after power is restored.

Some methods also guarantee atomicity for updating multiple data elements. For example, the atomicity of *Util.arrayCopy* method guarantees that all the bytes of an array are copied correctly, otherwise the destination array will return to the previous value.

An applet may not need atomicity for updating a field in an array. The *Util.arrayCopyNonAtomic* method is provided for this purpose.

4.4.3.2 Transactions

An application may require the atomic update of different fields or arrays in many different objects. Updates should be done correctly and consistently, or else the fields and the arrays return back to their previous values.

Java Card platform provides a transactions model in which an application can initiate a transaction by calling the *JCSystem.beginTransaction* method. Once called this method starts updating. Each update is completed by calling the *JCSystem.commitTransaction* method.

In case of power loss before the completion of *JCSystem.commitTransaction* method, all fields marked for updating return to their previous state. If the application encounters an internal problem or decide to terminate the transaction then calls the method *JCSystem.abortTransaction*.

Transactions duration

A transaction always ends when the runtime environment acquires the programming control after returning of methods select, deselect, process and install. This applies whether the transaction is completed normally or stopped by calling the *JCSystem.abortTransaction* method.

A transaction's duration is the interval of the call of *JCSystem.beginTransaction* method and the end of the transaction.

Nested transactions

Until now, Java Card platform does not allow nested transactions. Only one transaction can be executed at a time. If *JCSystem.beginTransaction* method is called during another transaction then an exception *TransactionException* is caused.

The *JCSystem.transactionDepth* method allows control of the existence of a transaction in progress.

4.5 Exceptions and exceptions handling

Exceptions are objects which are used for handling errors that may occur within a running Java Card applet. Exceptions allow programs to recognize errors and to respond to them. The management of exceptions in Java Card is the same with Java's, using the keywords *try*, *catch* and *finally*.

4.5.1 Exceptions of `java.lang` package

The *java.lang* package, as we have already seen, provides all necessary classes for programming in Java Card. Contains the base class *Object*, which is the root of the class hierarchy in Java Card. Each class has *Object* as superclass and all objects inherit methods of this class. Next in hierarchy after the *Object* class is the class *Throwable*.

The Throwable class is the superclass of all errors and exceptions caused in Java Card. Only objects that are instances of this class (or some of the subclasses) can cause an exception in the Java Card Virtual Machine as parameters in phrases *throw* and *catch*.

The next class in the hierarchy that inherits from the *Throwable* class is the class *Exception*. The *Exception* class and its subclasses indicate exceptions that may occur when running a Java Card applet.

From the class *Throwable* inherit the *CardException* and *RuntimeException classes*.

The *RuntimeException* class is the superclass of all those exceptions that may occur during the operation of the Java Card Virtual Machine. These exceptions are explained below:

- `ArithmeticException`: caused when a false arithmetic operation occurs, such as division by 0.
- `ArrayStoreException`: caused when attempting to store wrong type of data in an array object
- `ClassCastException`: caused when the program tries to access an object of a class which is not a snapshot
- `IndexOutOfBoundsException`: caused when the content of a data structure (e.g. an array) exceeds the size of the structure
- `NegativeArraySizeException`: caused if the application tries to create an array with negative size
- `NullPointerException`: caused when the application tries to use a null value where an object is required
- `SecurityException`: caused by the virtual machine to indicate a security violation. Security violation may occur when the applet attempts to access another applet's objects or when a call to a private method is attempted from another class

CardException class inherits from the *java.lang* package but belongs to the package *javacard.framework*. So it will be examined in the next section of the Java Card exceptions.

4.5.2 Java Card exceptions

The exceptions presented above are implemented on the Java Card platform, but are essentially a subset of the exceptions defined in the Java language in the package *java.lang*. The exceptions will be studied in this section are exceptions serving only Java Card technology, so they called Java Card exceptions.

4.5.2.1 Exceptions of `Javacard.framework` *package*

In *javacard.framework* package all exceptions have as superclass the *CardException* and *CardRunTimeException* classes. These exceptions are:

- `UserException`: This exception's superclass is the *CardException* class and is caused by the user. It takes as argument a parameter specifying the reason why the user caused the exception.
- APDUException: subclass of the *CardRunTimeException* class and is caused by the runtime environment in case of an error in a message APDU
- ISOException: subclass of the *CardRunTimeException* and takes as parameters a status word (status word-sw) whose value specifies the reason that caused the exception. The available codes of this status word are defined in the ISO 7816-4 standard
- PINException: subclass of the *CardRunTimeException* class and invoked in case of an error in the PIN code
- SystemException: its superclass is the *CardRunTimeException* class representing exceptions related to *JCSystem* class. This exception is also caused by the *register()* method and the constructor of the AID class
- TransactionException: its superclass is the *CardRunTimeException* class and caused in case of error in the transaction system of the runtime environment

4.5.2.2 Exceptions of `javacard.security` *package*

- CryptoException: the only exception defined in *javacard.framework* package. caused in case of an error caused during a cryptographic operation

4.6 Java Card programming techniques

The available memory of a smart card is the most important factor influencing the programming of Java Card applet. An applet must carefully manage memory and not wasting it. This limitation leads to some programming compromises which are not required in computers programming. Bellow are proposed some useful programming techniques for creating Java Card applets, in order to make the best possible utilization of the limited memory capabilities of a smart card.

Such programming techniques are described below

- *Local scope*: should be avoided creating objects with local scope. Each instance of an object or an array of local scope reserves space in memory. Smart cards do not support garbage collection, so the memory that is committed is not released. Each call to a local method binds memory until exhausting all memory resources.
- *Using constants*: is preferable to use the keywords *static* and *final* for the declaration of variables, so that variables behave as constants. This practice reduces the size of the applet and improves its performance.
- *Re-using variables*: the reuse of already created variables is proposed. The more new variables are created the more memory is consumed.
- *Local variables*: Local variables should be used as little as possible, because of memory impact.
- *Class hierarchy*: the class hierarchy should be kept simple. The call of classes within other classes occupies considerable memory space.
- *Parameters*: methods should not use more parameters than those that are absolutely necessary.
- *Methods call*: although calling methods from a method allows the articulation of the code, consumes more resources.
- *Objects freeing*: the program should always be checked for unused variables, constants, methods and classes. Before running the program in the converter, the program should be checked and any element that is not used should be removed.
- *Data types*: is preferable to use *short* data types and not *byte*. The *short* type consumes the same amount of memory as the *byte* type but with a bigger value range.

4.7 Security in Java Cards

Java Card platform was designed and developed from scratch for strengthening of smart card security. As a neutral platform, Java Card technology may be applied to a wide variety of smart cards of the security level ranging from low to very high. Modern features of Java Card provide a rich set of tools for developing reliable and secure applets. The Java Card Virtual Machine distinguishes applets from hardware and operating system of the card. The API of the Java Card provides a uniform way for developing different applications. This unique approach uses the advantages of widely accepted

The Java programming language specifies in its structure important safety features which characterize Java Card technology too. But Java Card's security is further enhanced by the use of transaction atomicity, cryptographic algorithms and applet firewall. Also Java Card security technology contributes dual architecture of the virtual machine on card and out card. Another feature is the use of compact files with cryptographic signatures providing secure distribution and installation of class files and Java Card applets.

4.7.1 Security derived from Java language

For the development of a Java Card applet, the Java programming language is used, providing features that protect the integrity of the platform:

- It is an object-oriented programming language that provides developers the means to combine the data only to those procedures that have exclusive access to them during processing.
- The language provides appropriate types nomenclature and procedures to control access to objects and methods. So the developer can use the keywords "public", "private", "protected" and "package-private" to control access to objects, methods and packages.
- Developers have the ability to reuse code already checked.
- Any mismatch between types and variables is detected during compiling code. Enhancing security at execution is based on the fact that the compiler creates logical continuous information data types.
- The binary code, generated by the compilation of a program is strictly designated by the specification sheets of Java. This code contains all the data types defined by the programmer for access control during runtime.
- Java language does not support pointers, protecting against security risks known in C and C++ languages.
- References to arrays cannot be used to access memory outside of the array to which they refer.
- The Java language provides transparent memory allocation, preventing programming errors that lead to unnecessary waste of memory.

The advantages of the Java language to achieve a desired security have been made unambiguously in the last years. Smart cards using different hardware and operating systems cannot obtain the desired level of security. But based on Java programming language, the Java Card technology is able to provide a level of security that meets the market requirements.

4.7.2 Java Card platform security techniques

Complementing the security features inherited from the Java language, Java Card platform enhances application security by providing a set of security techniques: applet firewall, object sharing and classes to support cryptographic operations and for authenticating CAP files.

4.7.2.1 Applet firewall

The Java Card platform provides a secure runtime environment using firewalls between different applets within the same card. Thus, a Java Card applet resides inside the card isolated from the other applets. The firewall is a feature of the Java Card runtime environment, which offers protection against the biggest safety concern: the leakage of sensitive data from one applet to another. An application can obtain a reference to an object located in a publicly available location. But if the object is owned by another applet of different package, firewall prevents access to that object. So, one dysfunctional applet or even a hostile applet can not affect the functionality of other applets or runtime environment.

Context

A firewall separates the Java Card object system into separate protected pieces of objects called contexts. The firewall is the boundary between two contexts. When an instance of an object is created, the runtime environment assigns it in a context called group context. All applet instances that belong to the same package share the same group context. There is no firewall between two applet instances in the same group context. Access to objects between applets belonging to the same group is permissible. On the other hand, access to applet’s objects belonging to a different group context is prohibited by the firewall.

The runtime environment maintains its own JCRE context. The context of runtime environment has particular advantages: it may access the context of any applet but the converse, i.e. access from an applet context to JCRE context is not permitted by the firewall. Figure 4-12 shows the separation of system objects in Java Card platform firewalls.

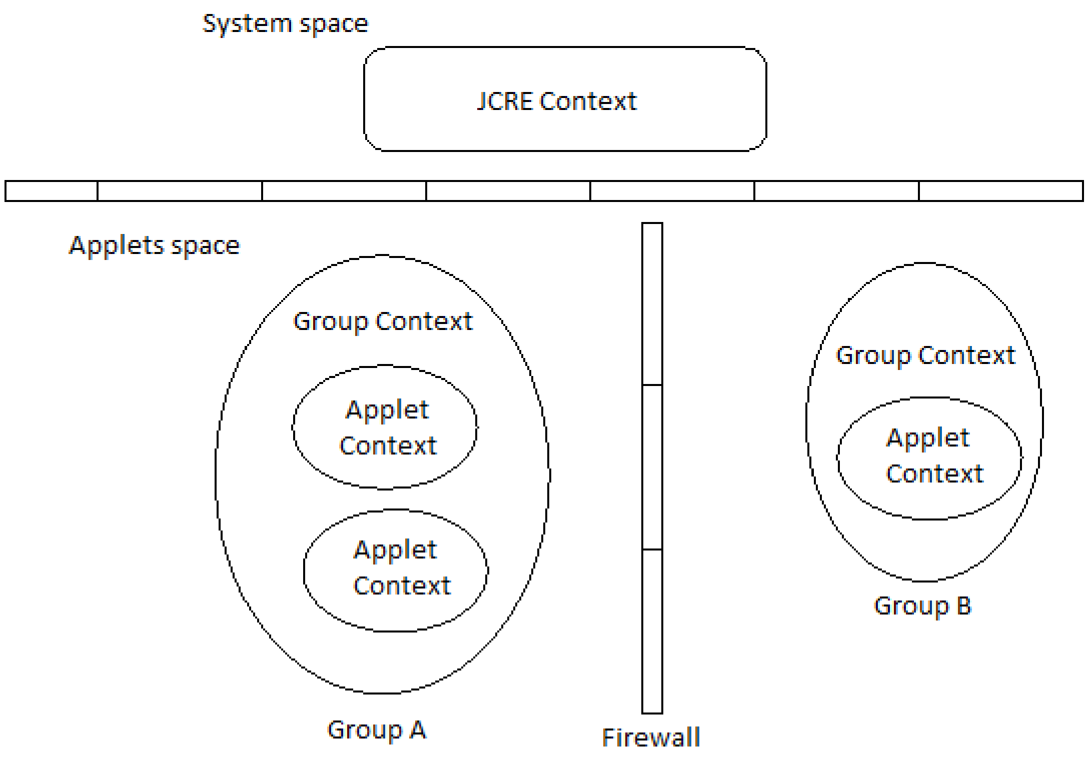


Figure 4-12: Separation of system objects in Java Card platform

Every time there is only one active context in the virtual machine: either JCRE context or applet group context that is running. When a new object is created, the currently active context is assigned to it. Access to the object is possible only by the applets that are in the active context. That new object is created belongs to the active applet of the current framework. If the current context is JCRE's context, then the object belongs to the runtime environment.

4.7.2.2 Object sharing

Firewall limits the actions of an applet in a specific context. The applet cannot reach beyond its context in order to access items belonging to the JCRE context, or in another applet of a different context. In cases, however, where the applets need to communicate with each other, the Java Card technology provides a set of well defined and secure mechanisms implemented by the following means:

- JCRE privileges
- JCRE Entry Point Objects
- Global arrays
- Shareable interfaces

JCRE privileges

In Java Card platform, runtime environment manages all processes that take place on the card. Because the context of the runtime environment is essentially the context of the card's system, it has special privileges. It is possible to call a method on any object or to access any field of an object inside the card. Such privileges relating to the management of the internal system of the card gives the runtime environment the ability to monitor system resources and manage applets.

When JCRE calls a method of an applet, the JCRE context changes the context of the specific applet. Now the control is passed to the applet which has no runtime environment privileges. Each object is created after changing the context belonging to the applet and related to the current context of the applet. When the applet completes, control returns to the runtime environment.

JCRE Entry Point Objects

The runtime environment may have access to the applet contexts, but applets cannot have access in the runtime environment. A secure computer system should support a mechanism that non-privileged users to be able to request system services performed by privileged system processes. In Java Card platform, the above requirement is handled by using entry point objects in runtime (JCRE Entry Point Objects).

JCRE Entry Point Objects are normally objects which belong to the runtime environment, but have been marked as containing entry point methods. Defining entry points allows public methods of these objects to be called from any context. When this happens, a context change is performed by the applet context under JCRE. So these methods are the means by which applications request privileged runtime environment services. Note that only the public methods of JCRE entry point objects are accessible through the firewall. The APDU object is probably the most used JCRE entry point object.

There are two categories of JCRE entry point objects:

- *Temporary JCRE Entry Point Objects*: like all other JCRE entry point objects, the methods of these temporary objects can be called from any context. But references to these objects cannot be stored in class variables, variables in existence or in array fields of an applet. The runtime environment detects and blocks any attempts to store such reports to protect the system from unauthorized applet reuse. The APDU object and all exception objects belonging to the runtime environment are examples of temporary JCRE entry point objects.
- *Permanent JCRE Entry Point Objects*: like all other JCRE entry points objects, the methods of these permanent objects can be called from any context. Also, references to these objects

can be stored and reused freely. The AID snapshots belonging to the runtime environment are examples of permanent JCRE entry point objects.

Only the runtime environment can determine the entry point objects and if they are permanent or temporary.

Global Arrays

In Java Card platform, the nature of some data requires to be accessible from any applet and from JCRE context. To achieve universal access to objects in a flexible way, the firewall allows the runtime environment to characterize primitive arrays global. Global arrays provide a shared memory buffer in which any applet and runtime environment can have access. Because it is only possible for a single context to operate at a time, there is no issue for synchronizing access.

Global arrays are a special type of JCRE entry point object. The firewall allows the public fields of these arrays to be freely accessible by any applet. Public methods of global arrays are treated in the same way as the methods of other JCRE entry point objects. The only method in the class of arrays is the method *equals*, which inherits from the superclass *Object*. The call of *equal* method of a global array causes the exchange of the current context with JCRE context.

Only primitive arrays, i.e. arrays containing primitive data types can be characterized as global and only the runtime environment define them. All global arrays should be temporary JCRE entry point objects. Thus, references to these objects cannot be stored in class variables, in status variables or array fields.

The only global arrays required in Java Card platform is the APDU buffer and bArray parameter of the *install* method. Typically JCRE passes the APDU buffer content to the bArray parameter of the *install* method. Because global arrays are available to anyone, the runtime environment resets the APDU buffer each time an applet is selected or before receiving a new APDU command, to protect sensitive applet data from a possible leak in another applet through the global APDU buffer.

Shareable interface

Java Card technology allows sharing of objects through the Shareable interface mechanism. A shareable interface is an interface that inherits directly or indirectly from the interface *javacard.framework.Shareable*

```
: public interface Shareable {}.
```

This interface does not define any method or field. Its sole purpose is to inherit from other interfaces and to mark these interfaces as they have special characteristics. Only the methods defined in an interface that inherits from the Shareable are available through the firewall. A class can implement any number of shared interfaces and can inherit other classes that implement shareable interfaces.

4.7.2.3 Cryptographic and Security classes

Cryptographic classes and security classes are implemented in packages *javacard.security* and *javacardx.crypto*. These packages support:

- symmetric encryption and decryption algorithms
- asymmetric encryption and decryption algorithms
- creation of symmetric key and public-private key pairs
- creating and verifying digital signatures
- message hashing
- random data generator
- PIN codes management

The above cryptography and security mechanisms allow the implementation of a secure mechanism to load Java Card applets in the smart card and the use of the card to accommodate services requiring increased security level.

The most important classes of *javacard.security* package are the following:

- *javacard.security.DESKey*: This interface provides access to the DES algorithm and the triple DES algorithm with two or three keys.
- *javacard.security.DSAKey* : This interface provides access to the DSA algorithm
- *javacard.security.DSAPrivateKey*: This interface provides access to the DSA algorithm for creating digital signatures.
- *javacard.security.DSAPublicKey*: This interface provides access to the DSA algorithm for verifying digital signatures.
- *javacard.security.Key*: the basic interface for all keys.
- *javacard.security.KeyBuilder*: class to generate keys.
- *javacard.security.KeyPair*: is a container type class that contains all the key pairs (private and public).
- *javacard.security.MessageDigest*: The base class of all hash algorithms.
- *javacard.security.PrivateKey*: is the interface for private keys used in asymmetric cryptographic algorithms.
- *javacard.security.PublicKey*: is the interface for the public keys used in asymmetric cryptographic algorithms.
- *javacard.security.RandomData*: is the base class for creating random numbers
- *javacard.security.RSAPrivateCrtKey*: is the interface for private keys used in asymmetric cryptographic algorithms in conjunction with the Chinese Remainder Theorem (CRT)

- *javacard.security.RSAPrivateKey*: This interface provides access to the RSA algorithm to create digital signatures.
- *javacard.security.RSAPublicKey*: This interface provides access to the RSA algorithm to verify digital signatures.
- *javacard.security.SecretKey*: is the interface for all symmetric keys.
- *javacard.security.Signature*: The base class for all algorithms digital signature.

The most important classes of `javacardx.crypto` package are the bellow:

- *javacardx.crypto.KeyEncryption*: This interface provides access to the keys used to encrypt or decrypt data.
- *javacardx.crypto.Cipher*: is the base class for all cipher algorithms.

4.7.2.4 Restrictions in cryptography and smart cards export

In many countries, special licenses are required for the export of smart cards that use general purpose operating systems and implement free encryption and decryption processes through internal interfaces. This means that these cards cannot be exported to certain countries, and anyone who wishes to export, may need to wait several months to obtain the permission from the competent authorities.

Therefore, the structure of the cryptographic functions in Java Card is such that it can be freely used for decrypting the data, but not for encryption. This is sufficient for many applications, and allows many countries to use a simplified export licensing policy.

If a specific application requires data encryption, the manufacturer of the card should incorporate *cryptoEnc.Des3_Enckey* classes and *cryptoEnc.Des_Enckey*, which will enable the process of encryption. But if someone look from a purely cryptographic scope, is definitely possible to design easily implemented procedures which can be freely used for encryption and decryption, without using the above cryptographic classes.

4.7.2.5 Authenticating CAP files.

As we have already seen, applets are loaded on the smart card in the form of CAP files. These CAP files have been created outside of the card using the converter and then loaded on the smart card where and implemented by the Java Card VM. But it should be a way to certify the authenticity of the CAP file that is loaded on the card, as the same card can not verify that the CAP files loaded have a proper format [41].

To achieve this, Java Card technology provides an off card verification mechanism (off card verifier) [42] of the binary code included in the CAP file. This mechanism is implemented by the same converter and uses cryptographic techniques that guarantee the reliability of the CAP's file content. Figure 4-13 shows the files that have as input the off card verifier in order to verify the reliability of a CAP file. These files are the CAP file itself, the specified export file and all packages export files referred by the CAP file. The process of verification of a CAP file includes three control points. Thus, checks verify that the CAP file is:

- Internally consistent
- Consistent with the respective export file
- Consistent with the export files referred

These controls do not need to be performed in the order given.

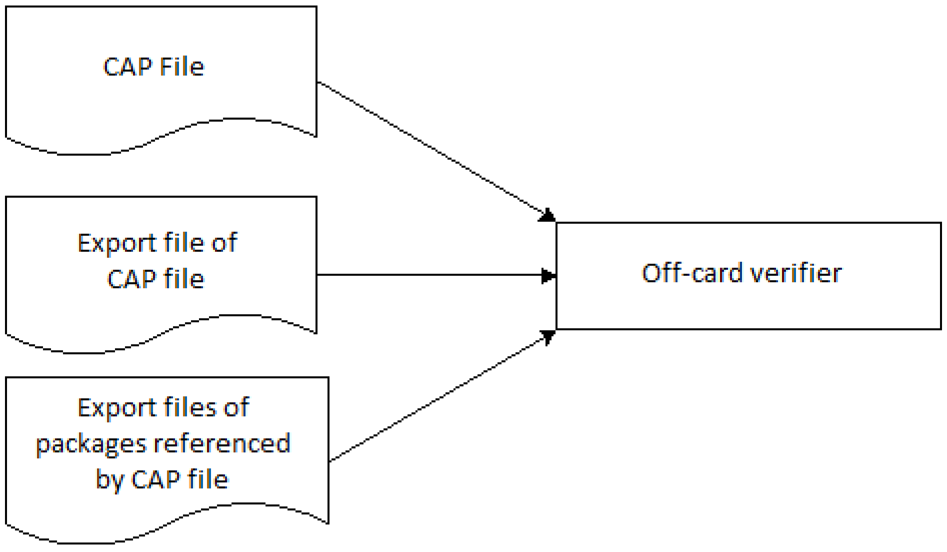


Figure 4-13: Off card verifier

The reason why the verification of the authenticity of a CAP file is made outside of the card is the lack of sufficient memory for implementing this mechanism by the card. The converter authenticates the CAP file and then digitally signs the results. When the code is loaded on the card, all is needed is to check the validity of digital signatures and not the code itself. This mechanism involves verifying the reliability of the digital signature. In case the keys that create and verify the digital signature are lost, the under verification binary code ceases to be reliable.

Chapter 5: Applet: Secure communication between 2 nodes

5.1 Introduction

This chapter presents a description of the applet implemented in the scope of this master thesis for communicating between 2 nodes in a secure way, using a symmetric-key algorithm and specifically the 128bit Advanced Encryption Standard (AES) algorithm for message encryption and 128bit Message Authentication Code (MAC) algorithm for providing integrity and authenticity assurances on the message.

This applet is a part of the European project nShield (mentioned below) and thus the requirements to be met are provided by the technical instructions of the project itself.

Within the following sections an attempt is made to describe the implementation of the applet, the purposes is made to serve and also an example of its use.

5.2 The nSHIELD Project

5.2.1 nSHIELD applet requirements

The nShield system consists of nodes that need to communicate with each other exchanging information required by applications or for administrative purposes. Some of these nodes may not have the capacity in terms of resources to adequately protect the information they handle, since they may not be designed for this purpose. However, the requirement to protect information should be satisfied as nodes may transmit sensitive information or operate in a hostile environment. Smart cards can provide a protected environment for sensitive data such as cryptographic hardware, storage and handling and a means to be able to properly secure the data exchanged. This makes smart cards very attractive solution for the development of powerful applications. Their tamper-resistance property helps cards to withstand physical attacks and in combination with appropriate logical measures, smart cards can guarantee that keys will never leave the card, the information exchanged may be adequately protected from unauthorized disclosure and modification, and the origin of the data can be proven. All these properties prove valuable for the nShield system [50].

5.2.1.1 Starting Point and objectives

In pSHIELD smart cards have been recognized as an important aspect regarding safety of certain critical functions. Their tamper-resistance against physical attacks makes it a strong candidate for the storage and handling of sensitive information. Therefore, they can be used to develop cryptographic protocols, implementation of cryptographic algorithms and managing secret keys embedded in unattended systems operating even in hostile environments.

Smart cards can provide multiple levels of security for data and applications stored in them. For example, the information stored on a smart card is accessible to authenticated users only, can be marked as read-only, can be used only in the smart card or between applications on the card. One of the main advantages of smart cards is that all sensitive functions are managed by the smart card and not the terminal or application, which in many cases are not considered trusted. Smart cards among to others provide the following security services:

1. Message Authentication code
2. Encryption
3. Identity validity
4. Digital signatures
5. Hash functions
6. Secure key management

On top of the above, smartcards can communicate through standardized interfaces, such as USB, or wirelessly, implemented on various platforms. Their small size and low power consumption makes them a strong candidate to deploy as a complementary device to adequately protect nodes and provide security to their stored data and communications.

The objective of this prototype in nShield is therefore to integrate smartcards in the nShield architecture to take advantage of their security characteristics in providing the required SPD functionalities.

5.3 Applet description

5.3.1 Development environment

For the development of the corresponding applet, a virtual machine has been created with Windows XP operating system for making it easy to deliver the entire project without compatibility, installation or configuration issues creating a platform-independent deliverable.

The following software tools have been used:

- Oracle Virtualbox virtualization software package
- Java Development Kit 1.3
- Java Card Development Kit 2.2.1
- Eclipse Galileo
- Eclipse javacard plugin (EclipseJCDE)

As mentioned above these corresponding software tools have been installed and configured in a Virtual Machine (VM) with Windows XP operating system with such a way that are manageable by the EclipseJCDE and as a consequence by Eclipse Galileo, offering a centralized development and administration of the project within the Integrated Development Environment (IDE).

For applet's execution, debugging and resources consumption monitoring, two simulators have been used available from Java Card Development Kit:

- Java Card Workstation Development Environment (JCWDE)
- C-language Java Card RE (CREF)

More information about these simulators is available in section 4.2.3 .

5.3.2 Topology

The nodes topology of the system that the applet is required to provide secure communication is that of a star, as shown in Figure 5-1, in which the master node is located at its center and the individual nodes are directly connected to it. Each node is able to communicate with another node only via the master node, which means that when the A node needs to send a message to B node, then A sends it to the master node, and then the master node forwards the message to B node.

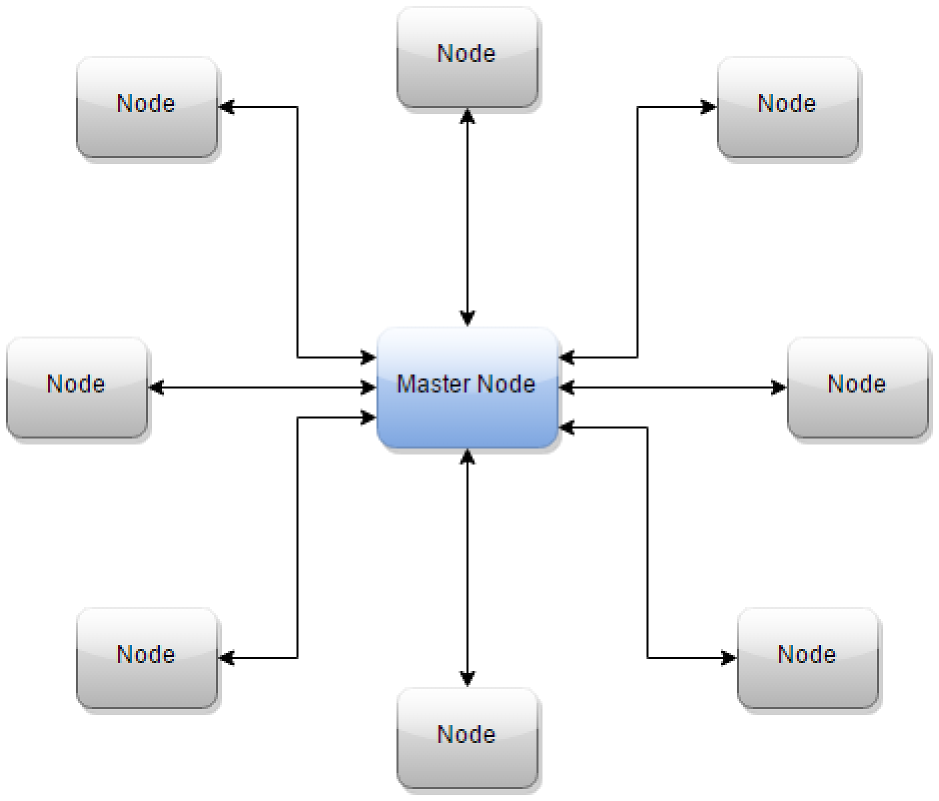


Figure 5-1: Nodes network topology

In the initial phase of operation of this topology, each node has to personalize itself with the master node and vice versa, in order for serial numbers and Node Master Keys (NMKs) to be disclosed and stored by the remote node for the latest to be able to implement data decryption and data verification operations.

The initial operation of personalization should take place in a safe environment since the data that needs to be exchanged are sensitive information and should be transferred through a secure channel. It is nodes and system's operator responsibility to ensure that these safety requirements are met.

In this implementation topology the master node is considered as an embedded system in a safe environment while the dispersed nodes are considered as unattended devices. This implies that the nodes should be protected by other means, considering the fact that they will store sensitive information.

This is the reason why smart cards have been chosen as the media for storing this information and undertake the secure communication of the transmitted data, mainly for their tamper-resistant features and their unauthorized access security, making it useless in case of theft.

5.3.3 Nodes communication

In order to achieve communication between the nodes participating in topology, each node firstly should transmit its message to the master node who then undertakes to contact the recipient node and so the master node always interferes in the communication between different nodes. This approach offers simplicity in explaining the system in which communication between only two nodes need to be described (a node with the master node).

Once the system is in place, certain procedures must be performed to enable communication between two nodes, assuming that the system is just activated the following describes the procedures to be followed by the nodes to be able to communicate.

5.3.3.1 Applet Installation/Personalization

The first thing to be done is the installation of the applet in smart card and personalization of the node with the master node and vice versa. In this stage an appropriate APDU command will provide to the applet the required information characteristics of the remote node such as the Serial Number and the Node Master Keys which will be used for message decryption and signature verification.

This stage is taking place with the initial installation of the applet and therefore the constructor is called to allocate all needed memory for the lifetime of the applet and store the remote node's information in EEPROM memory. Consequently this operation can only be done once for this instance of the applet.

For the node to be informed of the distant node's information, the latest should transmit this sensitive information over the media. This information contains sensitive security keys and therefore the communication channel and the physical access to the nodes of the topology during this stage should be protected. After this one time operation the master keys will never leave the card again in the applet's lifetime.

At the end of this stage each node will have the distant node master keys and a serial number serving as a node identification number to correlate the keys with.

5.3.3.2 New encrypt and decrypt sessions

Once the installation and personalization stage is completed a node or the master node has in its possession the master keys of the node that wants to communicate. These keys will not directly be used for cryptographic operation but instead for every communication session, new key pairs are created for each node.

The node that needs to transmit a message, has to create a new encrypt session producing a new encrypt session key pair and the recipient node will create a new decrypt session producing a new decrypt session key pair.

These sessions are described bellow in more detail:

A new encrypt session is requested from the applet for transferring a message. The application creates a new session key pair based on the master keys, from these two keys; one will be used for encrypting the message and the other for authenticating it. For generating the key pair a random number is used produced at that time and is returned to the node for sending it to the remote node for producing the same key pair for decryption.

A new decrypt session may be requested from the applet when a random number created by a remote node during it's encrypt session is available to decrypt that node's message. The random number is provided to the applet as an APDU command parameter and a new decrypt session is initiated creating the same session keys used by the remote node to encrypt and authenticate the message.

5.3.3.3 Encrypt and/or authenticate a message

When an encryption session is active for a node, it is possible for that node to encrypt and/or authenticate a message and transmit it securely to a remote node.

The message to be sent is prior feed to the applet as an APDU command with an option of the type of security that needs to be applied to the message; there are two methods for treating the message as shown below:

- The more secure method produces an authentication code of the message and this code along with the message is encrypted producing the secured message to be sent, the structure of the message returned with this method is shown in Figure 5-2.
- The second method is less secure since it does not encrypt the message but appends to it its authentication code for message and data origin authentication, the structure of the message produced by this method is shown in Figure 5-3.

The treated message is then returned as an APDU command back to the node which may send it to the remote node to decrypt it and/or verify it.

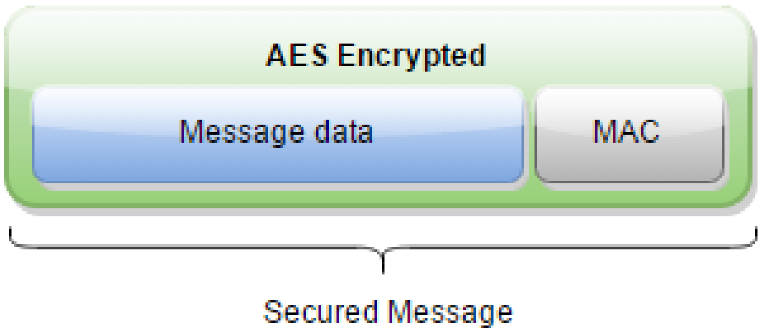


Figure 5-2: Structure of encrypted message with authentication code

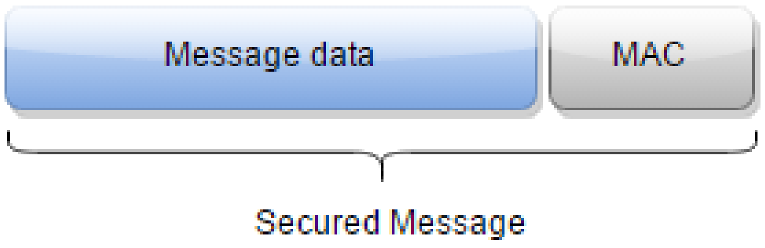


Figure 5-3: Structure of message with authentication code

5.3.3.4 Decrypt and/or verify a message

Until now, both nodes have been personalized and have access to each other keys. Also new sessions have been initiated, an encrypt session for the sender node and a decrypt session for the receiver node thus creating two identical pair of session keys making a node able to decrypt the other's encrypted message.

At this phase an encrypted and/or authenticated message has arrived to the node which will be forward as an APDU command parameter to the applet for handling.

Since the message may have two types of format (authentication code and encrypted or only authentication code) the node forwards the message to the applet with the appropriate APDU option for decrypting and verifying the message or just verifying it.

In the case where the message contains an authentication code and is encrypted, the applet will decrypt the message, will produce its own authentication code using the plain message and it will compare it with the authentication code received, if these two authentication codes are identical, the applet will return the plain message back to the node with a status code of success, otherwise the applet will return only an error code indicating that the message's data origin authentication and/or data integrity is corrupted.

On the other hand, if the message arrives unencrypted with an authentication code appended, the applet will only calculate the authentication code of the message and as above it will compare it with the authentication code received, if these is a match, a success status code is returned, otherwise an error code is returned indicating that the message’s data origin authentication and/or data integrity is corrupted.

5.3.4 Type of security implemented

In terms of security two main algorithms have been used, the first and most important is the 128 bit AES symmetric encryption algorithm used to encrypt the data in order to be transmitted over an unsecure media. The main advantage of AES and thus symmetric cryptography is the low requirements in terms of processing power, a very important factor concerning implementations in smart card technology.

```
// session encryption key declaration as AES 128bit Key
static AESKey      SESSION_ENCR_KEY = (AESKey) KeyBuilder.buildKey
            (KeyBuilder.TYPE_AES_TRANSIENT_RESET, KeyBuilder.LENGTH_AES_128, false);


// create cipher for 128bit AES
cipher    =          Cipher.getInstance(Cipher.ALG_AES_BLOCK_128_CBC_NOPAD, false);


// we create NMEK(Rn), use Rn as help array
cipher.init(NODE_MASTER_ENCR_KEY, Cipher.MODE_ENCRYPT);
cipher.doFinal(buffer, (short) 0, RANDOM_NUM_SIZE, Rn, (short) 0);


// SEK is produced using the NMEK(Rn)
SESSION_ENCR_KEY.setKey(Rn, (short) 0);
```

Figure 5-4: Example of Session Encryption Key creation

Figure 5-4 shows the in-card creation of the Session Encryption Key.

Apart from message encryption the application offers message integrity and origin authentication, for implementing these features the MAC algorithm has been used, hashing the message and exposing any corruption of it.

```
// declare a new signature instance
authCBCMAC = Signature.getInstance(Signature.ALG_AES_MAC_128_NOPAD, false);

// initialize MAC with SAK
authCBCMAC.init(SESSION_AUTH_KEY, Signature.MODE_SIGN);

// create MAC of data and store in buffer
authCBCMAC.sign(buffer, (short)0, dataSize, buffer, (short) dataSize);
```

Figure 5-5: Example of signing data with MAC algorithm

Figure 5-5 show a snippet of applet's code regarding the declaration of a new signature instance using it for data signing.

Furthermore, all sensitive variables such as keys or keys manufacturing components are stored in appropriate variable types that Java Card's API provides according to the specifications of Java Card [51], ensuring protection for most types of attacks. For these same sensitive information their lifetime is a very important factor, therefore only the keys required in applet's lifetime are stored in EEPROM memory, all other sensitive data, such as session keys and random numbers are stored in transient type arrays with a memory location pointer only stored in EEPROM and are marked for deletion in smart card's reset (power loss) or in applet's deselection by the deselect method of the applet.

```
// sensitive information will be cleared in case of reset
Random    =    JCSystem.makeTransientByteArray( (short)16, JCSystem.CLEAR_ON_RESET);
SESSION_ENCR_KEY   =   (AESKey) KeyBuilder.buildKey
          (KeyBuilder.TYPE_AES_TRANSIENT_RESET, KeyBuilder.LENGTH_AES_128, false);

// on deselect this applet, session keys will clear
// and their initialization condition return to false (by clearing the keys)
deselect() {
          Random = new byte[16];
          SESSION_ENCR_KEY.clearKey();
}
```

Figure 5-6: Proper disposal of temporary sensitive data

Figure 5-6 shows an example of how applet manages the proper disposal of temporary sensitive data.

5.3.5 Code efficiency

Efficiency is a critical part in smartcards applications considering the fact that a smartcard's applet has to operate by an environment with limited Random Access Memory (RAM) available considering the fact that the majority of smartcards today available typically ranges from 2 kB to 4 kB prohibiting the use of indicator variables created for a specific in code cause, forcing the re-use of variables and placing the creation of a new variable under good consideration. This is not the only limiting factor an applet has to face, the non-volatile memory EEPROM although it has adequate capacity, its NAND gates is made of, offer finite cycles of write/erase, typically around 10,000 times before it wear out, requiring careful planning of variables use, finally, a smart card contains a low frequency microcontroller adding delay to runtime when complex code or mathematical functions (cryptographic operations) are implemented.

Therefore designing an applet should be made in such a way that it minimizes the use of variables at runtime by creating general purpose public scope variables that are reused throughout the applet's lifetime, use of EEPROM memory should be done only in cases where there is a need for maintaining the variables permanently, both for the protection of memory from wearing out and the low access speed. Also, the use of 'expensive' programming routines should be limited to ensure that the speed of the applet will run at acceptable levels.

5.3.5.1 Memory efficiency

The memory required for the functionality of this applet is allocated at the initial installation phase in smart card eliminating the risk of running out of available memory in the runtime.

Variables which their data are likely to alter during the use of the applet are stored in the RAM memory and a pointer variable is stored in memory EEPROM, thus ensuring rapid access to the variable, the unlimited reuse and the ability that the variable will be accessible and is not lost that range of memory when closing the applet.

For general purpose variables and need of temporary storage for applet data, the APDU buffer is used which by default is an array and if defined, allocates 128 bytes of RAM memory.

Thereby the applet can operate with only 1.151 Bytes of available EEPROM memory for applet upload, installation and persistent variables and 253 Bytes of available RAM for transient objects.

5.3.5.2 Runtime efficiency

For applet’s functionality the methods and variables used are provided directly from javacard API for optimal execution speed of the code and for secure storing of the keys or the production components of those on appropriate variables ensuring their safe storage and providing protection from attacks such as power analysis attack.

5.3.6 Methods

Below are the methods implemented in the applet along with the possible values of the command APDU header and description of their function.

Method	CLA	INS	P1	P2	Description
Installation & Personalize	0x80	0xB8	0x00	0x00	Install applet and personalize node with the master node
encrypt	0x80	0x76	0x02	0x00	Creates new encrypt session
>>	0x80	0x76	0x04	0x00	Encrypt and Sign data
>>	0x80	0x76	0x06	0x00	Sign data
decrypt	0x80	0x78	0x02	0x00	Creates new decrypt session
>>	0x80	0x78	0x04	0x00	Verify data

>>	0x80	0x78	0x06	0x00	Decrypt and verify data
getRandom	0x80	0x80	0x00	0x00	Returns 16 Byte random number

5.3.6.1 Install / Personalize method

It is called only once, during the initial installation of the card in the node and personalizes the node with the remote master node.

The install method calls the constructor of the applet that allocates all needed memory required for the entire life of the applet in order to avoid running out of memory at run time

This initial one time only installation of the card the two nodes has to exchange their node keys in order to be able each one to produce the remotes node session keys, this exchange requires sensitive data (including keys) to be transmitted over the network (SN, NMEK and NMAK) and must be used in a safe environment. It is node's responsibility to create a secure channel for the installation phase.

APDU Parameters:

- AID of the applet to install.
- Master's node (remote node) Serial Number (R_SN),
- Master's node (remote node) Node Master Encryption Key (R_NMEK) and
- Master's node (remote node) Node Master Authentication Key (R_NMAK).

The APDU command is acceptable in the form (AID || R_SN || R_NMEK || R_NMAK).

R_SN and keys received are stored in EEPROM for the entire life of the applet using specialized variables of javacard for storing keys, the Serial Number is considered an ID for the remote node and keys are used for creating new decrypt sessions whenever the node receives data from the master node and a decryption needs to take place.

Serial Number could be any byte stream ranging from 0 to 16 Bytes and the applet prior to storing in the EEPROM will pad it to the left with zeroes to reach the 16 Bytes size.

Returns:

- SW: 9000 if personalization is completed
- SW: 6700 (SW_WRONG_LENGTH) if R_NMEK or R_NMAK are not a 16Byte array corresponding a 128 bit AES key.

5.3.6.2 Encrypt method

Called whenever a node needs to send data to another node.

There are 3 supported P1 values as described below:

- P1 = 0x02: **NEW_ENCRYPT_SESSION**

1. Creates a new encrypt session following these steps.
2. Creates a new random number (Rn) and stores it in RAM.
3. Encrypts Rn using NMEK key “NMEK(Rn)”.
4. Stores NMEK(Rn) as Session Ecryption Key (SEK) (SEK = NMEK(Rn))
5. Encrypts Rn using NMAK key “NMAK(Rn)”.
6. Stores NMAK(Rn) as Session Authentication Key (SAK) (SAK = NMAK(Rn))
7. If the operation succeeded returns Rn which it will be used from the remote node to produce the same session keys for decryption of messages for this session.

Session keys stay in the card and are destroyed after each session, deselect and reset (power failure or card removal).

- P1 = 0x04: **ENCRYPT _MSG**

  Encrypts and signs the data obtained through APDU commands using the session keys created from NEW_ENCRYPT_SESSION and

  Returns:

  - { AESsek (Data || MACsak(Data))} and SW: 9000 if successful
  - SW: 6305 (SW_NO_SESSION_ACTIVE) if session keys are not initialized which means there is not an active encrypt session

- P1 = 0x06: **GENERATE_MAC**

  Sign the incoming data using MAC algorithm.

  Returns:

  - { Data || MACsak(Data)} and SW: 9000 if successful
  - SW: 6305 (SW_NO_SESSION_ACTIVE if session keys are not initialized which means there is not an active encrypt session.

5.3.6.3 Decrypt method

Called whenever the node needs to decrypt data received from another node.

The supported P1 values are 3 and described below:

- P1 = 0x02: **NEW_DECRYPT_SESSION**

  If a node creates a new encryption session, card returns Rn to the node which sends it to the remote node to be used for a new decrypt session in order to be able to produce the same Session Keys.

  This method is called with an APDU command that contains the Rn of the remote node for creating the same Session Keys with the specified node.

Rn received but also both session keys (encrypt and authenticate) created are stored in appropriate variables in RAM for fast accessing and security and erased in the end of every session, card reset (power failure or card removal) or applet deselect.

- P1 = 0x04: **DECRYPT_MSG**

  Decrypts and authenticates the data received.

  This method accepts APDU commands in the form {AESsek (Data || MACsak(Data))}.

  Returns:

  - Plain Data and SW: 9000 if MAC is verified.
  - SW: 6304 (SW_MAC_VERIFICATON_FAILED) if MAC verification failed.
  - SW: 6306 (SW_INVALID_SESSION) if remote node session keys are not initialized which means there is not an active decrypt session.

- P1 = 0x06: **VERIFY_MAC**

  Verifies data received for message origin authentication and data integrity.

  This method accepts APDU commands in the form {Data || MACsak(Data)}

  Returns:

  - SW: 9000 if MAC verification passed.
  - SW: 6304 (SW_MAC_VERIFICATON_FAILED) if MAC verification failed.
  - SW: 6306 (SW_INVALID_SESSION) if remote node session keys are not initialized which means there is not an active decrypt session.

5.3.6.4 GetRandom method

This method is used by the node when there is a need of a secure random number, for example when a large amount of data needs to be exchanged and the smartcard's physical characteristics do not allow their fast encryption/decryption and transmission.

Returns:

- A 16Byte random number.

5.3.7 Scenario of use

In this section a scenario of use of the specified applet will be provided, applying both encryption/signing and decryption/ verification in order to cover the greater part of this applet's functionality. In order to use these functions the use of *Installation/Personalization, NewEncryptSession and NewDecryptSession* should also be used to prepare the applet and instantiate the keys for encryption and/or decryption.

The specified scenario has been implemented in the environment shown in chapter Development environment, Figure 5-7 graphically visualizes the actions taken place.

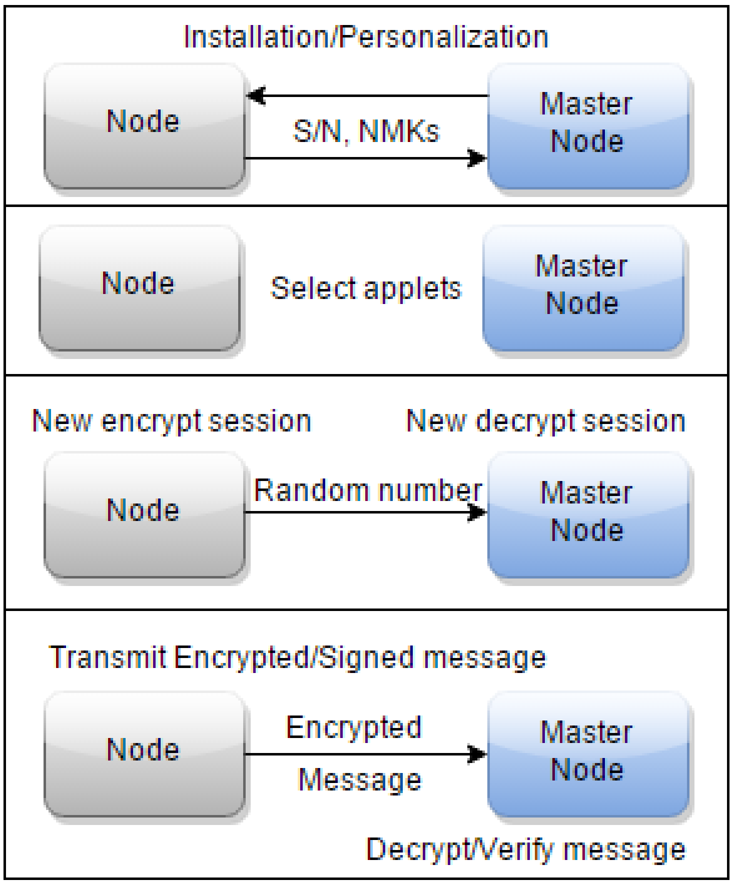


Figure 5-7: Scenario implemented

5.3.7.1 Installing applets and personalizing nodes

Assuming that this is the first time the smart cards are used by the topology both for master node and the node, the Installation/Personalization phase should take place, each node has a pair of keys called Node System Keys (NSKs) for encryption and for authentication stored in its Hardware Security Module (HSM), these keys should never leave the node, so the node creates another pair keys called Node Master Keys (NMKs) using the equation {Node Master Key (NMK) = AES_{SMK} (Node's Serial Number)} and provides them to the card where are store for the lifetime of the applet. These keys are used by the applet for performing cryptographic operations as shown bellow in this section.

Assuming that the Node Master Keys provided to the applet from the node that uses this smart card which for convenience from now on we will refer as Node Master Encryption Key (NMEK) and Node Master Authentication Key (NMAK) are the following:

Node keys:

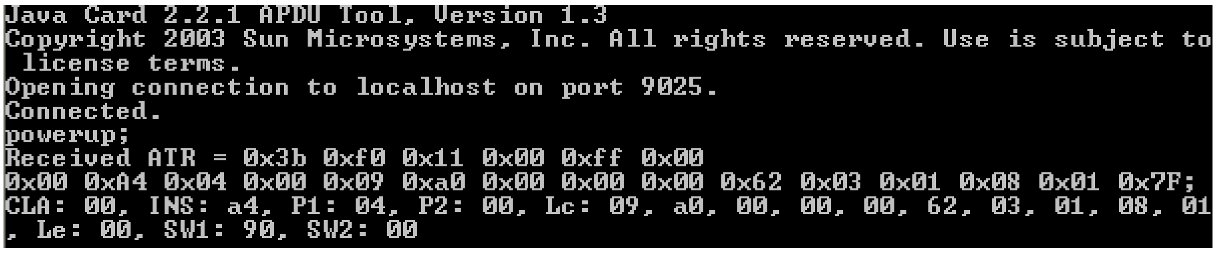
- M_SN = 0x020304
- NMEK = 0x0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a0a
- NMAK = 0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b

Master Node Keys:

- M_SN = 0x050607
- M_NMEK = 0x35db086881c13c700362c801d1031566
- M_NMAK = 0x389478b07dee47d7d7d2d17e507638b6

During the Installation phase of the applet, each node should provide its own Serial Number and Node Master Keys to the remote node in order to be stored permanently inside the non-volatile memory of the card. This operation should take place under a controlled and safe environment (for example via a secure tunnel created between these two nodes) and is only necessary to be made once for each node (at the Installation phase).

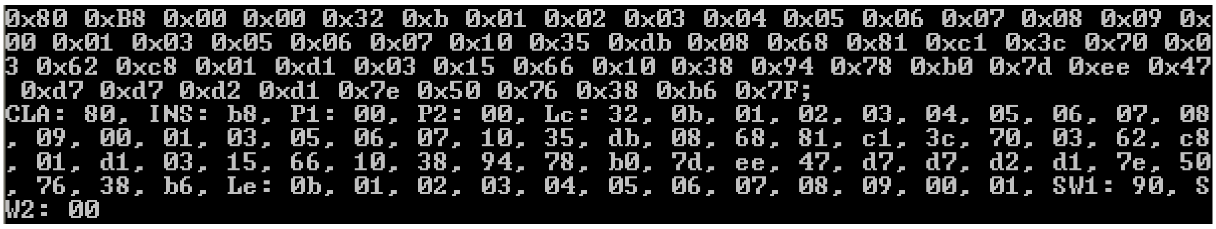
The node should issue a select APDU command to the JCVM providing the AID of the applet that wants to select, the applet that should be selected at this moment is the installer applet with
AID = 0x09 a0 00 00 00 62 03 01 08 01 7F for both node and master node.



```
Java Card 2.2.1 APDU Tool, Version 1.3
Copyright 2003 Sun Microsystems, Inc. All rights reserved. Use is subject to
 license terms.
Opening connection to localhost on port 9025.
Connected.
powerup;
Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x00
0x00 0xA4 0x04 0x00 0x09 0xa0 0x00 0x00 0x00 0x62 0x03 0x01 0x08 0x01 0x7F;
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 09, a0, 00, 00, 00, 62, 03, 01, 08, 01
, Le: 00, SW1: 90, SW2: 00
```

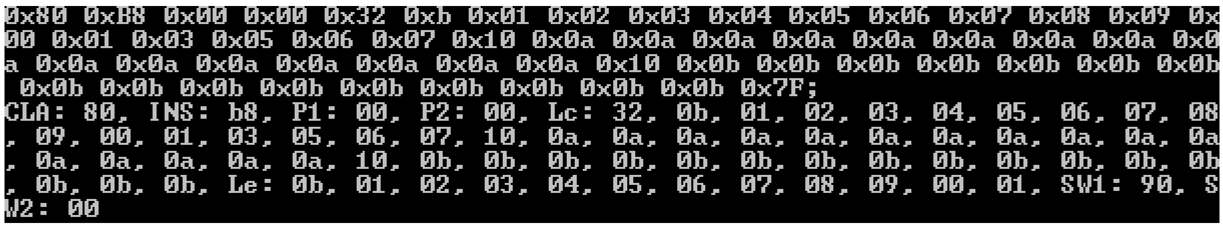
Figure 5-8: Selecting installation applet

As we see in figure Figure 5-8, apdutool is connected to CREF simulator on port 9025 and issues the *powerup* command receiving the ATR (explained in Transferring data to smart cards section) response, the *select* command may be seen bellow the ATR response where the first 4 bytes indicate the type of APDU command in this case a *select* command with CLA = 0x00, INS = 0xA4, P1 = 0x04 and P2 = 0x00 and the following bytes indicate the AID of the applet to select (the last byte, 0x7F mark the end of the command). The simulator response with an echo appending the response which in this case is SW1: 90, SW2: 00 meaning that the command was accepted and the applet is now selected.
The next step that of Installation/Personalization phase and we need to provide to the installation applet the parameters for the applet we want to install and personalize with a node. The following installation APDU command will be in the form {0x80, 0xB8, 0x00, 0x00, Command Length, AID Length, AID, S/N Length, S/N, NMEK Length, NMEK, NMAK Length, NMAK, 0x7F}, the AID of this thesis applet is AID = 0x012345678901, the other values where shown above.



```
0x80 0xB8 0x00 0x00 0x32 0xb 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x
00 0x01 0x03 0x05 0x06 0x07 0x10 0x35 0xdb 0x08 0x68 0x81 0xc1 0x3c 0x70 0x0
3 0x62 0xc8 0x01 0xd1 0x03 0x15 0x66 0x10 0x38 0x94 0x78 0xb0 0x7d 0xee 0x47
 0xd7 0xd7 0xd2 0xd1 0x7e 0x50 0x76 0x38 0xb6 0x7F;
CLA: 80, INS: b8, P1: 00, P2: 00, Lc: 32, 0b, 01, 02, 03, 04, 05, 06, 07, 08
, 09, 00, 01, 03, 05, 06, 07, 10, 35, db, 08, 68, 81, c1, 3c, 70, 03, 62, c8
, 01, d1, 03, 15, 66, 10, 38, 94, 78, b0, 7d, ee, 47, d7, d7, d2, d1, 7e, 50
, 76, 38, b6, Le: 0b, 01, 02, 03, 04, 05, 06, 07, 08, 09, 00, 01, SW1: 90, S
W2: 00
```

Figure 5-9: Installation parameters for node



```
0x80 0xB8 0x00 0x00 0x32 0xb 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x
00 0x01 0x03 0x05 0x06 0x07 0x10 0x0a 0x0a 0x0a 0x0a 0x0a 0x0a 0x0a 0x0a 0x0
a 0x0a 0x0a 0x0a 0x0a 0x0a 0x0a 0x0a 0x10 0x0b 0x0b 0x0b 0x0b 0x0b 0x0b 0x0b
 0x0b 0x0b 0x0b 0x0b 0x0b 0x0b 0x0b 0x0b 0x0b 0x7F;
CLA: 80, INS: b8, P1: 00, P2: 00, Lc: 32, 0b, 01, 02, 03, 04, 05, 06, 07, 08
, 09, 00, 01, 03, 05, 06, 07, 10, 0a, 0a, 0a, 0a, 0a, 0a, 0a, 0a, 0a, 0a, 0a
, 0a, 0a, 0a, 0a, 0a, 10, 0b, 0b, 0b, 0b, 0b, 0b, 0b, 0b, 0b, 0b, 0b, 0b, 0b
, 0b, 0b, 0b, Le: 0b, 01, 02, 03, 04, 05, 06, 07, 08, 09, 00, 01, SW1: 90, S
W2: 00
```

Figure 5-10: Installation parameters for master node

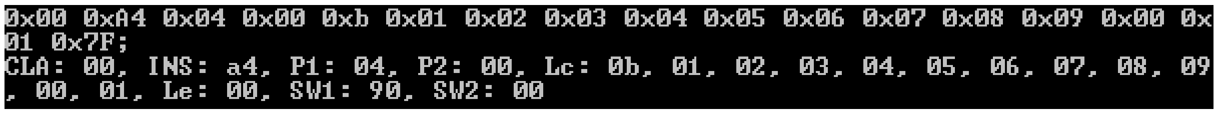
Figure 5-9 and Figure 5-10 shows the installation procedure for node and master node respectively where we may observe two blocks of APDU commands in each figure, the upper one is the command sent to the applet and the other is the response.

The installation applet replies with an echo appending its answer at the end of the command which in this case is the AID of the applet installed and SW1: 90 and SW: 00 meaning that no errors occurred and the applet is now installed and each node is personalized with its remote node.

5.3.7.2 ***Selecting the applets***

At this moment the JCVM has selected the installer applet, in order to continue with our scenario we should select our applet. The procedure is the same as selecting the installer applet with the only difference that we are using the AID of our applet.

For both node and master node the parameters and response are the same, thus only one image is provided.



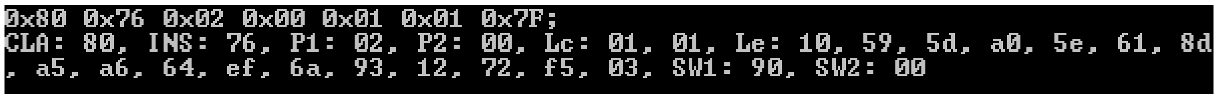
```
0x00 0xA4 0x04 0x00 0xb 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x
01 0x7F;
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 0b, 01, 02, 03, 04, 05, 06, 07, 08, 09
, 00, 01, Le: 00, SW1: 90, SW2: 00
```

Figure 5-11: Selecting applet

Figure 5-11 shows that the CREF simulator returns a SW: 0x9000 response, indicating no errors and a successful selection of the applet for both nodes.

5.3.7.3 New encryption and decryption sessions

In order to be able to use the cryptographic methods of this applet new encrypt session for the node and a new decrypt session for the master node should be created.



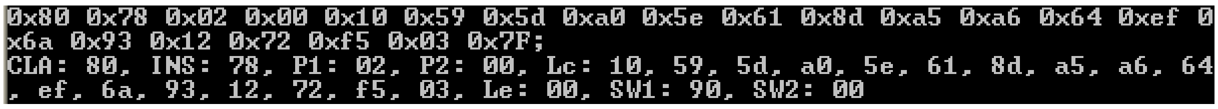
```
0x80 0x76 0x02 0x00 0x01 0x01 0x7F;
CLA: 80, INS: 76, P1: 02, P2: 00, Lc: 01, 01, Le: 10, 59, 5d, a0, 5e, 61, 8d
, a5, a6, 64, ef, 6a, 93, 12, 72, f5, 03, SW1: 90, SW2: 00
```

Figure 5-12: New encrypt session for node

In Figure 5-12 we may observe the first 4 bytes of the encrypt APDU command following with the Le field (number of data bytes returned) and 1 data byte with value 0x01 (it is possible to send commands with no payload, in this example is not possible due to a bug in the simulator).

The applet returned a 16 byte random number that it used to create the session keys in-card to be used for encryption. The purpose of returning this random number is mainly for sending it to the master node to create the same session keys for decrypting the message, also can be used for giving the capability to the node to make its own session keys and encrypt the data in situations where large amount of data should be transmitted in short time, also returns SW: 0x9000 indicating no errors in this phase.
If the new encrypt session is not called before an encryption or signing of data is attempted a SW: 0x6303 meaning there is not an active session will be returned.

After the node has transmitted the random number to the master node the last is able to create a new decrypt session passing as APDU parameter the random number and receive as response an echo of the random number an SW = 0x9000 if no error occurred as shown in Figure 5-13.

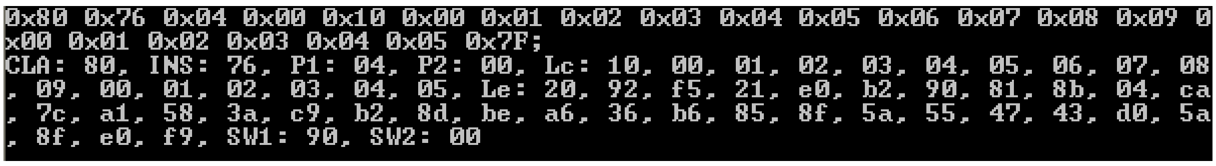


```
0x80 0x78 0x02 0x00 0x10 0x59 0x5d 0xa0 0x5e 0x61 0x8d 0xa5 0xa6 0x64 0xef 0
x6a 0x93 0x12 0x72 0xf5 0x03 0x7F;
CLA: 80, INS: 78, P1: 02, P2: 00, Lc: 10, 59, 5d, a0, 5e, 61, 8d, a5, a6, 64
, ef, 6a, 93, 12, 72, f5, 03, Le: 00, SW1: 90, SW2: 00
```

Figure 5-13: New decrypt session for master node

5.3.7.4 Encrypt/sign and decrypt/verify data

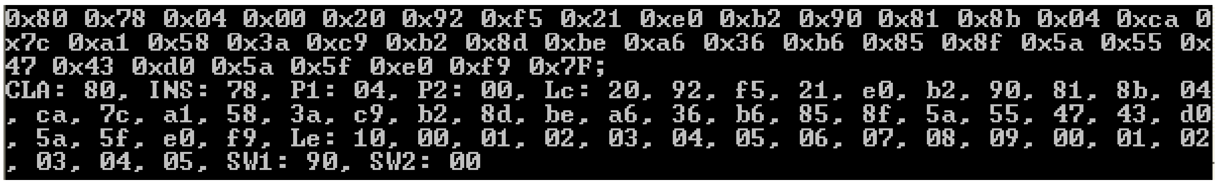
Having active encrypt and decrypt sessions it is possible to encrypt data in-card in node's side and decrypt those data in master node's side, as shown in section Encrypt method the APDU command syntax for calling encrypt method is {0x80, 0x76, 0x04, 0x00, DataLength, data for encryption and signing}.
The data sent in this scenario for encryption and signing are: 0x0123456789012345.



```
0x80 0x76 0x04 0x00 0x10 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0
x00 0x01 0x02 0x03 0x04 0x05 0x7F;
CLA: 80, INS: 76, P1: 04, P2: 00, Lc: 10, 00, 01, 02, 03, 04, 05, 06, 07, 08
, 09, 00, 01, 02, 03, 04, 05, Le: 20, 92, f5, 21, e0, b2, 90, 81, 8b, 04, ca
, 7c, a1, 58, 3a, c9, b2, 8d, be, a6, 36, b6, 85, 8f, 5a, 55, 47, 43, d0, 5a
, 8f, e0, f9, SW1: 90, SW2: 00
```

Figure 5-14: Data encryption and signing for node

As shown in Figure 5-14 the applet responds with 32 bytes of data (Le = 0x20), this is normal considering that the expected output is in the form {AESsek (Data || MACsak(Data))}, both data and MAC(Data) are 16 bytes long, therefore this 32 byte response is the message and its authentication code both encrypted with AES algorithm.
This byte string can be securely transmitted over an unsecure medium to the master node for decryption and verification as shown in the next figure.



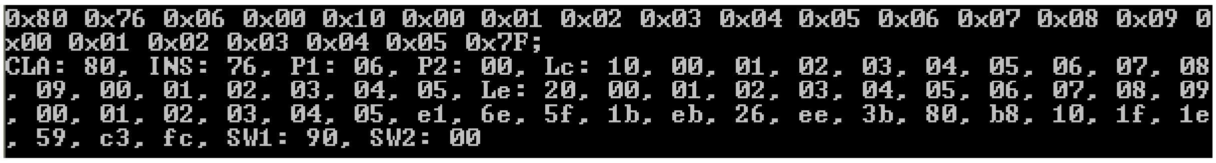
```
0x80 0x78 0x04 0x00 0x20 0x92 0xf5 0x21 0xe0 0xb2 0x90 0x81 0x8b 0x04 0xca 0
x7c 0xa1 0x58 0x3a 0xc9 0xb2 0x8d 0xbe 0xa6 0x36 0xb6 0x85 0x8f 0x5a 0x55 0x
47 0x43 0xd0 0x5a 0x5f 0xe0 0xf9 0x7F;
CLA: 80, INS: 78, P1: 04, P2: 00, Lc: 20, 92, f5, 21, e0, b2, 90, 81, 8b, 04
, ca, 7c, a1, 58, 3a, c9, b2, 8d, be, a6, 36, b6, 85, 8f, 5a, 55, 47, 43, d0
, 5a, 5f, e0, f9, Le: 10, 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 00, 01, 02
, 03, 04, 05, SW1: 90, SW2: 00
```

Figure 5-15: Data decryption and verification for master node

For decryption and verification of the encrypted data produced in the node's side the decrypt method should be called in the master node's side with the APDU command {0x80, 0x78, 0x04, 0x00, DataLength, data for decryption and verification}, the data should be the exact output of the node's encrypt method call.
As shown in Figure 5-15 the encrypted data are passed as parameters in the encrypt method call and the output of this method is the plain initial data and SW = 0x9000 indicating that the data are not only decrypted successfully but verified too.

5.3.7.5 Sign and verify data

If encryption is not necessary yet integrity protection remains a strong requirement, a node may choose only to sign the data which intends to transmit giving as APDU parameters to the card the data and get as result the data and the signature of the data at the tail {Data || MAC(Data)}.
The data for signing will be the same as before, data = 0x0123456789012345.

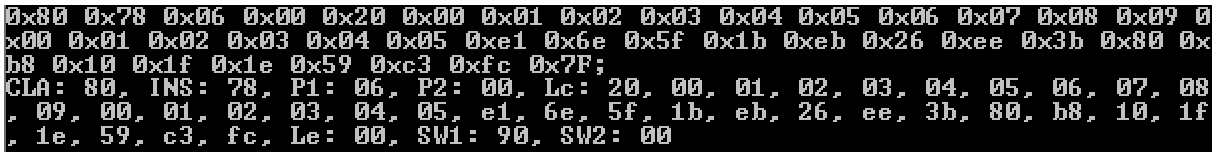


```
0x80 0x76 0x06 0x00 0x10 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0
x00 0x01 0x02 0x03 0x04 0x05 0x7F;
CLA: 80, INS: 76, P1: 06, P2: 00, Lc: 10, 00, 01, 02, 03, 04, 05, 06, 07, 08
, 09, 00, 01, 02, 03, 04, 05, Le: 20, 00, 01, 02, 03, 04, 05, 06, 07, 08, 09
, 00, 01, 02, 03, 04, 05, e1, 6e, 5f, 1b, eb, 26, ee, 3b, 80, b8, 10, 1f, 1e
, 59, c3, fc, SW1: 90, SW2: 00
```

Figure 5-16: Data signing for node

Figure 5-16 shows the data sign command and the response of the data and the signature.

The output data and signature of the data can be transmitted to the master node for verification.



```
0x80 0x78 0x06 0x00 0x20 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0
x00 0x01 0x02 0x03 0x04 0x05 0xe1 0x6e 0x5f 0x1b 0xeb 0x26 0xee 0x3b 0x80 0x
b8 0x10 0x1f 0x1e 0x59 0xc3 0xfc 0x7F;
CLA: 80, INS: 78, P1: 06, P2: 00, Lc: 20, 00, 01, 02, 03, 04, 05, 06, 07, 08
, 09, 00, 01, 02, 03, 04, 05, e1, 6e, 5f, 1b, eb, 26, ee, 3b, 80, b8, 10, 1f
, 1e, 59, c3, fc, Le: 00, SW1: 90, SW2: 00
```

Figure 5-17: Data verification for master node

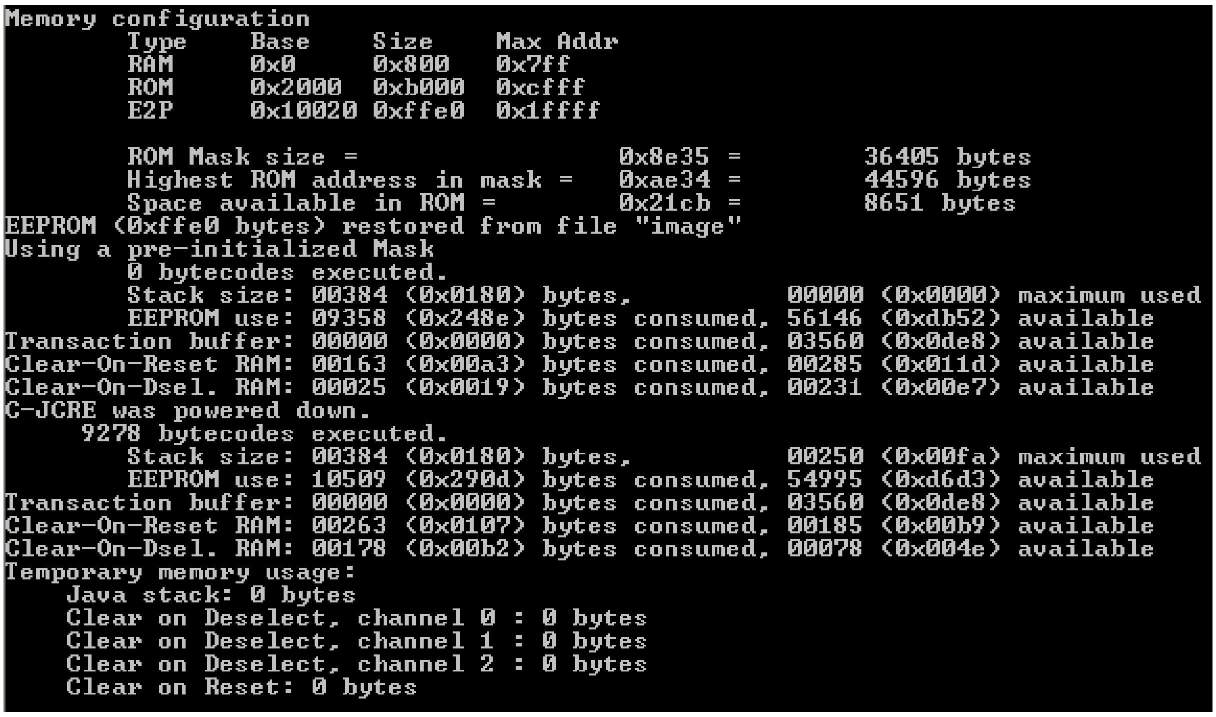
Figure 5-17 shows the procedure for data verification in the master node's side.
The applet responds to the master node with SW = 0x9000 indicating that the data have been verified.

5.3.7.6 Resources consumption

The CREF simulator used for the implementation of this scenario has the following resources:

- RAM = 0x800 Bytes = 2048 Bytes
- ROM = 0xb000 Bytes = 45056 Bytes
- EEPROM = 0x10020 Bytes = 65504 Bytes

In Figure 5-18 we may see the CREF simulator reporting its available resources for uploading and executing applets at the top of the figure as well as the resources consumed by the applet during runtime.



```
Memory configuration
        Type     Base     Size     Max Addr
        RAM      0x0      0x800    0x7ff
        ROM      0x2000   0xb000   0xcfff
        E2P      0x10020  0xffe0   0x1ffff

        ROM Mask size =                0x8e35 =        36405 bytes
        Highest ROM address in mask =  0xae34 =        44596 bytes
        Space available in ROM =       0x21cb =        8651 bytes
EEPROM (0xffe0 bytes) restored from file "image"
Using a pre-initialized Mask
        0 bytecodes executed.
        Stack size: 00384 (0x0180) bytes,          00000 (0x0000) maximum used
        EEPROM use: 09358 (0x248e) bytes consumed, 56146 (0xdb52) available
Transaction buffer: 00000 (0x0000) bytes consumed, 03560 (0x0de8) available
Clear-On-Reset RAM: 00163 (0x00a3) bytes consumed, 00285 (0x011d) available
Clear-On-Dsel. RAM: 00025 (0x0019) bytes consumed, 00231 (0x00e7) available
C-JCRE was powered down.
     9278 bytecodes executed.
        Stack size: 00384 (0x0180) bytes,          00250 (0x00fa) maximum used
        EEPROM use: 10509 (0x290d) bytes consumed, 54995 (0xd6d3) available
Transaction buffer: 00000 (0x0000) bytes consumed, 03560 (0x0de8) available
Clear-On-Reset RAM: 00263 (0x0107) bytes consumed, 00185 (0x00b9) available
Clear-On-Dsel. RAM: 00178 (0x00b2) bytes consumed, 00078 (0x004e) available
Temporary memory usage:
    Java stack: 0 bytes
    Clear on Deselect, channel 0 : 0 bytes
    Clear on Deselect, channel 1 : 0 bytes
    Clear on Deselect, channel 2 : 0 bytes
    Clear on Reset: 0 bytes
```

Figure 5-18: Applet's resources consumption

The 2 consumption reports shown are the first for executing the CREF simulator and the last for terminating it. The commands issued to the CREF between starting and terminating it was:

1. Calling the installer applet
2. Install the project applet
3. Select the project applet
4. Call a new encrypt session
5. Encrypt data
6. Terminate CREF

In order to calculate the resources consumption of this applet we have subtracted the bytes referred by CREF at its termination report with those from its starting report.

The results obtained are the consumptions statistics of this project applet as shown below:

- Clear-On-Reset RAM: 263 – 163 = 100 Bytes
- Clear-On-Reset RAM: 178 – 25 = 153 Bytes
- Stack maximum size: 250 Bytes
- EEPROM used: 10509 – 9358 = 1151 Bytes

Chapter 6: Conclusions

In this document we have described the basic rules that define a smart card along with their advantages and the security benefits these provide especially when used with the java card technology, also we have shown an example scenario implemented with an applet created in the context of this master thesis, providing secure communication between two nodes using symmetric cryptography for message encryption and message authentication code for message authentication and data integrity.

6.1 Future work

This applet can provide secure communication between two nodes in a star topology where each node in order to communicate with a remote node, it is a prerequisite to communicate with the master node and the last will transfer the data to the remote node.

The node at this stage is able to personalize and thus communicate with only one node, an implementation where the master node could be able to personalize with each node in the topology and manage their master node keys would be a great advance.

Currently the java card version used is version 2.2.1, while there are more recent versions of java card (2.2.2 and 3), the 2.2.1 version was adopted for increased compatibility with the available smart cards in the market today. This choice comes with a drawback. The 2.2.1 version does not have support for the key-hashed message authentication code (HMAC), an algorithm superior to the MAC algorithm currently used. This drawback may be resolved with the adoption of 2.2.2 version of javacard or a custom implementation of the HMAC algorithm which unfortunately is highly possible to add a security and performance issue on the execution of this applet.

Another security improvement for this applet is the PIN security implementation, making possible for the card to be exchanged between nodes, protect the system from a card stealing and/or unauthorized use of a node.

A minor improvement also possible is the use of asymmetric cryptography and direct communication of the card with the Certificate Authority (CA), a heavy performance task for a limited hardware capabilities device but offering the advantage for the card to be able to create by its own the secure channel required for the initial phase of nodes personalization.

6.2 Next Generation smart cards

The increasing use of smart cards in our daily life and the increasing demands of modern applications for memory and computing power necessitate the evolution of smart card technology. The traditional smart card technology though until now is sufficient, it seems that in a few years will not be able to meet the development of new applications. The next generation smart cards is expected to take advantage of developments in semiconductor technology, promising faster microprocessors, greater memory capacity, faster pulse generators (clocks) and two-way (full duplex) communication between the card and the reader. Table 6-1 shows a comparison between current smart cards technology and next generation smart cards [52].

Traditional Smart Cards	Next generation smart cards
8/16 bit CPU	32 bit CPU
2K RAM	16K RAM
48-64K ROM	>256K ROM
8-32K EEPROM	>128K EEPROM
External Clock 1-5 MHz	Internal Clock 50 MHz
Half duplex Input/output Serial port	Full duplex high speed Input/output Serial port 1,5Mbps–12Mbps

Table 6-1: Comparison of current smart card technology with the next generation of smart cards

6.3 Current research directions

The research in smart card technology is focusing on the ease of access to remote service points in a secure way for the card holder, smart cards contributing as internet access tokens taking advantage of the capabilities of the cards to provide security, portability, wallet compatible form-factor and tamper-resistance [53].

Furthermore, due to the nature of smart cards as secure devices, there is an undiminished interest in research for new authentication schemas, using new techniques and technologies, assisted by the technological advances of smart card's hardware and operating systems.

Due to the maturity of smart cards that exist on the market for over two decades, there is a decrease in the number of new types of attacks discovered, on the contrary, the advances in security continue unabated, continually increasing the security provided by these devices.

The greatest research interest concerning security in smart cards is focusing on improving the authentication mechanisms available and particularly mechanisms that are recent in the field of smart card technology, such as: Biometric-based user authentication, chaotic map-based authentication and Elliptic Curves Cryptography (ECC)

The use of biometric characteristics of a user is considered as a very secure method for user authentication, since it uses the unique biometric characteristics of a person to provide authentication. The current research is focusing in ways of implementing this mechanism usually using hash functions [54], [55] for matching the input biometric characteristics of a user with pre-stored template in smart card's memory but requiring a perfect match of those for a successful authentication, while other researches examine more flexible approaches, capable of recognizing the percentage of matching between the biometric characteristics received and those stored [56],[57].

In the last decade, chaos has emerged as a new promising candidate for cryptography because of its many useful characteristics such as broadband spectrum, ergodicity and high sensitivity to initial conditions, thus, generating confusion and diffusion conditions, invaluable for generating good ciphers. The use of maps that exhibits chaotic behavior does not offer only these advantages but also simplicity and rapidity making understood why there is so much interest in this area. Since 2007, where Xiao et al. [58] proposed a novel key agreement protocol, based on chaotic maps showed the usefulness of this technique and the need for further research in this area. Since then a lot of major research [59], [60] and improvements [61], [62], [63] has been made on this authentication method.

Another mechanism that has recently found usability in smart cards is this of elliptic curves cryptography. Elliptic Curve Cryptography (ECC) initially proposed by V. S. Miller in 1986 [64] and N. Koblitz in 1987 [65] us a new promising technology in cryptography has recently produced a growing interest for implementing it in smart cards. In practice, ECC is used to instantiate public-key cryptography protocols and more than 25 years after their introduction to cryptography, the practical benefits of using elliptic curves are well-understood: they offer smaller key sizes [66] and more efficient implementations, a key benefit for use in smart cards applications, at the same security level as other widely deployed schemes such as RSA. This is the reason why the current research is focusing in this area [67], [68].

Bibliography

[1] D. Chadwick, "Smart cards aren't always the smart choice," *IEEE Computer,* pp. 142-143, 1999.

[2] "Diners Club - Company history of Diners Club," [Online]. Available: https://www.dinersclubus.com/home/about/dinersclub/story. [Accessed 8 February 2015].

[3] "Wikipedia," GSM (Global System for Mobile Communications), [Online]. Available: http://en.wikipedia.org/wiki/GSM.

[4] W. V. Winkle, "Strange Horizons, Volume 1," Smashwords Edition, 2011, p. 13.

[5] "Visa International," [Online]. Available: http://www.visa.com. [Accessed 2 February 2015].

[6] "Mondex," [Online]. Available: http://www.mondex.com. [Accessed 2 February 2015].

[7] N. A. Raja, "Application of Smart Card Technology in The University of Mumbai Library to Monitor User Behaviour, Library Traffic, Library Planning, Physical Security and Managing E-purse for Cashless Transactions," in *INFLIBNET Centre*, Gangtok, 2012.

[8] "Development of an Automatic Fare Collection System for Athens Urban Transport with PPP – Public Consultation," OASA, [Online]. Available: http://www.oasa.gr/news.php?id=funk295&lang=en. [Accessed 2015 February 15].

[9] D. Bodson, "The European Telecommunications Standards Institute Cohosts Global IPv6 Transition Test Event [Standards]," *Vehicular Technology Magazine, IEEE,* vol. 7, no. 4, pp. 131-134, 2012.

[10] N. Daswani, C. Kern and A. Kesavan, Foundations of SECURITY, Apress, 2007.

[11] "International Card Manufacturers Association," ICMA, [Online]. Available: https://icma.com. [Accessed 2015 February 18].

[12] "The Voice of the Smart Security Industry," Eurosmart, [Online]. Available: http://www.eurosmart.com/. [Accessed 2015 February 18].

[13] "Smart Secure Devices Market Analysis," Eurosmart, [Online]. Available: http://www.eurosmart.com/expertise/market-analysis. [Accessed 2015 February 19].

[14] "Global Trends Futuring," ICMA, [Online]. Available: https://icma.com/wp-content/uploads/2014/10/global-Europe-trends-futuring.pdf. [Accessed 2015 February 19].

[15] "Growth trend for solutions combining convenience and security continues," Eurosmart, [Online]. Available: http://www.eurosmart.com/publications/market-overview/37-

uncategorised/261-figures-2013. [Accessed 20 February 2015].

[16] "Hyperconnected World," World Economic Forum, [Online]. Available: http://www.weforum.org/projects/hyperconnected-world. [Accessed 20 February 2015].

[17] "CARTES Secure Connexions," [Online]. Available: http://www.cartes.com/. [Accessed 20 February 2015].

[18] "Vision 2020," Eurosmart, [Online]. Available: http://www.eurosmart.com/images/Eurosmart%20Vision%20Paper%202020.pdf. [Accessed 2015 February 19].

[19] "Smart Card Basics," [Online]. Available: http://www.smartcardbasics.com/standards.html. [Accessed 2 February 2015].

[20] "EMVCo," [Online]. Available: http://www.emvco.com. [Accessed 3 February 2015].

[21] R. Wolfgang and E. Wolfgang, Smart Card Handbook Fourth Version, WILEY, 2010, p. 241.

[22] F. Klaus, RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, RFID and NFC (Third Edition), WILEY, 2010.

[23] P. Saha, Handbook of Enterprise Systems Architecture in Practice, Idea Group Inc, 2007.

[24] H. Yahya and T. Thomas, Smart Card Manufacturing: A Practical Guide, WILEY, 2002.

[25] N. Deepak, Welcome to The World of Smart Cards, 2014.

[26] D. Deville, A. Galland, G. Grimaud and S. Jean, Smart card operating systems: Past, present and future, Citeseer, 2003.

[27] T. M. Jurgensen and S. B. Guthery, Smart Cards: The Developer's Toolkit, Prentice Hall Professional, 2002.

[28] "Java Card Forum," [Online]. Available: http://javacardforum.com/. [Accessed 3 February 2015].

[29] "Java Card Technology," [Online]. Available: http://www.oracle.com/technetwork/java/embedded/javacard/overview/index.html. [Accessed 3 February 2015].

[30] "Oracle," [Online]. Available: http://www.oracle.com/index.html. [Accessed 3 February 2015].

[31] "MultOS," [Online]. Available: http://www.multos.com/. [Accessed 3 February 2015].

[32] W. Kok-Seng, K. Minsu, L. Sangjun and H. K. Myung, "Practical Biometrics On-Card Matching for Remote User Authentication Using Smart Cards," *Computer Science and its Applications,* vol.

330, pp. 505-510, 2015.

[33] G. Bouffard, "A Generic Approach for Protecting Java Card Smart Card Against Software," HAL, 2015.

[34] H. Helena, "Contactless Technology Security Issues," *Information Security Bulletin,* vol. 9, pp. 95-100, 2004.

[35] K. Hoon and D. C. Ronnie, "A Review of Smartcard Security Issues," *Journal of Security Engineering,* vol. 1, no. 11, pp. 359-370, 2011.

[36] P. Kocher, J. Jaffe and B. Jun, "Differential Power Analysis," *Advances in Cryptology — CRYPTO' 99,* vol. 1666, pp. 388-397, 1999.

[37] E. C. Ortiz, "An Introduction to Java Card Technology - Part 2, The Java Card Applet," [Online]. Available: http://www.oracle.com/technetwork/java/javacard/javacard2-138597.html. [Accessed 4 February 2015].

[38] H. Engelbert, O. Martijn and P. Erik, "From Finite State Machines To Provably Corect Java Card Applets," *Security and Privacy in the Age of Uncertainty,* vol. 122, pp. 465-470, 2003.

[39] "Writing a Java Card Applet," Oracle, [Online]. Available: http://www.oracle.com/technetwork/java/javacard/intro-139322.html. [Accessed 4 February 2015].

[40] "Java Card Specifications," Oracle, [Online]. Available: http://pfa12.free.fr/doc_java/javacard_specifications/specs/jcre/html/JCRESpec05transient.html. [Accessed 5 February 2015].

[41] Oracle, "Java Card™ Platform Security: Technical White Paper," [Online]. Available: http://www.oracle.com/technetwork/java/javacard/documentation/javacardsecuritywhitepaper-149957.pdf.

[42] "Java Card 2.2 Off-Card Verifier: White Paper," [Online]. Available: http://www.oracle.com/technetwork/java/embedded/javacard/documentation/offcardverifierwp-150021.pdf.

[43] "Card Technology Today," *Elsevier,* vol. 20, no. 5, p. 3.

[44] "Java Card Platform Specification," Oracle Std. Version 3.0.1, 2009.

[45] "RFC 1122 - Requirements for Internet Hosts - CommunicationLayers," Tech. Rep., United States, 1989.

[46] T. Dierks and E. Rescorla, "RFC 5246 - The Transport Layer Security(TLS) Protocol Version 1.2," Tech. Rep., 2008.

[47] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee, "RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1,," Tech. Rep, United States, 1999.

[48] E. Rescorla and A. Schiffman, "RFC 2660 - The Secure HyperTextTransfer Protocol," Tech. Rep., United States, 1999.

[49] "Smartcard Web Server Enabler Architecture," Open Mobile Alliance (OMA), 2008.

[50] "nSHIELD project," [Online]. Available: http://www.newshield.eu/. [Accessed 6 February 2015].

[51] "Java Card™ Platform Security : Technical White Paper," Oracle, 2001.

[52] R. Tanjore, T. Florian and V. Thierry, "Java Card Evolution: Future Directions," Java Card Team, Sun Microsystems, 2006.

[53] A. M. Ali, "Zero Footprint Secure Internet Authentication Using Network Smart Card," *Smart Card Research and Advanced Applications,* vol. 3928, pp. 91-104, 2006.

[54] M. C. Chuang and C. C. Meng, "An anonymous multi-server authenticated key agreement scheme based on trust computing using smart cards and biometrics," *Expert Systems with Applications,* vol. 41, no. 4, pp. 1411-1418, 2014.

[55] L. Xiong, N. Jian-Wei, M. Jian, W. Wen-Dong and L. Cheng-Lian, "Cryptanalysis and improvement of a biometrics-based remote user authentication scheme using smart cards," *Journal of Network and Computer Applications,* vol. 34, no. 1, pp. 73-79, 2011.

[56] W. Kok-Seng, K. Minsu, L. Sangjun and H. K. Myung, "Practical Biometrics On-Card Matching for Remote User Authentication Using Smart Cards," *Computer Science and its Applications,* vol. 330, pp. 505-510, 2015.

[57] C. Younsung, N. Junghyun, L. Donghoon, K. Jiye, J. Jaewook and W. Dongho, "Security Enhanced Anonymous Multiserver Authenticated Key Agreement Scheme Using Smart Cards and Biometrics," *The Scientific World Journal,* vol. 2014, p. 15, 2014.

[58] D. Xiao, X. Liao and S. Deng, "A novel key agreement protocol based on chaotic maps," *Information Sciences,* vol. 177, no. 4, pp. 1136-1142, 2007.

[59] S. M. Farash and A. M. Attari, "An efficient and provably secure three-party password-based authenticated key exchange protocol based on Chebyshev chaotic maps," *Nonlinear Dynamics,* vol. 77, no. 1-2, pp. 399-411, 2014.

[60] D. Mishra, K. A. Das and S. Mukhopadhyay, "A secure user anonymity-preserving biometric-based multi-server authenticated key agreement scheme using smart cards," *Expert Systems with Applications,* vol. 41, no. 18, pp. 8129-8143, 2014.

[61] Q. Xie, B. Hu and T. Wu, "Improvement of a chaotic maps-based three-party password-authenticated key exchange protocol without using server's public key and smart card,"

Nonlinear Dynamics, pp. 1-14, 2014.

[62] C.-C. Lee, D.-C. Lou, C.-T. Li and C.-W. Hsu, "An extended chaotic-maps-based protocol with key agreement for multiserver environments," *Nonlinear Dynamics,* vol. 76, no. 1, pp. 853-866, 2013.

[63] S. M. Farash and A. M. Attari, "Cryptanalysis and improvement of a chaotic map-based key agreement protocol using Chebyshev sequence membership testing," *Nonlinear Dynamics,* vol. 76, no. 2, pp. 1203-1213, 2014.

[64] S. V. Miller, "Use of Elliptic Curves in Cryptography," *Advances in Cryptology — CRYPTO '85 Proceedings,* vol. 218, pp. 417-426, 1986.

[65] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comp.,* vol. 48, pp. 203-209, 1987.

[66] K. A. Lenstra and R. E. Verheul, "Selecting Cryptographic Key Sizes," *Journal of Cryptology,* vol. 14, pp. 255-293, 2001.

[67] A. Abidi, "Implementation of elliptic curve digital signature algorithm (ECDSA)," *Computer & Information Technology,* pp. 1-6, 2014.

[68] H. L. Yeh, T. H. Chen and W. K. Shih, "Robust smart card secured authentication scheme on SIP using Elliptic Curve Cryptography," *Computer Standards & Interfaces,* vol. 36, no. 2, pp. 397-402, 2014.

[69] N. R. Akram and K. Markantonakis, "Smart Cards: State-of-the-Art to Future Directions, Invited Paper," in *IEEE International Symposium on Signal Processing and Information Technology (ISSPIT 2013)*, IEEE Computer Science, 2013.

[70] "ARTEMIS Joint Undertaking," [Online]. Available: http://www.artemis-ju.eu/home_page. [Accessed 6 February 2015].

[71] S. ,. M. Farash and A. M. Attari, "An efficient and provably secure three-party password-based authenticated key exchange protocol based on Chebyshev chaotic maps," *Nonlinear Dynamics,* vol. 77, no. 1-2, pp. 399-411, 2014.