

Implementation of Inspection System for Biometric Passports based on ICAO Specifications

Technical Report

February 11, 2009

Responsible
Luis Terán Tamayo

Prof. Andrzej Drygajlo
Assistant: Dr. Jonas Richiardi

Acknowledgments

I would like to thank Prof. Andrzej Drygajlo for giving me the opportunity to work with him and for supervising the development of this project.

I also would like to thank Dr. Jonas Richiardi for his continued advice, guidance and patience in the development of this project.

Thanks to Dr. Lionel Beaugé and Loïs Lherbier from COVADIS S.A for their support and feedback during the implementation of this project.

To my parents, Lupe Tamayo and Saul Terán, my brothers, Christian and Jazmine, and my friends, thanks for the support during this time far from home.

And my special gratitude to Pitu for being with me during this time. I love you so much.

List of Abbreviations and Acronyms

APDU	Application Protocol Data Units.
API	Application Programming Interface.
BHT	Biometric Header Template.
BDB	Biometric Data Block.
CAP	Converted Applet.
CSCA	Country Signing Certificate Authority
C_{CSCA}	Country Signing Certificate Authority Certificates.
C_{DS}	Document Signer Certificates.
CRL	Certificate Revocation Lists.
CREF	C Reference Implementation of Java Card Run-time Environment.
C-APDU	Command Application Protocol Data Unit.
DF	Dedicated File.
EEPROM	Electrically Erasable Programmable Read-Only Memory
EF	Elementary File.
ePassport	An MRTD passport that has a circuit chip imbedded in it, in accordance with the standards proposed by ICAO.
GUI	Graphical User Interfaces.

ICAO	International Civil Organization.
ICAO TAG/MRTD	ICAO's Technical Advisory Group for Machine Readable Travel Documents.
JCVM	Java Card Virtual Machine.
JCRE	Java Card Runtime Environment.
JCWDE	Java Card Workstation Development Environment.
<i>KPr_{AA}</i>	Active Authentication Private Key.
<i>KPu_{AA}</i>	Active Authentication Public Key.
<i>KPr_{CSCA}</i>	Country Signing Certificate Authority Private Keys.
<i>KPu_{CSCA}</i>	Country Signing Certificate Authority Public Keys.
<i>KPr_{DS}</i>	Document Signer Private Keys.
<i>KPu_{DS}</i>	Document Signer Public Keys.
<i>K_{ENC}</i>	Document Basic Access Encryption Key.
<i>K_{MAC}</i>	Document Basic Access Authentication Key.
<i>K_{IFD/ICC}</i>	Key Seed.
<i>K_{SENC}</i>	Encryption Session Key.
<i>K_{S_{MAC}}</i>	Authentication Session Key.
LDS	Logical Data Structure.
MF	Master File
MRTD	Machine Readable Travel Document.
MRZ	Machine Readable Zone.
NTWG	New Technologies Working Group.
PKD	Public Key Directory.
PKI	Public Key Infrastructure.
ROM	Read-only Memory.
RAM	Random-access memory.
R-APDU	Response Application Protocol Data Unit.
<i>SO_D</i>	Document Security Objects.
TLV	Tag - Length - Value.
UML	Unified Modeling Language.

Contents

1. Introduction	7
1.1. Motivation	8
1.2. Goals	8
1.3. Structure	8
2. A Brief Overview of Machine Readable Travel Documents Specifications	9
2.1. Machine Readable Travel Documents	9
2.1.1. Evolution of MRTD's	10
2.1.2. Machine Readable Zone (MRZ):	10
2.2. ICAO Technical Advisory Group	11
2.2.1. Logical Data Structure Technical Report	12
2.2.1.1. File Structure	13
2.2.1.2. Common Data Elements	15
2.2.1.3. Data Group 1	15
2.2.1.4. Data Group 2 to Data Group 4	15
2.2.1.5. Data Group 5 and Data Group 7	16
2.2.1.6. Data Group 8 to Data Group 10	17
2.2.1.7. Data Group 11	17
2.2.1.8. Data Group 12	18
2.2.1.9. Data Group 13	18
2.2.1.10. Data Group 15	18
2.2.1.11. Data Group 16	19
2.2.1.12. LDS Security Data	19
2.2.1.13. Data Group codification example	19
2.2.2. Public Key Infrastructure Technical Report	21
2.2.2.1. Country Signing Certificate Authority Certificates	21
2.2.2.2. Document Signer Certificates	21
2.2.2.3. Certificate Revocation	21
2.2.2.4. ICAO Public Key Directory	22
2.2.2.5. Authentication of MRTD's	22
2.2.2.5.1. Passive Authentication	22
2.2.2.5.2. Active Authentication	22
2.2.2.6. Access Control	22
2.2.2.6.1. Extended Access Control	23
2.2.2.6.2. Basic Access Control	23
2.2.2.7. Secure Messaging	25
2.2.2.8. Security Methods Comparison	25
3. Description and Justification of Used Technologies	27
3.1. Smart Cards	27
3.1.1. Smart Card Memory System	28
3.1.2. Smart Card Communication	28
3.1.3. Smart Card File System	29
3.2. Java Card Technology	30
3.2.1. Specifications	30
3.2.1.1. Java Card Virtual Machine	30
3.2.1.2. Java Card Runtime Environment	31
3.2.1.3. Java Card Application Programming Interface	31
3.2.2. Java Card Evolution	31
3.3. Java	32
3.3.1. Swing	33

3.3.2. JDBC	33
3.3.3. Smart Card I/O API	34
3.4. DataBase MySQL	34
3.5. GPSHELL Framework	38
3.6. Alya	39
3.7. Application of Technologies	39
4. Description of Software Developed	40
4.1. Architecture	40
4.2. Database Manager	41
4.2.1. Database Connection	41
4.2.2. Get Data from Database	42
4.2.3. Get Image from Database	42
4.2.4. Upload Data to Database	43
4.2.5. Upload Image to Database	44
4.3. JCWDE Simulator	44
4.3.1. Configure	44
4.3.2. Get Data from JCWDE Simulator	46
4.3.3. Get Image from JCWDE Simulator	46
4.4. Reader	47
4.4.1. Write Card from Data Base	47
4.4.2. Read Card	48
4.5. Security	49
4.6. Graphical Interface	49
4.6.1. Read from Database	50
4.6.2. Write to Database	50
4.6.3. Configure JCWDE Simulator	51
4.6.4. Read from JCWDE Simulator	52
4.6.5. Read Card	52
4.6.6. Write Card from Database	53
5. Conclusion and Future Work	54
5.1. Conclusions	54
5.2. Future Work	55
5.2.1. Performance Improvement	55
5.2.2. Database Improvement	55
5.2.3. Security Improvements	55
Annex A	56
INSTALLATION OF GPSHELL FOR LEOPARD MAC OS 10.5	56
INSTALLATION OF API IN GEMALTO XPRESS PRO USING GPSHELL ON LEOPARD MAC OS 10.5	57
Annex B	60
INSTALLATION OF EASYECLIPSE DESKTOP JAVA	60
INSTALLATION OF ECLIPSEJCDE PLUGIN FOR IDE ECLIPSE	60
Bibliography	61

Chapter 1

1. Introduction

Over the last two years, Biometric Passports have been introduced in many countries to increase the security in Inspection Systems and enhance procedures and systems that prevent identity and passport fraud. Along with the deployment of new technologies, countries need to test and evaluate its systems since the International Civil Aviation Organization (ICAO) provides the guidelines, but the implementation is up to each issuing country.

The specific choices of each countries as to which security features to include or not include makes a major difference in the level of security and privacy protection available. Table 1.1 shows some examples of ePassport implementations and the security features selected.

Country	RFID Type	Deployment	Security	Biometric
Belgium	14443	2004	Unknown	Photo
U.S.	14443	2005	Passive, Active Authentication	Photo
Netherlands	14443	2005	Passive, Active Authentication, BAC	Photo
Germany	14443	2005	Passive, Active Authentication, BAC	Photo

Table 1.1: ePassport Deployments

1.1. Motivation

Currently it is possible to implement Biometric Passport applets according to ICAO specifications. The motivation of this project is to develop and implement a system capable of testing and evaluating an ePassport applet in order to better understand its functionalities and capabilities; and to propose new alternatives related to security and management of this type of systems.

1.2. Goals

The goals of this thesis can be summarized as follows:

- Design of an ePassport applet according to the ICAO specifications for Biometric Passports
- Design and implementation of an Inspection System for Biometric Passports.
- Implementation of a Basic Access Control for Biometric Passports.

1.3. Structure

The next chapters of this thesis are organized as follows: Chapter 2 describes the standards and specifications of Machine Readable Travel Documents (MRTD's). Chapter 3 describes the technologies used for the implementation of the inspection system for Biometric Passports. Chapter 4 provides a description of the implementation and its utilization. Chapter 5 concludes this thesis and gives an outlook of the future work.

Chapter 2

2. A Brief Overview of Machine Readable Travel Documents Specifications

2.1. Machine Readable Travel Documents

The Machine Readable Travel Document (MRTD) is an international travel document, which contains eye and machine-readable data of the travel document holder.

MRTDs facilitate the identification of travelers and enhance the security levels.

MRTDs are developed with the assistance of the ICAO's Technical Advisory Group for Machine Readable Travel Documents (ICAO TAG/MRTD) and the ISO Working Group 3 (JTC1/SC17/WG3).

- Check Digit - Document Number
- Nationality
- Date of Birth
- Check Digit - Date of Birth
- Sex
- Date of Expiry
- Check Digit - Date of Expiry
- Optional Data
- Check Digit - Optional Data (ID-3 Document Only)
- Check Digit - Composite.

2.2. ICAO Technical Advisory Group

ICAO New Technologies Working Group (NTWG) focuses on machine assisted identity confirmation of persons based on biometric features.

ICAO uses face recognition as the main biometric with fingerprints and iris recognition as a secondary biometrics.

It also specifies a Logical Data Structure (LDS), which is electronically encoded in the travel document using Public Key Infrastructure to protect and authenticate the data.

Machine Readable Travel Documents (MRTDs) such as passports, visas or other travel documents, have the following key considerations:

- Global Interoperability
- Uniformity
- Technical Reliability
- Practicality
- Durability

In biometrics terminology, ICAO uses the term “verification” to perform a one-to-one match between proffered biometric data obtained from the MRTD, and biometrical template created when the holder enrolled in the system.

“Identification” is used in order to perform a one-to-many search between proffered biometric data and a collection of templates representing all of the subjects who have enrolled in the system

The components of a biometric system are:

- Capture of biometric sample from holder
- Extraction of biometric features from biometric sample

- Template Creation for storage of biometric sample
- Comparison with the information of an stored template

ICAO has elaborated different technical reports. The main technical reports, which are part of ICAO specifications and considered in this project, are:

- Logical Data Structure Technical Report
- Public Key Infrastructure Technical Report

2.2.1. Logical Data Structure Technical Report

This report specifies the data that will be stored using capacity data expansion technologies (e.i. IC(s) with contacts, contactless IC(s), optical memory and bar codes).

The LDS technical report specifies a global and interoperable data structure for recording identity details including biometric data. The data stored in the LDS is read electronically and designed to be flexible and expandable for future needs. A series of mandatory and optional elements has been defined for LDS, which are used in MRTDs. The use of biometrics is optional except the use of the encoded face.

Figure 2.2 shows the complete structure of LDS, which includes mandatory and optional data elements defined for LDS (version 1.7)

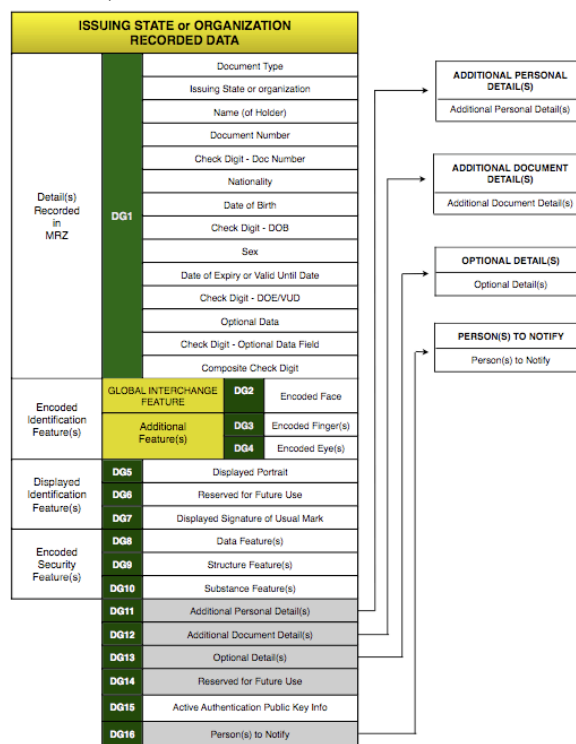


Figure 2.2: Mandatory and Optional Data Elements defined for LDS (version 1.7)

Table 2.1 shows the contents of MRTD's chip.

GROUP	DESCRIPTION	SPECIFICATION
K_{ENC}	Encryption Key	Optional
K_{MAC}	Message Authentication Code Key	Optional
K_{PrAA}	Active Authentication Private Key	Optional
COM	Present Data Groups	Mandatory
DG1	Same as MRZ	Mandatory
DG2	Encoded Face	Mandatory
DGn	Data Groups from DG3 to DG16	Optional
S_{OD}	Security Field	Mandatory

Table 2.1: MRTD contents.

2.2.1.1. File Structure

The information stored in the MRTD IC chip is saved according to the ISO/IEC 7816-4. This information is saved in a file system, which is organized hierarchically as follows:

- MASTER FILE (MF): this file maybe the root of the File System.
- DEDICATED FILE (DF's): this file contains Elementary Files or other Dedicated Files.
- ELEMENTARY FILES (EF's): elementary files contain the information of each data group.

Each data group is stored in one transparent EF addressable by short file ID as shown in Table 2.2.

Data Group	EF Name	Short EF identifier	FID	Tag
Common	EF.COM	'1E'	'01 1E'	'60'
DG1	EF.DG1	'01'	' 01 01'	'61'
DG2	EF.DG2	'02'	' 01 02'	'75'
DG3	EF.DG3	'03'	' 01 03'	'63'
DG4	EF.DG4	'04'	' 01 04'	'76'
DG5	EF.DG5	'05'	' 01 05'	'65'
DG6	EF.DG6	'06'	' 01 06'	'66'

Data Group	EF Name	Short EF identifier	FID	Tag
DG7	EF.DG7	'07'	' 01 07'	'67'
DG8	EF.DG8	'08'	' 01 08'	'68'
DG9	EF.DG9	'09'	' 01 09'	'69'
DG10	EF.DG10	'0A'	' 01 0A'	'6A'
DG11	EF.DG11	'0B'	' 01 0B'	'6B'
DG12	EF.DG12	'0C'	' 01 0C'	'6C'
DG13	EF.DG13	'0D'	' 01 0D'	'6D'
DG14	EF.DG14	'0E'	' 01 0E'	'6E'
DG15	EF.DG15	'0F'	' 01 0F'	'6F'
DG16	EF.DG16	'10'	' 01 10'	'70'
Security Data	EF.SOD	'1D'	' 01 1D'	'77'

Table 2.2: Data Groups.

Each data group is stored in different EF's. The structure and coding of data objects are defined in ISO/IEC 7816-4[1] and 7816-6[2]. Each data object is encoded using Tag - Length - Value (TLV).

ISO/IEC/7816 supports Tag fields of one, two or three bytes. The Tag field specified for data groups is shown in Table 2.2. Each data group contains different elements, which also have their own Tag field.

The Length field provides the length of the value field, and it is coded in short and long form. In short form the bit 8 is set to 0 and the first 7 bits encode the length of bytes in the value field. In long form, the Length field consists of two or more bytes. Table 2.3 shows the encoding of Length field for different sizes of value field.

	1 st byte	2 nd byte	3 rd byte	4 th byte	5 th byte	Length of Value Field
1 byte	'00' to '7F'	-	-	-	-	0 to 127
2 bytes	'81'	'00' to 'FF'	-	-	-	0 to 255
3 bytes	'82'	'0000' to 'FFFF'		-	-	0 to 65535
4 bytes	'83'	'000000' to 'FFFFFFF'			-	0 to 16777215
5 bytes	'84'	'00000000' to 'FFFFFFFF'				0 to 4294967295

Table 2.3: Length Field.

Any data object is denoted {T - L - V} with a tag field followed by a length field encoding a number, which represents the size of the value field. If the size is equal to zero, the data field is absent. A constructed data object is denoted {T - L - V {T1 - L1 - V1}...{Tn - Ln - Vn}}, which represent a concatenation and interweaving of data objects. This type of structure is used in Data Groups containing more than one value field, which are preceded by specific Tag and Length field.

2.2.1.2. Common Data Elements

The common group is mandatory and contains information about versions and the data groups included in the application. Table 2.4 shows the content common data element.

Tag	L	Value
'5F01'	4	LDS version. Format: ab where, a = version of LDS, b = update level
'5F36'	6	Unicode version. Format: abc where, a = Major version, b = Minor version, c = Release level
'5C'	X	Tag List of data groups present

Table 2.4: Common Data Element.

2.2.1.3. Data Group 1

This data group 1 is mandatory and contains the Machine Readable Zone (MRZ) represented as one data element. Table 2.5 shows the content of data group 1.

Tag	L	Value
'5F1F'	44	The MRZ data objects

Table 2.5: Data Group 1.

2.2.1.4. Data Group 2 to Data Group 4

Data Groups 2 to 4 contain biometric information of face, fingerprint and irises. Data group 2 is mandatory and data groups 3 and 4 are optional. The Biometric Header Template is encoded using the tag 'Ax' designation starting from A1, and increasing the value or 'x' for new biometric header template as shown in Table 2.6. Table 2.6 shows the content of data group 2 to 4.

Tag	L	Value				
'7F61'	X	Biometric Information Group Template				
		Tag	L	Value		
		'02'	1	Integer number of instances		
		'7F60'	X	1 st Biometric Information Template		
			Tag	L	Value	
			'Ax'	X	Biometric Header Template (BHT). Start of template, where x (x =1, 2,3...) increments for each occurrence	
				Tag	L	Value
				'80'	'02'	ICAO header version (optional)
				'81'	'01'	Biometric Type (optional)
				'82'	'01'	Biometric Feature (optional for DG2, mandatory for DG3 and DG4)
				'83'	'07'	Creation Date and Time (optional)
				'84'	'08'	Validity Period (optional)
				'86'	'02'	Creator of the biometric reference data (optional)
				'87'	'02'	Format owner (mandatory)
				'88'	'02'	Format type (mandatory)
			'5F2E' or '7F2E'	X	Biometric data. Also called biometric data block (BDB)	

Table 2.6: Data Group 2 to Data Group 4.

2.2.1.5. Data Group 5 and Data Group 7

Data Groups 5 and 7 contain the portrait image and signature or usual mark, fingerprint and irises. Data group 5 and 7 are optional. Table 2.7 shows the content of data group 5 and 7.

Tag	L	Value
'02'	1	Integer number of instances of this type of displayed image
'5F36' or '5F36'	X	Displayed portrait Displayed signature or mark

Table 2.7: Data Group 5 and Data Group 7.

2.2.1.6. Data Group 8 to Data Group 10

Data groups 8 to 10 are used for security features, and remain to be defined. Table 2.8 shows the content of data group 8 to 10.

Tag	L	Value
'02'	1	Integer number of instances of this type of template
	X	Header Template. Details to be defined

Table 2.8: Data Group 8 to Data Group 10.

2.2.1.7. Data Group 11

Data group 11 is used for additional information about the document holder. This data group is optional and could include non-latin characters. Table 2.9 shows the content of data group 11.

Tag	L	Value
'5C'	X	Tag list of data elements in the template
'5F0E'	X	Full name of document holder in national characters
'A0'	'01'	Content specific constructed data object of names
'02'	01	Number of other names
'5F0F'	X	Personal number
'5F2B'	8	Full date of birth yyyymmdd
'5F11'	X	Place of birth. Fields separated by '<'
'5F42'	X	Permanent address. Fields separated by '<'
'5F12'	X	Telephone
'5F13'	X	Profession
'5F14'	X	Title
'5F15'	X	Personal summary
'5F16'	X	Proof of citizenship
'5F17'	X	Other valid TD numbers. Separated by '<'
'5F18'	X	Custody information

Table 2.9: Data Group 11.

2.2.1.8. Data Group 12

Data group 12 is used for additional information about the document. This data group is optional. Table 2.10 shows the content of data group 12.

Tag	L	Value
'5C'	X	Tag list of data elements in the template
'5F19'	X	Issuing authority
'5F26'	X	Date of Issue. yyymmdd
'A0'	'01'	Context specific constructed data object of other people
'02'	'01'	Number of other people
'5F1A'	X	Name of other person formatted per ICAO 9303 rule
'5F1B'	X	Endorsements, observations
'5F1C'	X	Tax, exit requirements
'5F1D'	X	Image of front of document
'5F1E'	X	Image of rear of document
'5F85'	X	Date and time of document personalization yyymddhhmmss
'5F86'	X	Serial number of personalization system

Table 2.10: Data Group 12.

2.2.1.9. Data Group 13

This Data Group is reserved for national specific data. Its format is country defined.

2.2.1.10. Data Group 15

Data Group 15 contains the Active Authentication Public Key Information. Table 2.11 shows the content of data group 15.

Tag	L	Value
'6F'	X	Active Authentication Public Key

Table 2.11: Data Group 15.

2.2.1.11. Data Group 16

Data Group 16 contains emergency information. It is encoded using the tag ‘Ax’ designation. The data is not signed, allowing for updating by the document holder. Table 2.12 shows the content of data group 16.

Tag	L	Value
‘02’	1	Number of templates (occurs only in the first template)
‘Ax’	X	Start of template, where x (x =1, 2,3...) increments for each occurrence
‘5F50’	X	Date data recorded
‘5F51’	X	Name of person
‘5F52’	X	Telephone
‘5F53’	X	Address

Table 2.12: Data Group 16.

2.2.1.12. LDS Security Data

This data group contains a signed data structure of all data groups implemented using the public key of issuing state. Table 2.13 shows the content LDS Security Data.

Tag	L	Value
‘77’	X	Hash values of all data groups signed with public key of issuing state

Table 2.13: LDS Security Data.

2.2.1.13. Data Group codification example

The following example shows the encoding of Data Group 2 of a facial biometric data block and using TLV encoding. The Tags are shown in blue, lengths are shown in red and values are shown in black. All the values are coded in hexadecimal representation.

The length value is encoded according to Table 2.3. For this example the length is codified using one and two bytes (Tag ‘82’ is used in order to codify lengths bigger than 255 bytes).

The information about the biometric template used in this example is:

Data Block Length: 12642 bytes ('3162' in hexadecimal representation)

PID codification: '0001'

Format type: '0004'

Template provider: '000A'

Creation date: 15 March 2002 13:30:00

Validity: from 1 April 2002 through 31 March 2007

Figure 2.3 shows the codification of data group 2 according to information described above.

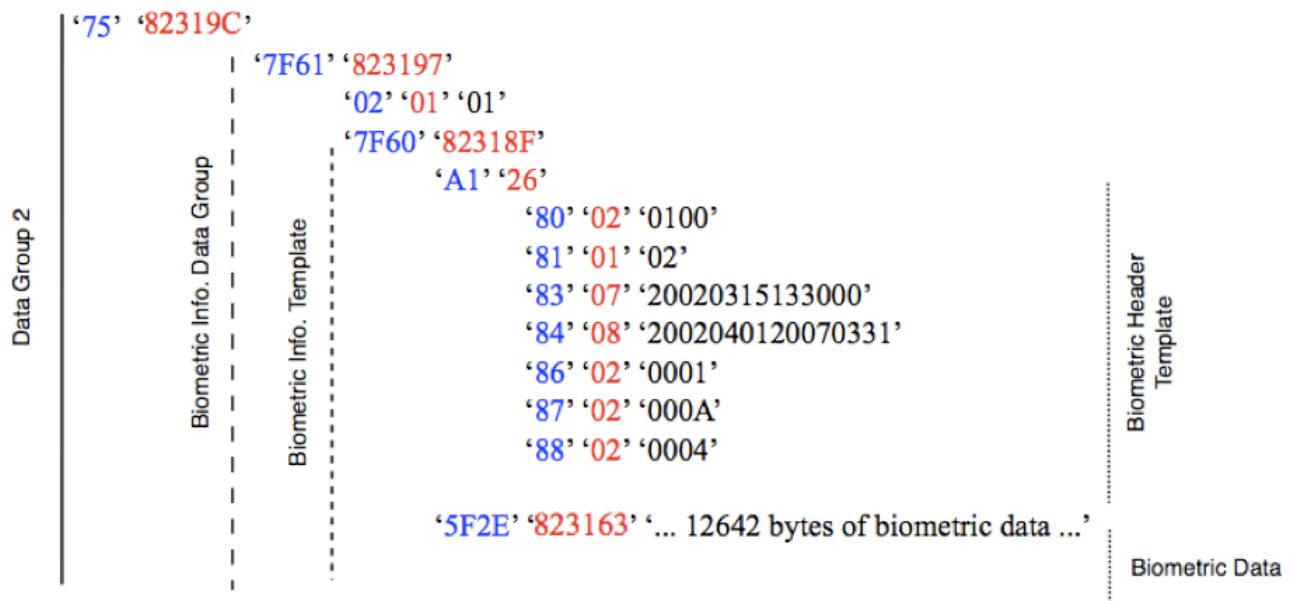


Figure 2.3: Data Group 2 example.

In figure 2.3, the first Tag ('75') is used to specify the data group 2. The second Tag ('7F61') is used in order to identify that biometric information is codified in this data block. The third Tag ('02') specifies the number of instances used. The fourth Tag ('7F60') is used in order to identify the first biometric template of the first instance of biometric data. In this example only one instance is used.

Tag 'A1' is used to identify the biometric header of the first instance. This biometric header contains information about the biometric instance, and uses Tag values: '80', '81', '83', '84', '86', '87', '88', according to Table 2.6

Finally, Tag '5F2E' is used to identify the Biometric Data Block (BDB), which in this example corresponds of 12642 bytes.

2.2.2. Public Key Infrastructure Technical Report

The aim of the PKI report scheme is to enable MRTD-inspecting authorities (Receiving States) to verify the authenticity and integrity of the data stored in the MRTD.

2.2.2.1. Country Signing Certificate Authority Certificates

Each Issuing Country is recommended to use a hierarchical Certification Authority infrastructure. The highest level in the Public Key Infrastructure is the Country Signing Certificate Authority (CSCA), which acts as a trust point for the receiving state. The generation of Key Pairs, Country Signing Certificate Authority Private Keys (KPr_{CSCA}) and Country Signing Certificate Authority Public Keys (KPu_{CSCA}) must be generated in a highly protected, off line security fashion.

Country Signing Certificate Authorities generate Country Signing Certificate Authority Certificates (C_{CSCA}), which carries the Country Signing Certificate Authority Public Keys (KPu_{CSCA}). Country Signing Certificate Authority Certificates must be distributed to ICAO with strictly secure diplomatic means in order to ensure authenticity of Document Signer Certificates (C_{DS}).

2.2.2.2. Document Signer Certificates

CSCA generates Key Pairs, Document Signer Private Keys (KPr_{DS}) and Document Signer Public Keys (KPu_{DS}), in a highly protected, off line security fashion. A Document Signer Certificate (C_{DS}), which contains KPu_{DS} , is generated by CSCA and must be forwarded to ICAO. It also may be included in the MRTD chip.

KPr_{DS} is used to sign the Document Security Objects (SO_D), and the KPu_{DS} is used to verify the authenticity of information saved in the MRTD.

2.2.2.3. Certificate Revocation

In case of key compromise, Issuing States must use a revocation method in order to communicate all participating States and ICAO about the revocation within 48 hours.

In case of absence of incidents, Issuing States should distribute bilaterally the Certificate Revocation Lists (CRL) to all participating States and to ICAO every 90 days.

2.2.2.4. ICAO Public Key Directory

ICAO will implement a Public Key Directory (PKD) Service for all participating countries to be able to distribute their C_{DS} . The updating service is restricted only to member States.

2.2.2.5. Authentication of MRTD's

ICAO has specified two types of authentications: Passive and Active Authentication. These types of authentication are shown in the following sections.

2.2.2.5.1. Passive Authentication

In this type of authentication the IC chip contains the information of all Data Groups and SO_D , which is a hashed value of all the Data Groups signed by the issuing State.

In this type of authentication, an Inspection System, which contains the appropriate public keys of each state will be able to verify the Document Security Object.

This authentication process does not need processing capabilities in the IC chip to establish a dynamic session.

2.2.2.5.2. Active Authentication

Active Authentication prevents a chip substitution using a challenge-response protocol between the Inspection System and the MRTD chip.

With Active Authentication the IC chip saves its own Authentication private and public Key pairs KPr_{AA} and KPu_{AA} , which are used together with the visual authentication of MRZ.

This type of authentication ensures that SO_D has been read from a non-copied chip. This authentication process needs processing capabilities in the IC chip.

2.2.2.6. Access Control

The main treats of a contactless IC chip compared with a traditional contact chip are that the information stored in a contactless chip could be read without opening the document, and that an unencrypted communication between a chip and a reader could be eavesdropped.

2. The first 8 bits of $HASH1$ is set as $K_a(ENC)$ and the last 8 bits of $HASH1$ are set as $K_b(ENC)$ in a two keys triple DES cryptographic algorithm.
3. The first 8 bits of $HASH2$ is set as $K_a(MAC)$ and the last 8 bits of $HASH2$ are set as $K_b(MAC)$ in a two keys triple DES cryptographic algorithm.

Authentication and Establishment of Session Keys

1. The inspection system requests an 8 byte challenge ($RND.ICC$) from ICC chip, and generates a random 8 bytes ($RND.IFD$) together with a random 16 bytes triple DES key (K_{IFD}).
2. The inspection system concatenates $RND.ICC$, $RND.IFD$, and K_{IFD} .

$$S = RND.IFD || RND.ICC || K_{IFD}$$

3. The inspection system sends to MRTD the encrypted version of S using K_{ENC} and K_{MAC} using MUTUAL_AUTENTICATE function of ISO7816-4

$$cmd_data = E[K_{ENC}](S) || MAC[K_{MAC}](E[K_{ENC}](S))$$

4. MRTD decrypts and verifies the received data.
5. MRTD computes a 16 bytes triple DES key (K_{ICC}).
6. MRTD concatenates $RND.ICC$, $RND.IFD$, and K_{ICC} .

$$R = RND.ICC || RND.IFD || K_{ICC}$$

7. MRTD sends to the inspection system the encrypted version of R using K_{ENC} and K_{MAC} using MUTUAL_AUTENTICATE function of ISO7816-4.

$$resp_data = E[K_{ENC}](R) || MAC[K_{MAC}](E[K_{ENC}](R))$$

8. The inspection system decrypts and verifies the received data.
9. Both, the inspection system and MRTD, create a 16 bytes key seed $K_{IFD/ICC}$, which is the exclusive or between K_{ICC} and K_{IFD}

$$K_{IFD/ICC} = K_{ICC} \oplus K_{IFD}$$

10. Both, the inspection system and MRTD, compute the 16 byte Encryption and Authentication Session Keys (KS_{ENC} and KS_{MAC}) using $K_{IFD/ICC}$ as described previously in section “Compute session keys from seed key $K_{IFD/ICC}$ ”.

2.2.2.7. Secure Messaging

The Secure Messaging should be implemented for the communication of Application Protocol Data Units (APDUs) between MRTDs and the inspection systems.

Secure Messaging has been specified according to *CWA 14890-1: 2004, Application Interface for smart cards used as Secure Signature Creation Devices* [20]. It is required for Passive Authentication and it is optional for Active Authentication.

Figure 2.4 shows an example of Secure Messaging using session keys KS_{ENC} and KS_{MAC} computed previously.

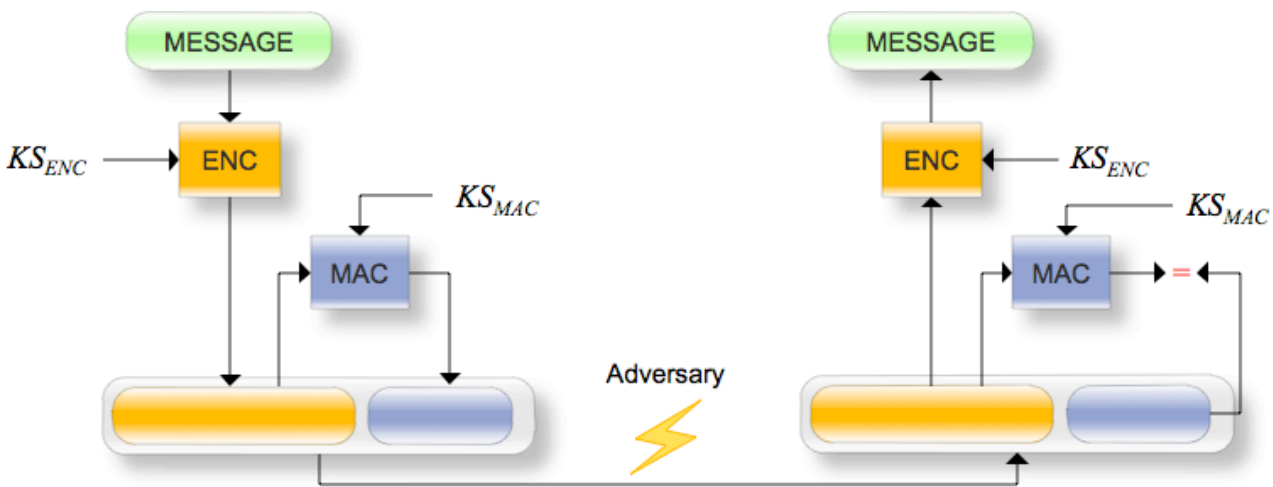


Figure 2.4: Secure Messaging

2.2.2.8. Security Methods Comparison

Besides Passive Authentication by Digital Signatures, additional security methods must be used in order to protect the chip and its data. Table 2.14 shows the advantages and disadvantages of each security method described in this section.

METHOD	ADVANTAGES	DISADVANTAGES
Passive Authentication	<ul style="list-style-type: none"> Proves that the content of SO_D and LDS are authentic 	<ul style="list-style-type: none"> Does not prevent exact copy or chip substitution Does not prevent unauthorized access Does not prevent skimming

METHOD	ADVANTAGES	DISADVANTAGES
Active Authentication	<ul style="list-style-type: none"> • Prevents copying of SO_D and proves that it has been read from authentic chip • Proves that chip has not been substituted 	<ul style="list-style-type: none"> • Requires processor chips • Adds complexity
Basic Access Control	<ul style="list-style-type: none"> • Prevents skimming. • Prevents eavesdropping 	<ul style="list-style-type: none"> • Does not prevent exact copy or chip substitution (requires also copying of conventional document) • Adds complexity • Requires processor chips
Extended Access Control	<ul style="list-style-type: none"> • Prevents unauthorized access to additional biometrics • Prevents skimming of additional biometrics 	<ul style="list-style-type: none"> • Requires additional Key management • Does not prevent exact copy or chip substitution (requires also copying of conventional document) • Adds complexity • Requires processor chips

Table 2.14: Security Methods Comparison.

Chapter 3

3. Description and Justification of Used Technologies

In order to provide an application and allow the user to emulate and evaluate an ePassport applet, we consider the following technologies.

3.1. Smart Cards

A smart card is a portable and tamper-resistant computer. Smart cards have both, processing power and information. The physical appearance and properties of smart card are defined in the standard ISO 7816-1 [27].

3.1.1. Smart Card Memory System

Smart cards usually contain three types of memory: ROM, EEPROM, and RAM. The ROM memory is used for storing the fixed program of the card and contains the operative system routines as well as permanent data and user applications.

The EEPROM memory is similar to ROM memory in that it can preserve information when the power of the memory is turned off. The main difference is that the content of this memory can be modified. We could compare EEPROM memory with the hard disk on a PC.

The RAM memory is used as temporary working space for storing and modifying data.

3.1.2. Smart Card Communication

The communication model of smart cards is half-duplex, which means that the data can be send either from the host to the card or from the card to the host.

Figure 3.1 shows a basic communication model.

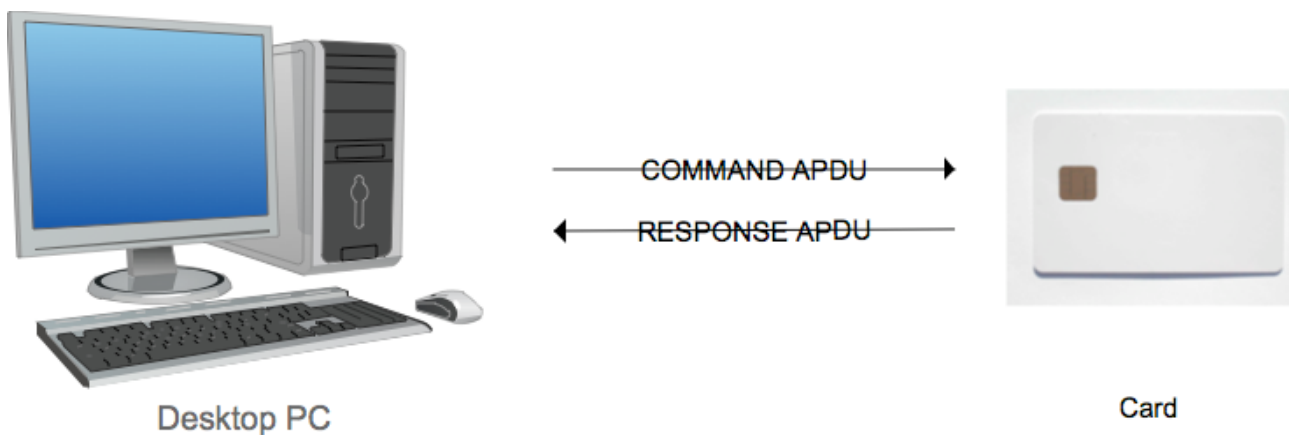


Figure 3.1. Smart Card Communication Model

Smart cards use a communication protocol, specified in ISO 7816-4[1], to communicate between a host and a smart card. The structure of a Command Application Protocol Data Unit (C-APDU) and a Response Application Protocol Data Unit (R-APDU) is shown in Table 3.1 and 3.2.

Mandatory Header				Optional Body		
CLA	INS	P1	P2	Lc	Data Field	Le

Table 3.1. Command APDU Structure

Optional Body	Mandatory Trailer	
Data Field	SW1	SW2

Table 3.2. Response APDU Structure

The command APDU consists of a header of 4 bytes, the data field and an optional Lc byte:

CLA: class of instruction.

INS: instruction code.

P1 and P2: parameters.

Lc: length of data field

Le: length of response data field

The response of an APDU command consists of an optional data field and two mandatory bytes called SW1 and SW2. The values of SW1 and SW2 depend on the type of response sent by the applet. The standard responses are defined in ISO 7816-4[1].

An example of a response APDU is the two bytes “0x9000”, which means that the command has been executed successfully and completely.

3.1.3. Smart Card File System

ISO 7816-4 defines a hierarchical file system structure, which supports three types of files: Master File (MF), Dedicated File (DF) and Elementary File (EF) as shown in Figure 3.2.

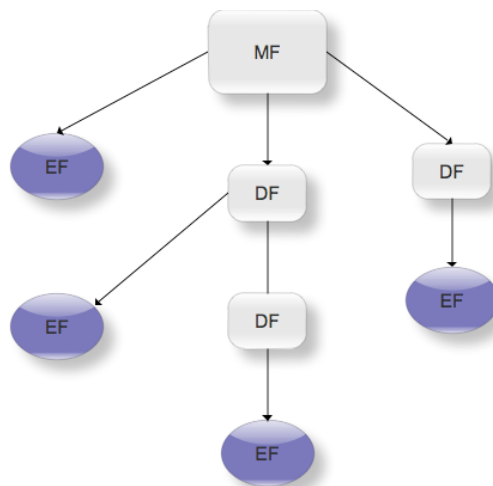


Figure 3.2. File System Structure

The MF is the root of the file system; the DF is a smart card directory; and the EF is the data file.

3.2. Java Card Technology

Java Card Technology allows Java-based applications (called applets) to run on smart cards including memory and processing capabilities, essentially Java Card Technology. This project uses the distribution of Sun Microsystems [36] for smart cards (Java Card Development Kit V2.2.1 and V2.2.2) in order to develop an ePassport applet according to ICAO specifications.

In order to develop the ePassport applet, we use the plugin EclipseJCDE [37] which is used together with the IDE EasyEclipse [34]. The installation process of EasyEclipse and EclipseJCDE is shown in Annex B.

3.2.1. Specifications

The Java Card Technology defines three parts: Java Card Virtual Machine (JCVM), Java Card Runtime Environment (JCRE) and Java Card Application Programming Interface (API).

3.2.1.1. Java Card Virtual Machine

The JCVM defines a subset of Java programming language. JCVM is implemented as two separated pieces as shown in Figure 3.4. The on-card portion of JCVM includes the Java Card byte-code *interpreter*. The Java Card *converter* runs on a PC, converts and loads the class files generating a CAP (converted applet) file.

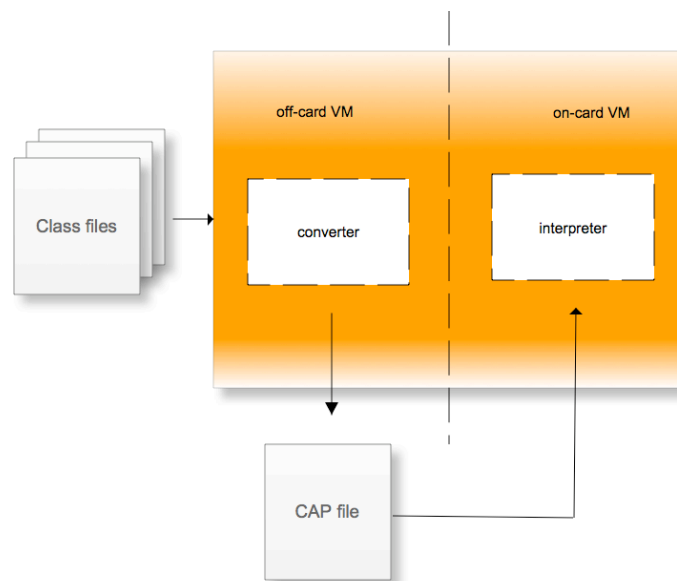


Figure 3.3. Java Card Virtual Machine.

3.2.1.2. Java Card Runtime Environment

The JCRE is the Java Card system that runs inside the card. It is responsible for card resource management, network communication, applet execution and security.

3.2.1.3. Java Card Application Programming Interface

The Java Card API is a set of classes for programming smart cards according to ISO 7816 model. The API contains three packages and one extension package:

- Java API package:
 - java.lang
- Java Card specific packages:
 - javacard.framework
 - javacard.security
 - javacardx.crypto (extension package)

Figure 3.4 shows the architecture of Java Card Technology, which includes the three components mentioned above.

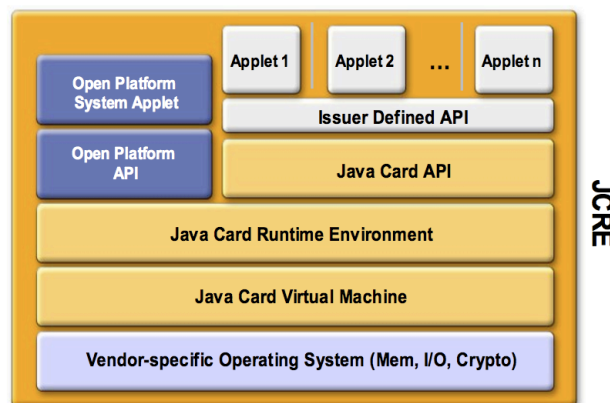


Figure 3.4. Java Card Architecture.

3.2.2. Java Card Evolution

The Smart Card Technology has been evolving over time and becoming more powerful, increasing the memory management and processing capabilities. Table 3.3 shows the evolution of Java Card Platforms over time.

Year	Version	Details
1996	---	Introduction of Java Card Technology
1997	Java Card 2.0	Technology Foundations
1999	Java Card 2.1	Interoperable File Format
2000	Java Card 2.1.1	Additional Crypto APIs
2002	Java Card 2.2	Next Generation Crypto, memory management
2003	Java Card 2.2.1	Enhancements for USIM
2004	Java Card S	Entry Level Fixed Function cards
2006	Java Card 2.2.2	ETSI and Contactless
2008	Java Card 3.0	“Classic” and “Connected”

Table 3.3. Java Card Evolution.

In this project, two versions of Java Card Technologies have been used. The version of Java Card 2.2.1 is used in the ePassport applet installed on the card; and the version of Java Card 2.2.2 is used in the implementation using Java Card Workstation Development Environment (JCWDE) simulator.

3.3. Java

One of the most important decisions to be taken before starting the development of the prototype is the programming language. The requirements of the application to be developed are:

- Graphical User Interface (GUI)
- Communication with Data Base
- Communication with Java Card Applet

In order to fulfill all requirements, Java Version 1.6 is used for the implementation. Table 3.4 shows the packages that are part of Java 1.6.

Package	Usage
Swing	Graphical User Interface
JDCB	SQL database management
Smart Card I/O API	Java Card Management

Table 3.4. Java Packages.

In order to develop the Inspection Systems applet, we used the IDE EasyEclipse Desktop Java [38], which is used for the development of Desktop GUI applications with Swing or SWT.

3.3.1. Swing

Swing is a Java toolkit used to create Graphical User Interfaces (GUI) for Java programs. Figure 3.5 shows a basic example of Swing API utilization.

```
import javax.swing.JFrame;
import javax.swing.JLabel;

public class HelloWorld {
    public static void main(final String[] args) {
        JFrame frame = new JFrame("Hello, World!");
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.add(new JLabel("Hello, World!"));
        frame.pack();
        frame.setVisible(true);
    }
}
```

Figure 3.5. Utilization of Swing API.

3.3.2. JDBC

The JDBC API provides connectivity between Java Programming and a wide range of Databases. JDBC driver is used in this project due to the fact that it is distribution free. Figure 3.6 shows a basic example JDBC API utilization

```
import java.sql.*;

public class Connect{
    public static void main (String[] arg) throws SQLException{
        String User_Name = "root";
        String Password = "root";
        String url = "jdbc:mysql://localhost/ePassport_DataBase";
        Connection conn = null;

        // Select Driver
        Class.forName ("com.mysql.jdbc.Driver").newInstance ();
        conn = DriverManager.getConnection(url, userName, password);
        System.out.println ("Database connection established");
    }
}
```

Figure 3.6. Utilization of JDBC API.

3.3.3. Smart Card I/O API

Smart Card I/O API is used for communication with Smart Cards using ISO/IEC 7816-4 APDUs. It allows the interaction of Java applications with applications running on the Smart Card, to store and retrieve data on the card, etc. Figure 3.7 shows a simple example of Smart Card I/O API utilization.

```
import javax.smartcardio.*;

public class TestSmartCardIO {

    public static void main(String[] args) throws CardException, ExecutionException {

        // show the list of available terminals
        TerminalFactory factory = TerminalFactory.getDefault();
        List<CardTerminal> terminals = factory.terminals().list();
        System.out.println("Terminals: " + terminals);

        // get the first terminal
        CardTerminal terminal = terminals.get(0);

        // establish a connection with the card
        Card card = terminal.connect("T=0");
        System.out.println("card: " + card);
        CardChannel channel = card.getBasicChannel();
        byte[] selectAPI = { (byte) 0x00, (byte) 0xA4, (byte) 0x04, (byte) 0x0c,
                            (byte) 0x07, (byte) 0xA0, (byte) 0x00, (byte) 0x00,
                            (byte) 0x02, (byte) 0x47, (byte) 0x10, (byte) 0x01};
        ResponseAPDU r = channel.transmit(new CommandAPDU(selectAPI));
        System.out.println("response: " + toString(r.getBytes()));

        // disconnect
        card.disconnect(false);
    }
}
```

Figure 3.7. Utilization of Smart Card I/O API.

3.4. DataBase MySQL

MySQL is a multi-user, multi-system, and open source Relational DataBase management system. It is build on C and C++ and supports different platforms. MySQL is used in this project due to the fact that it is distribution free.

A Data Base *ePassport_DataBase* is used in this project. It contains 20 tables, each table represent a data group specified by ICAO (see Table 2.2). Each table contains different fields, which contain information required by ICAO.

Table 3.5 shows the structure of the database *ePassport_DataBase*.

ePassport_DataBase		
Table	Field	Type
COM	id	INT
	LDS_Version	TEXT
	Unicode_Version	TEXT
	Data_Group_List	TEXT
DG1	id	TEXT
	Document_Type	TEXT
	Issuing_State_or_Organization	TEXT
	Name_of_Holder	TEXT
	Document_Number	INT
	Check_Digit_Doc_Number	INT
	Nationality	TEXT
	Date_of_Birth	INT
	Check_Digit_DOB	INT
	Sex	TEXT
	Date_of_Expiry_or_Valid_Until_Date	INT
	Check_Digit_DOE	INT
	Optional_Data	TEXT
	Check_Digit_Optional_Data_Field	INT
Composit_Check_Digit	INT	
DG2	id	INT
	Number_of_Face_Biometrics	INT
	Header_version	TEXT
	Biometric_type	TEXT
	Biometric_feature	TEXT
	Creation_date	INT
	Validity_period	INT
	Creator	TEXT
	Format_owner	TEXT
	Format_type	TEXT
	Biometric_data_face	LOB
	id	INT
	Number_of_Finger_Biometrics	INT
	Header_version	TEXT

ePassport_DataBase		
Table	Field	Type
DG3	Biometric_type	TEXT
	Biometric_feature	TEXT
	Creation_date	INT
	Validity_period	INT
	Creator	TEXT
	Format_owner	TEXT
	Format_type	TEXT
	Biometric_data_finger	LOB
DG4	id	INT
	Number_of_Eye_Biometrics	INT
	Header_version	TEXT
	Biometric_type	TEXT
	Biometric_feature	TEXT
	Creation_date	INT
	Validity_period	INT
	Creator	TEXT
	Format_owner	TEXT
	Format_type	TEXT
	Biometric_data_eye	LOB
DG5	id	INT
	Number_of_Portraits	INT
	Image_Portrait	LOB
DG6	id	INT
	Reserved_for_Future	TEXT
DG7	id	INT
	Number_of_Signature_or_Mark	INT
	Image_Signature_or_Mark	LOB
DG8	id	INT
	Number_of_Data_Features	INT
	Header	LOB
DG9	id	INT
	Number_of_Data_Features	INT
	Header	LOB

ePassport_DataBase		
Table	Field	Type
DG10	id	INT
	Number_of_Data_Features	INT
	Header	LONGBLOB
DG11	id	INT
	Tag_list	TEXT
	Name_of_holder	TEXT
	Content_specific	TEXT
	Number_of_other_names	TEXT
	Other_Names	TEXT
	Personal_Number	TEXT
	Date_of_birth	INT
	Place_of_Birth	TEXT
	Address	TEXT
	Telephone_Numbers	TEXT
	Profession	TEXT
	Title	TEXT
	Personal_Summary	TEXT
	Proff_of_Citizenship	TEXT
Other_Valid_Travel_Document	TEXT	
Custody_Information	TEXT	
DG12	id	INT
	Tag_list	TEXT
	Issuing_Authority	TEXT
	Date_of_Issue	INT
	Context	TEXT
	Number_of_other_people	INT
	Other_Person_Included_on_MRTD	TEXT
	Endorsements_Observations	TEXT
	Tax_Exit_Requirements	TEXT
	Image_of_Front_of_MRTD	LONGBLOB
	Image_of_Rear_of_MRTD	LONGBLOB
	Date_and_time_of_perzonalization	INT
	Serial_number	INT

ePassport_DataBase		
Table	Field	Type
DG13	id	INT
	Optional_Details	LONGBLOB
DG14	id	INT
	Reserved_For_Future_Use	LONGBLOB
DG15	id	INT
	Active_Authentication_Key	LONGBLOB
DG16	id	INT
	Number_of_templates	INT
	Date_data_recorded	INT
	Names_of_Persons_to_NotifyCopy	TEXT
	Telephone	TEXT
	Address	TEXT
SOD	id	INT
	Key	LONGBLOB
KENC	Encryption Key	VARBINARY(16)
KMAC	Authentication Key	VARBINARY(16)

Table 3.5. ePassport_DataBase.

3.5. GPSHELL Framework

GPSHELL Framework [21] is a script interpreter, which talks to a smart card. It is written on top of the GlobalPlatform library, which is a standard for the management of the contents on a smart card.

GPSHELL uses smart card communication protocols ISO 7816-4 and OpenPlatform 2.0.1 and GlobalPlatform 2.1.1. It can establish a secure channel with a smart card, load, instantiate, delete, and list applets on a smart card.

In this project GPSHELL is used to install an ePassport applet build using Java Card Development Kit version 2.2.1 in a smart card Gemalto TOP IM GX4 [24]. We selected GPSHELL Framework because it is distribution free.

ANNEX A shows the installation process of GPSHELL for MAC OS Leopard, and the management of GPSHELL in order to install an applet in a smart card Gemalto TOP IM GX4, which is used in this project.

3.6. Alya

ALYA [23] (see Figure 3.8) is a solution built by the association of biometric (fingerprint reader) and a smart card reader, which allows the user to access a workstation through the fingerprint reader, access all Microsoft Windows based applications and protect confidential files and folder.

This product is distributed by Covadis S.A. [22], and is used in this project only as a reader in order to get access to an ePassport applet installed in a smart card Gemalto TOP IM GX4.



Figure 3.8. ALYA

3.7. Application of Technologies

Figure 3.9 shows the application of technologies used in this project in order to build a system, which can write and read in an ePassport applet using a database in order to update the information of users.

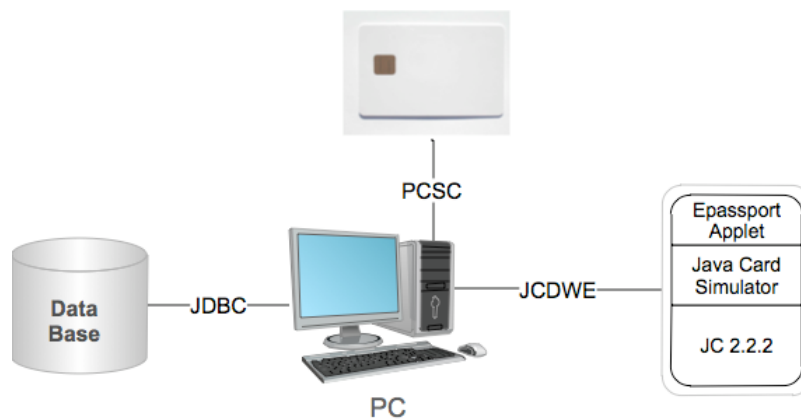


Figure 3.9. Application of Technologies

Chapter 4

4. Description of Software Developed

The software developed in this project is a system, which can write and read in a ePassport applet according to the ICAO specifications for Machine Readable Travel Documents.

4.1. Architecture

The software developed in this project consists of three main graphical interfaces: Database manager, JCWDE (Java Card Workstation Development Environment) simulator, and Reader manager.

Each graphical interface consists of two managers, which allow for using the following functions:

- Write information in Database
- Read information from Database
- Configure a JCWDE simulator
- Read information of an ePassport applet using JCWDE simulator
- Write information of users from a database in an ePassport applet installed in a Smart Card.
- Read information of ePassport applet installed in a Smart Card

Figure 4.1 shows a tree structure that represents how the user can navigate through the application. The boxes represent the application screens and the arrows represent the navigation paths.

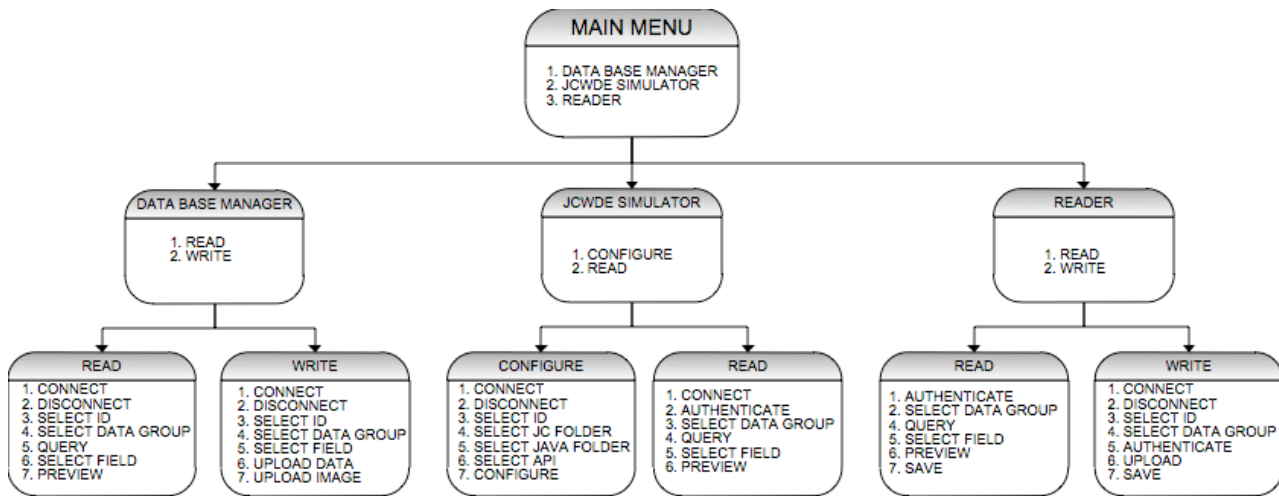


Figure 4.1. System Architecture

4.2. Database Manager

The Database Manager allows the user to update information in the database *ePassport_DataBase*, which contains 18 tables and each table includes the data fields according to ICAO specifications (refer to section 3.4 DataBase MySQL). The Database Manager has two graphical interfaces: Read, and Write (refer to Figure 4.1).

4.2.1. Database Connection

The graphical interfaces: Read database, Write database, Configure JCWDE simulator and Write Reader use connection with the database, in order to select the user to be read or write.

The Database Connection operation sends the information required to get access to MySQL database (username, password, and url), and receives in the graphical interface the list of ids and name of persons in the database *ePassport_DataBase*. Thus, the user can select the id of the person to be updated, read, or used for other functions.

Figure 4.2 shows the UML (Unified Modeling Language) sequence diagram of the Database Connection.

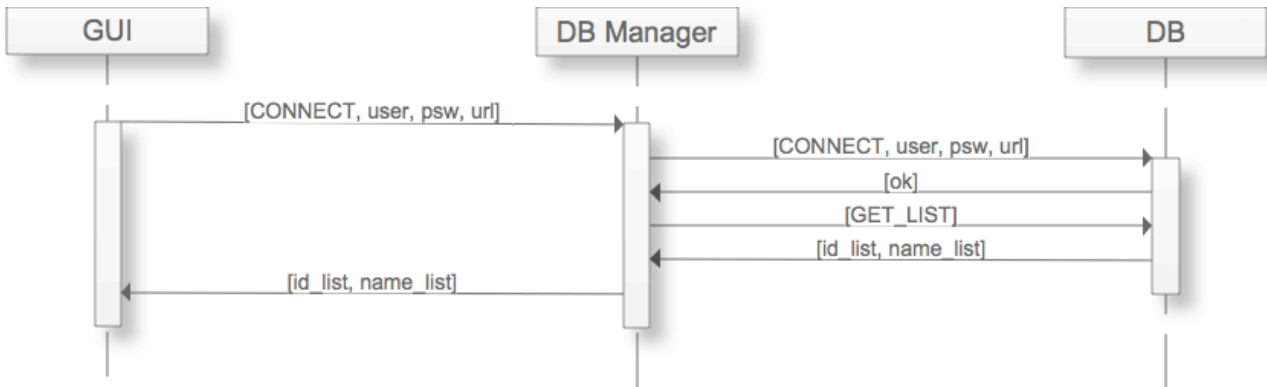


Figure 4.2. Database Connection.

4.2.2. Get Data from Database

In order to read information from the MySQL database, the user have to select the Id and Data Group to be read. Then clicking the Query button, the DB manager gets access to the database and retrieves the information of the specific Id and Data Group. This information is displayed as a list of Fields with its content in the GUI.

Figure 4.3 shows the UML sequence diagram of operation Get Data from Database.

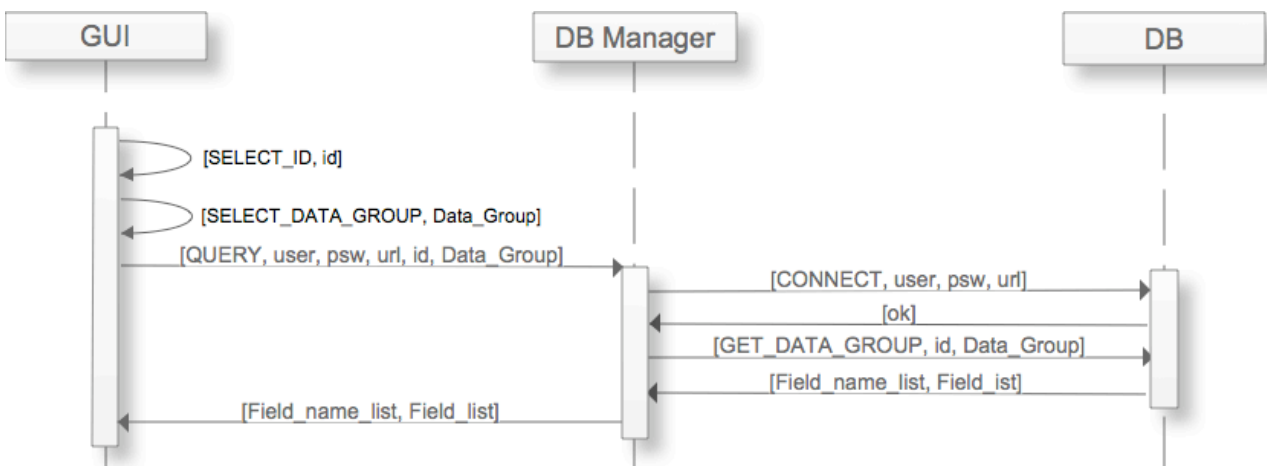


Figure 4.3. Get Data from Database.

4.2.3. Get Image from Database

The Data Groups 2, 3, 4, 5, and 7 contain images according to ICAO specifications. In order to display the images, the user has to perform first the operation Get Data from Database. Afterwards, the GUI displays a new option, which allows the user to select the name of the Filed, which contains an

image. Thus, the user can display the image, which is saved in one of the Fields in the selected Data Group.

Figure 4.4 shows the UML sequence diagram of operation Get Image from Database.

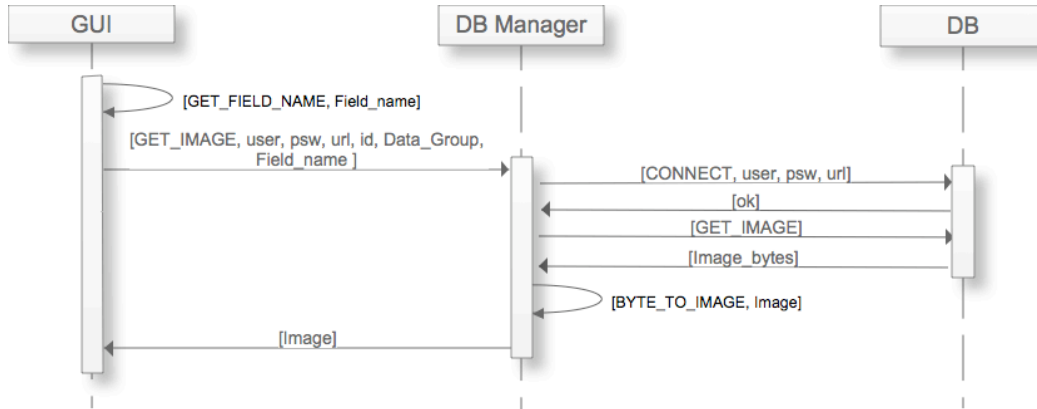


Figure 4.4. Get Image from Database.

4.2.4. Upload Data to Database

In order to update text fields in the database ePassport_DataBase, the user can use the Interface Write of Database Manager.

Before using the operation Upload Data to Database, the user has to perform the operation Database Connection. Afterwards, the user has to select the Id, Data Group, and Field to be updated with the value inserted in the text field.

Figure 4.5 shows the UML sequence diagram of operation Upload Data to Database.

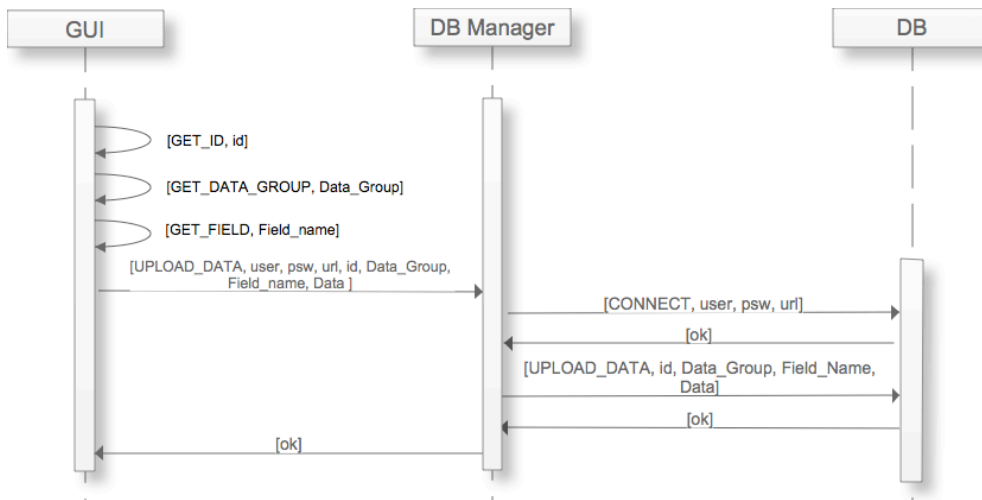


Figure 4.5. Upload Data to Database.

4.2.5. Upload Image to Database

In order to update Image fields in the database ePassport_DataBase, the user can use the Interface Write of Database Manager.

Before using the operation Upload Image to Database, the user has to perform the operation Database Connection. Afterwards, the user has to select the Id, Data Group, and Field to be updated. Then, the user has to select the Image to be uploaded using a graphical selector.

Figure 4.5 shows the UML sequence diagram of operation Upload Image to Database.

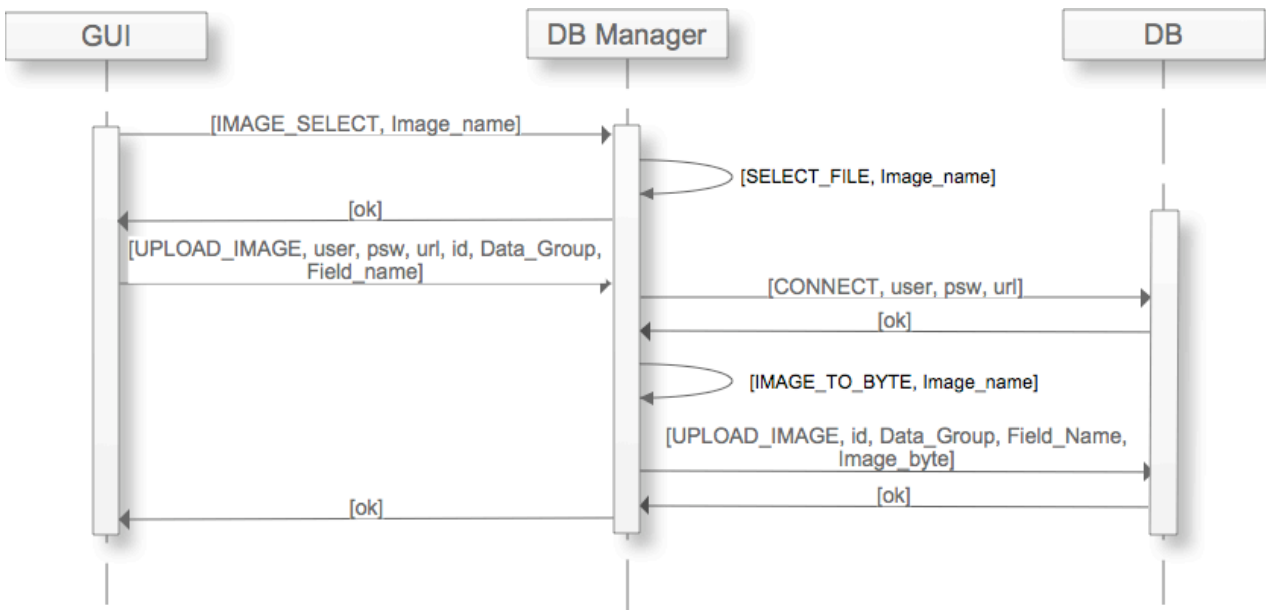


Figure 4.5. Upload Data to Database.

4.3. JCWDE Simulator

The JCWDE simulator interface allows the user to test the ePassport applet using the JCWDE simulator included in the Java Card Development Kit version 2.2.2. It has two graphical interfaces: Configure, and Read (refer to Figure 4.1).

4.3.1. Configure

The Configure interface is used for the user in order to prepare the FileSystem.java file used by the ePassport applet, and to setup environmental variables needed by JCWDE simulator.

Firstly, the user gets connection with the database and the list of Id's and names of persons is displayed in the GUI. Then, the user selects the Id of a user.

Afterwards, the user selects the applet ePassport, the Java File, and the Java Card Development Kit using a file selector.

Finally, the user executes Configure in the GUI, in order to get access to the information of the selected Id from the database. Then, the JCWDE manager creates the FileSystem.java needed by the ePassport applet, and compiles the ePassport applet and the FileSystem.java using Java Card Kit previously selected.

Finally, the JCWDE manager generates the configuration file needed by JCWDE manager, and sends an "OK" response to the GUI.

Figure 4.6 shows the UML sequence diagram of interface Configure.

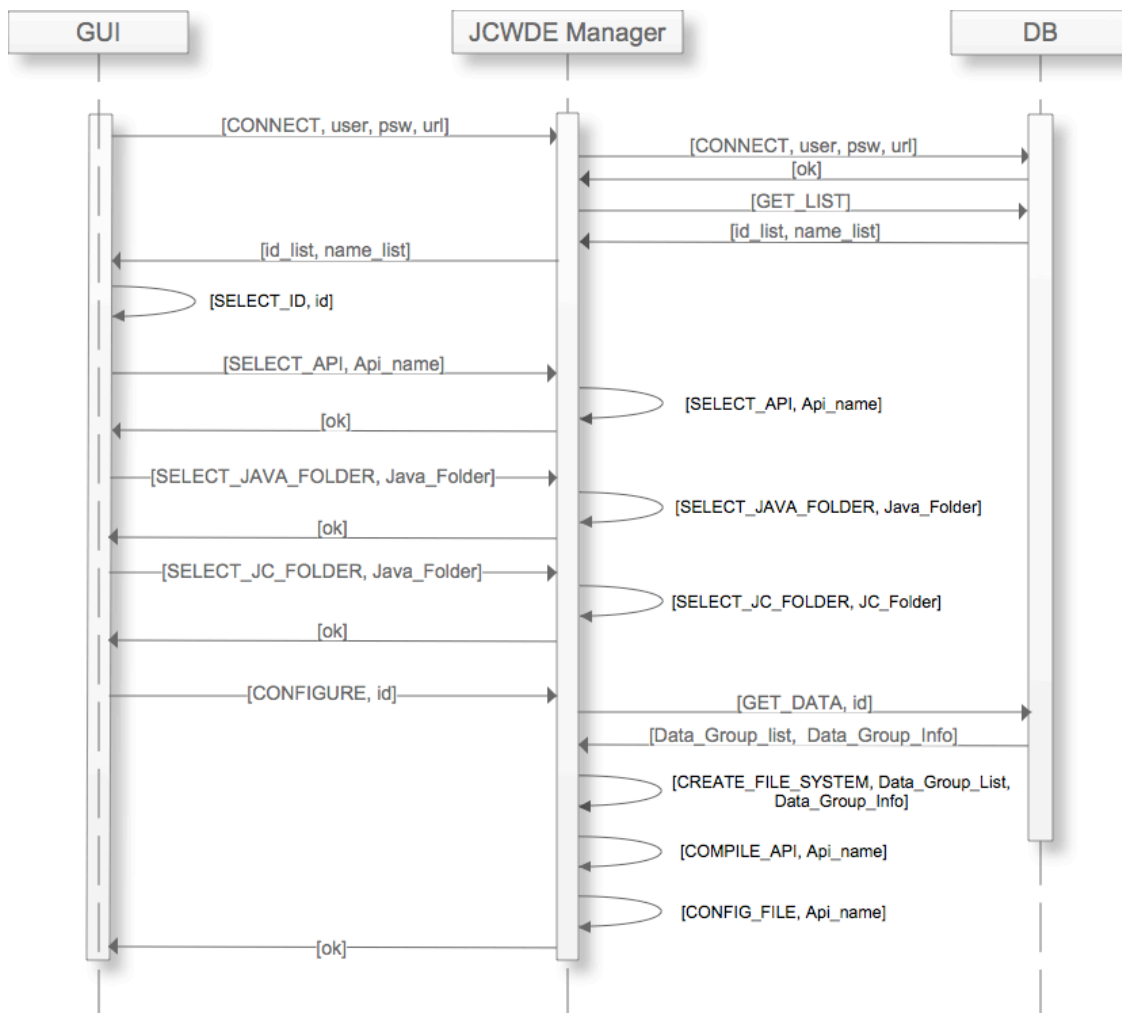


Figure 4.6. Configure.

4.3.2. Get Data from JCWDE Simulator

In order to read information using the JCWDE simulator, which is included in the Java Card Development Kit, the user has to perform the next executions:

First, the user selects the Data Group to be read. Second, the user has to run the JCWDE simulator using the Connect button.

Finally, the user test the ePassport applet previously selected using the button Test. The information of the specific Data Group is displayed as a list of Fields with its content in the GUI.

Figure 4.7 shows the UML sequence diagram of operation Get Data from JCWDE simulator.

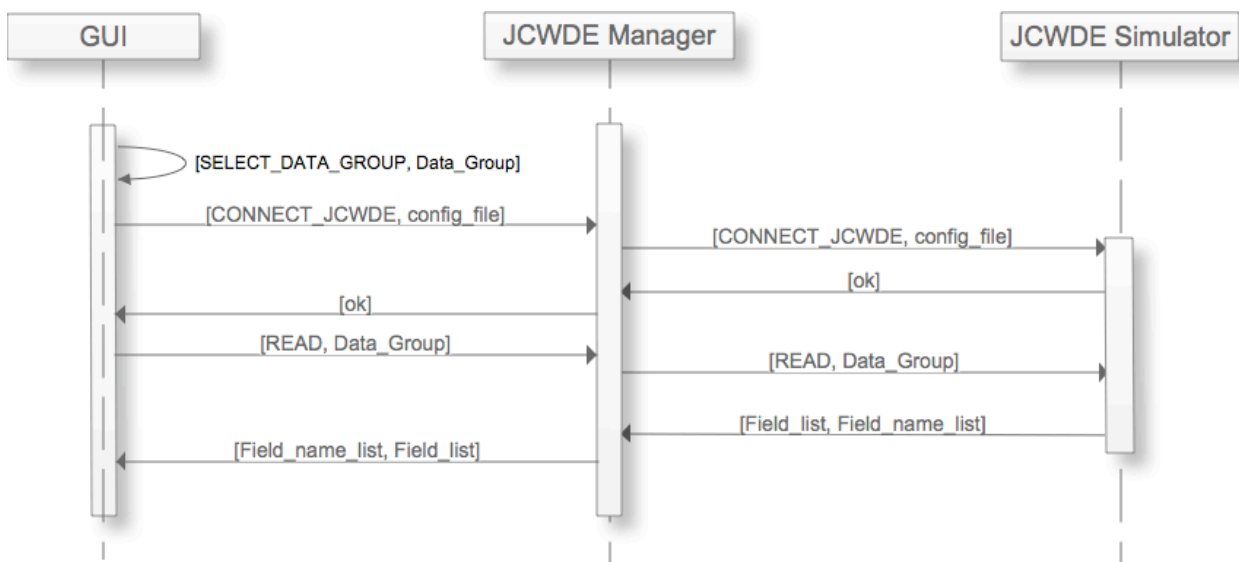


Figure 4.7. Get Data from JCWDE simulator.

4.3.3. Get Image from JCWDE Simulator

The Data Groups 2, 3, 4, 5, and 7 contain images according to ICAO specifications. In order to display the images, the user has to perform first the operation Get Data from JCWDE. Afterwards, the GUI displays a new option, which allows the user to select the name of the Filed, which contains an image. Thus, the user can display the image, which is saved in one of the Fields in the selected Data Group.

Figure 4.8 shows the UML sequence diagram of operation Get Image from JCWDE simulator.

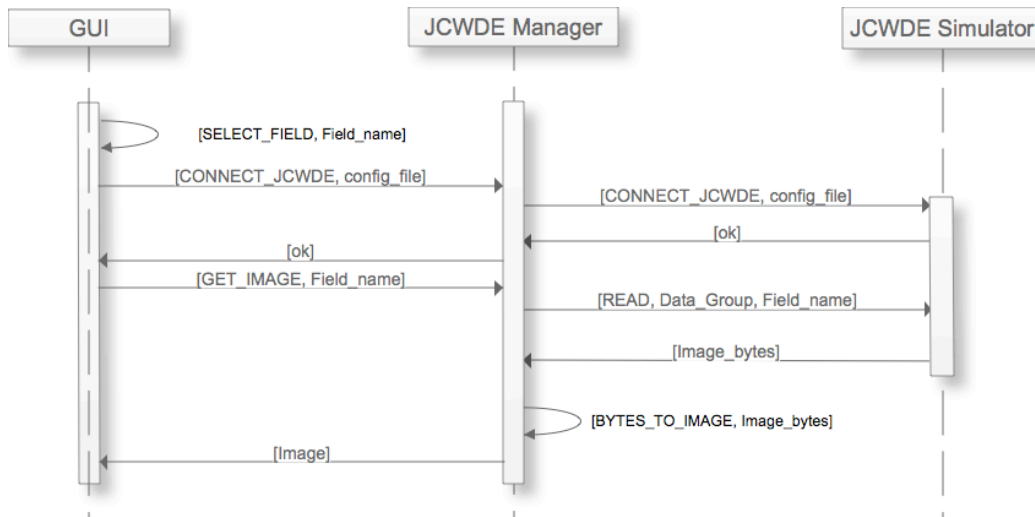


Figure 4.8. Get Data from JCWDE simulator.

4.4. Reader

The Reader interface allows the user to test the ePassport applet installed in a smart card. It has two graphical interfaces: Write, and Read (refer to Figure 4.1).

4.4.1. Write Card from Data Base

The Write Card from Data Base operation allows the user to upload or update information in the File System of the ePassport applet installed in a smart card.

Before uploading information, the user must perform the Database Connection operation (refer to section 4.2.1 Database Connection). Afterwards, the user selects the Id and Data Group to be updated.

Finally, the user executes the Update function. In this function, the Write Card Manager performs the next operations:

First, it connects with the database and retrieves the information of the selected Data Group.

Secondly, it converts the information from the database into a TLV format (refers to section 2.2.1.1 File Structure).

Finally, it sends information in TLV format to the smart card using APDU commands. Additionally the GUI displays the set of APDU commands and responses used in the communications between the Write Card Manager and the smart card. This information could be saved in a text file.

Figure 4.9 shows the UML sequence diagram of operation Write Card from Data Base.

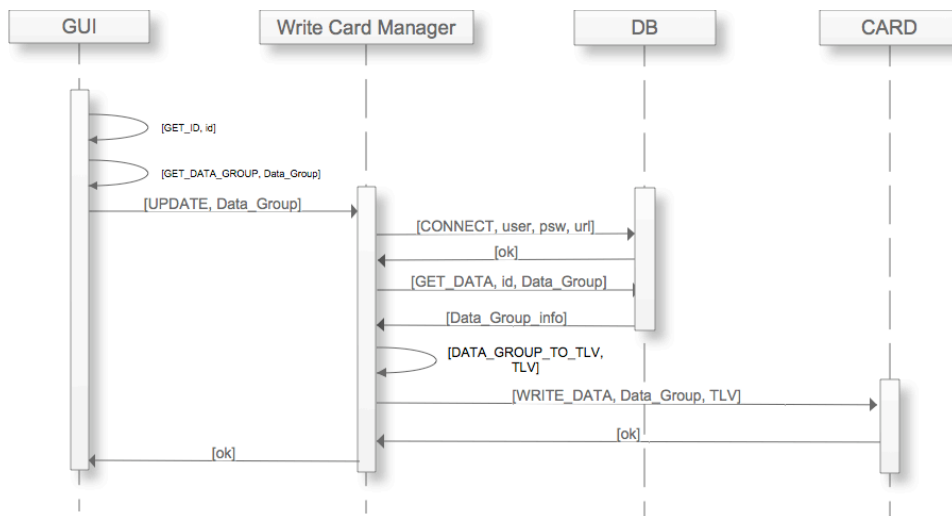


Figure 4.9. Write Card from Data Base.

4.4.2. Read Card

In order to read information from an ePassport applet installed in a smart card, the user has to perform the two operations:

First, the user selects the data group in the GUI, and finally, with the button Query the information saved in the smart card is displayed in the GUI.

Additionally the GUI displays the set of APDU commands and responses used in the communications between the Write Card Manager and the smart card. This information could be saved in a text file.

Figure 4.10 shows the UML sequence diagram of operation Read Card.

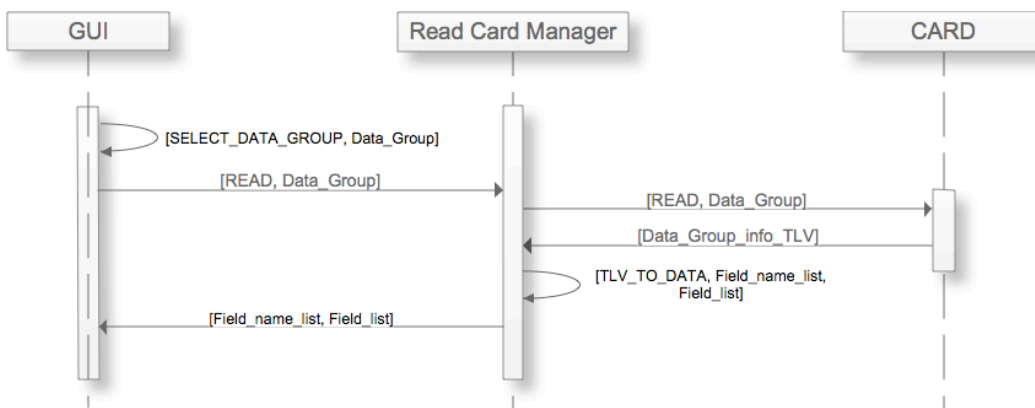


Figure 4.10. Read Card.

4.5. Security

The ePassport applet uses the Basic Access Control defined by ICAO specifications (refer to section 2.2.2.6.2 Basic Access Control) used by ePassport applet, which is by JCWDE Simulator and the smart card.

Figure 4.11 shows the UML sequence diagram for the Security implemented in the ePassport applet.

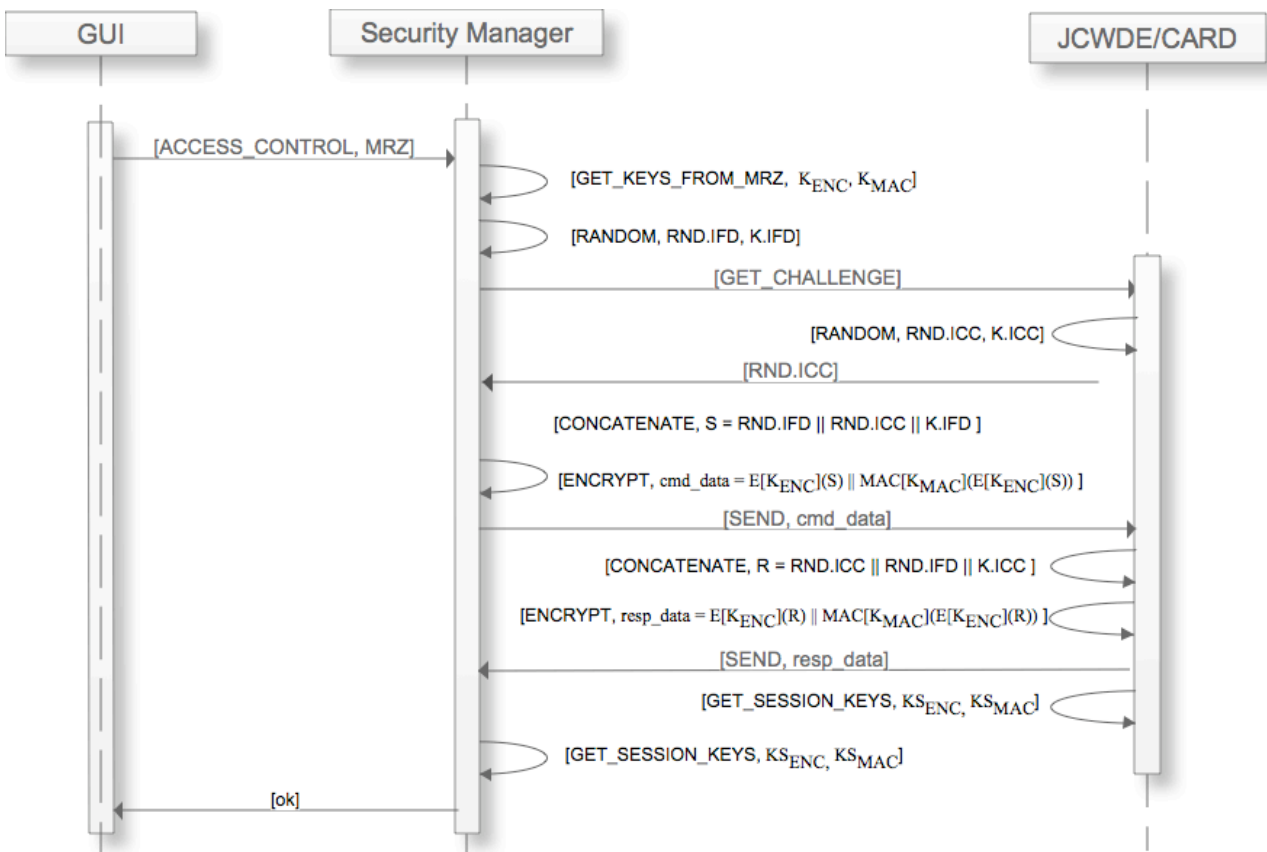


Figure 4.11. Security.

4.6. Graphical Interface

The graphical interface was developed with the IDE Easy Eclipse [34]. It allows the user to interact with the different interfaces described in 4.1 Architecture. ANNEX A shows the utilization mode of the application ePassport.

The GUI consists of six interfaces:

- Read from Database

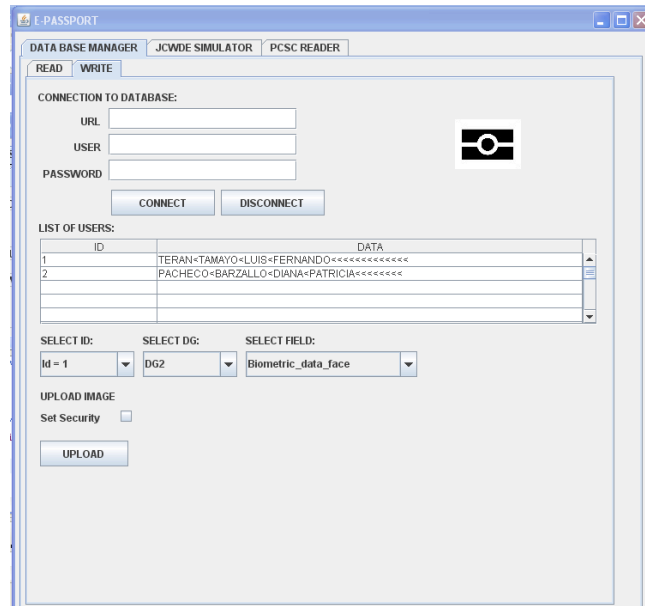


Figure 4.12. Write to Database.

4.6.3. Configure JCWDE Simulator

The GUI Configure JCWDE Simulator allows the user to create the File System required by ePassport applet from database ePassport_DataBase. It also configures the environmental variables; configuration files and compiles the Java Card classes required by JCWDE Simulator.

Figure 4.13 shows the GUI Configure JCWDE Simulator.

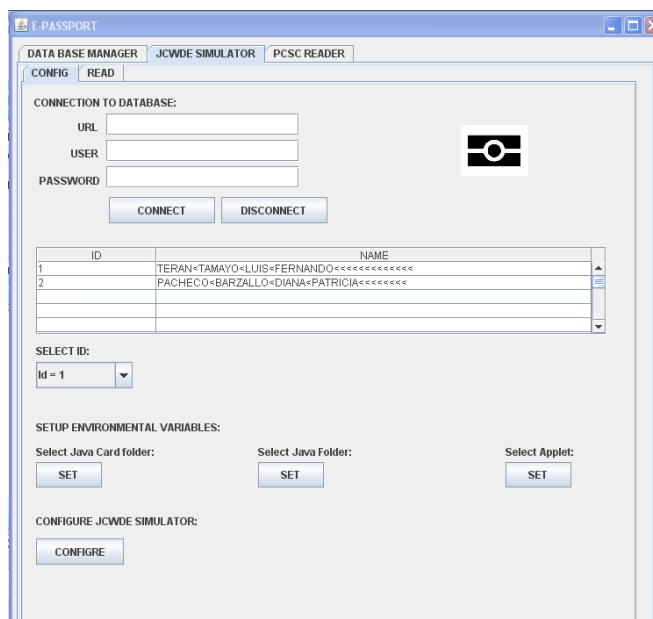


Figure 4.13. Configure JCWDE Simulator

4.6.4. Read from JCWDE Simulator

The GUI Read from JCWDE Simulator allows the user to read the File System generated in Configuration JCWDE Simulator, using the ePassport applet.

Figure 4.14 shows the GUI Read from JCWDE Simulator.

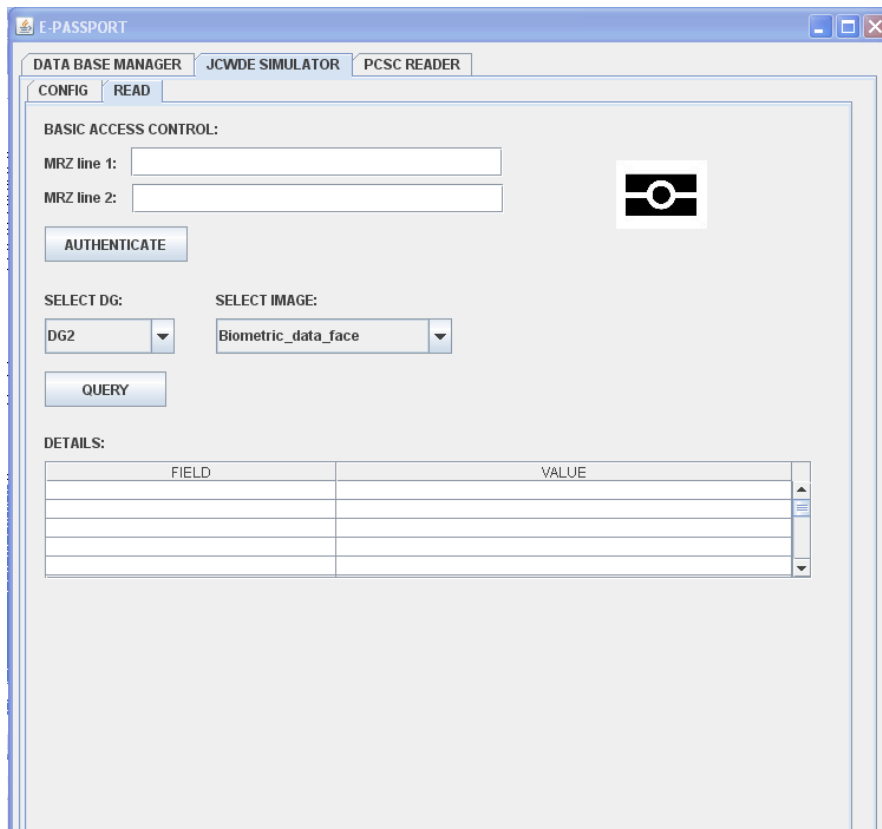


Figure 4.14. Read

4.6.5. Read Card

The GUI Read Card allows the user to read information from a smart card, which has installed the ePassport applet. The user also can save the save in a file text the sequence of commands APDU interchanged between the API and the smart card.

Figure 4.15 shows the GUI Read Card.

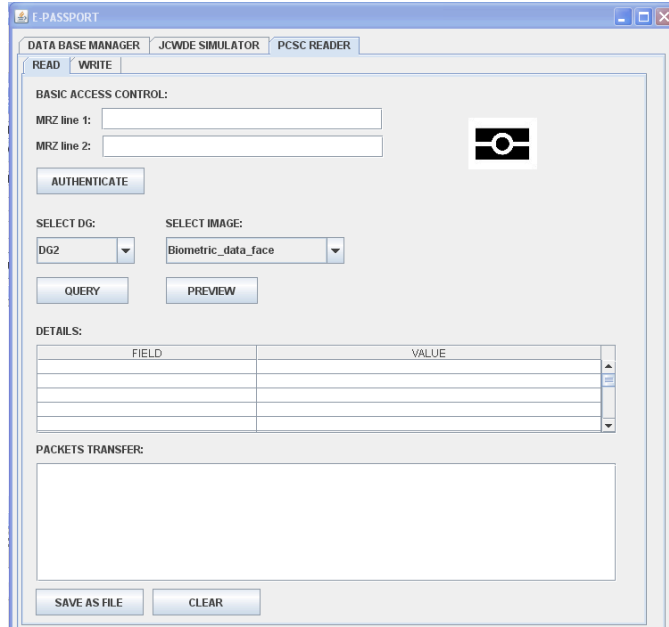


Figure 4.15. Read Card

4.6.6. Write Card from Database

The GUI Write Card from Database allows the user to write information from the ePassport_Database in a smart card, which has installed the ePassport applet. The user also can save the save in a file text the sequence of commands APDU interchanged between the API and the smart card.

Figure 4.16 shows the GUI Write Card from Database.

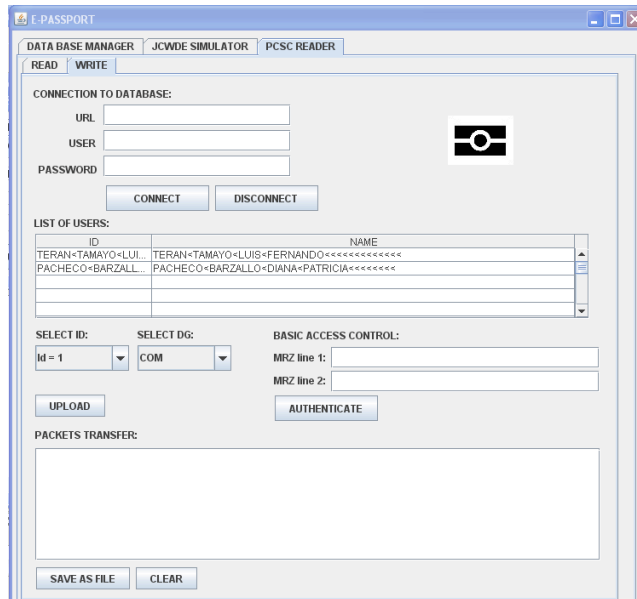


Figure 4.16. Write Card from Database

Chapter 5

5. Conclusion and Future Work

In this chapter, the conclusion of this master project is presented along with major findings and proposals for possible future work.

5.1. Conclusions

In this project it was designed an Inspection System for Biometric Passports according to ICAO specifications. The resulting system covers most of the user requirements, for instance: upload and update of information, verification, and security.

The ePassport applet was built using Java Card Development Kit version 2.2.1, which has limitations like: management of APDU commands, less cryptographic algorithms, etc.

The current system can test an ePassport applet with or without a real smart card using the JCWDE simulator included in the Java Card Development Kit 2.2.2.

5.2. Future Work

The current implementation of an Inspection System is limited, and could be enhanced with additional features to further improve the system capabilities.

The usage of CREF (C reference implementation of Java Card run-time environment) simulator, which is a more sophisticated card emulator, can be used in future work in order to enable card state saving (card EEPROM image). This CREF simulator can be used instead of JCWDE simulator.

5.2.1. Performance Improvement

In order to improve the performance of the system is important to consider new versions of Java Card Development Kit (version 2.2.2), which provides management of extended APDU commands.

5.2.2. Database Improvement

Future work can enhance the management of database providing the user the capability of create and delete persons in the database. It also could be used to administrate in a new fashion the management of Certificates and Public Keys.

5.2.3. Security Improvements

In order to improve the security of the system is recommended to consider a new version of Java Card Development Kit (version 2.2.2), which provides better cryptographic algorithms.

Annex A

INSTALLATION OF GPSHELL FOR LEOPARD MAC OS 10.5

1. Download and install the following libraries:

OpenSSL <http://www.openssl.org/>

Zlib <http://www.zlib.net/>

Important: GPSHELL also needs PC/SC Lite but it is included in Leopard. Do not install this library.

2. Download gpshell-1.4.2 and globalplatform-5.0.0 from:

http://sourceforge.net/project/showfiles.php?group_id=143343

3. Unzip both

4. Edit configure file, which comes with globalplatform as mentioned next:

Comment out the following lines:

* 21019,21024,21026 to 21032 (for pcsclite)

* 21256,21263 to 21269 (for zlib)

* 21490,21497 to 21503 (for openssl)

5. Using command line, move to the directory `globalplatform-5.0.0` and execute the following commands:

```
./configure PCSCLITE_CFLAGS=-I../globalplatform-5.0.0/pcsclite-includes/ PCSCLITE_LIBS="-framework PCSC"  
make  
make install
```

6. Using command line, move to the directory `gpshell-1.4.2` and execute the following commands:

```
./configure PCSCLITE_CFLAGS=-I../globalplatform-5.0.0/pcsclite-includes/ PCSCLITE_LIBS="-framework PCSC"  
make
```

INSTALLATION OF API IN GEMALTO XPRESS PRO USING GPSHELL ON LEOPARD MAC OS 10.5

In order to install an API in a Gemalto Xpress Pro using GPSHELL under Leopard, it is required to compile your API and create your CAP file using Java Card Development Kit 2.2.1. and the version of Java 1.4 for mac OS. This procedure is shown next:

1. Download Java Card Development Kit 2.2.1 from:

<http://java.sun.com/javacard/downloads/index.jsp>

2. Unzip the file and save in a secure place, in this example we save the content of the Java Card Kit 2.2.1 in the file `jc221`, which is in the following directory:

```
/JavaCard/jc221
```

Important: we use this path for the environmental variables; you must change the variables if you save the Java Card kit in another directory.

3. Java Card Development Kit 2.1.1 requires setup environmental variables and the path of the binaries of your Java Card Development Kit. We show how to setup variables statically editing the "profile" file as is shown next:

- 3.1. Using command line type the following commands:

```
cd /etc  
sudo vi profile
```

Important: you have to have administrative privileges to modify the "profile" file

3.2. Using vi, add the following lines in your "profile":

```
export JAVA_HOME=/Library/Java/Home
export JC_HOME=/JavaCard/jc221
```

Important: the value of variable JC_HOME depends on where did you save your java card kit and the name of file.

3.3. Edit the variable PATH, adding the location of the folder called "bin", which is saved in your Java Card Development Kit. In this example, we added the path /JavaCard/jc221/bin/ at the end of the variable PATH as shown next:

```
export PATH=$PATH:/opt/local/bin:/opt/local/sbin:/JavaCard/jc221/bin/
```

3.4. Save your changes

Important: if you want to test your modification you can use the command "export" in the command line.

4. Compile your API.java:

Using command line, type the following commands:

```
export JAVA_JVM_VERSION=1.4
javac -g -classpath ./classes:/JavaCard/jc221/lib/api.jar:/JavaCard/jc221/lib/installer.jar /PATH-OF-API/*.java
```

Important: The first line changes the version of java to java 1.4, which is required. You also must specify correctly your classpath of api.jar and installer.jar, and the path where you saved your API.java

5. Generate your CAP file

Using command line, type the following command (the following 4 lines are only one command):

```
converter -classpath "/PATH-OF-API/" -d "/PATH-OF-API/" -exportpath
"/JavaCard/jc221/api_export_files:/PATH-OF-API/" -out CAP -applet
0xA0:0x00:0x00:0x02:0x47:0x10:0x01 com.epassport.Epassport com.epassport
0xA0:0x00:0x00:0x02:0x47:0x06:0x07:0x08:0x09:0x00 1.0
```

Important: this example shows how to convert a java card API called Epassport.java, which is saved in: com/epassport/ (this is also the package of the applet). In order to convert your class file you must specify your own applet ID (0xA0:0x00:0x00:0x02:0x47:0x10:0x01), applet name (com.epassport.Epassport), package name (com.epassport), and package ID (0xA0:0x00:0x00:0x02:0x47:0x06:0x07:0x08:0x09:0x00).

The last value in the command (1.0) is the version.

6. Install the cap file using GPshell script as is described next:

- 6.1. Using command line, move to the directory where you saved your CAP file.
- 6.2. Create a Script file using any text editor with the following lines, and save it as a .txt file.
(In this example we save the file as Install.txt)

```
mode_201
gemXpressoPro
enable_trace
establish_context
card_connect
select -AID A000000018434D00
open_sc -security 3 -keyind 0 -keyver 0 -key 47454d5850524553534f53414d504c45 // Open secure channel
install -file epassport.cap -sdAID A000000018434D00 -nvCodeLimit 4000
card_disconnect
release_context
```

Important: in your script you must specify the name of your cap file. In this example we use the epassport.cap.

- 6.3. Using command line, type the following command:

```
gpshell Install.txt
```

7. In order to check the installation of your applet, you can use the file listgemXpressoProR3_2E64.txt, which comes with gpshell using the following comand:

```
gpshell listgemXpressoProR3_2E64.txt.
```

Your application must be present in the list of applications installed in your card.

Authors:

- Lois Lherbier (COVADIS S.A.)
- Luis Terán (EPFL)

Annex B

INSTALLATION OF EASYECLIPSE DESKTOP JAVA

1. Download the package EasyEclipse Desktop Java from the following link:

<http://www.easyeclipse.org/site/distributions/index.html>

2. Run the installation package of EasyEclipse Desktop Java

INSTALLATION OF ECLIPSEJCDE PLUGIN FOR IDE ECLIPSE

1. Follow the installation process of EclipseJCDE from the following link:

<http://eclipse-jcde.sourceforge.net/>

Bibliography

- [1] ISO. *Identification cards - Integrated circuit cards with contacts - Part: 4. Organization, security and commands for Interchange*. ISO/IEC 7816-4. International Organization for Standardization, Geneva, Switzerland, 2005.
- [2] ISO. *Identification cards - Integrated circuit cards with contacts - Part: 6. Interindustry data elements for interchange*. ISO/IEC 7816-6. International Organization for Standardization, Geneva, Switzerland, 2004.
- [3] ISO. *Identification cards - Integrated circuit cards with contacts - Part: 8. Commands for security operations*. ISO/IEC 7816-8. International Organization for Standardization, Geneva, Switzerland, 2004.
- [4] ISO. *Identification cards - Integrated circuit cards with contacts - Part: 9. Commands for card management*. ISO/IEC 7816-9. International Organization for Standardization, Geneva, Switzerland, 2004.
- [5] ISO. *Identification cards - Integrated circuit cards with contacts - Part: 11. Personal Verification through biometric methods*. ISO/IEC 7816-11. International Organization for Standardization, Geneva, Switzerland, 2004.
- [6] ISO. *Identification cards - Integrated circuit cards with contacts - Part: 15. Cryptographic information application*. ISO/IEC 7816-15. International Organization for Standardization, Geneva, Switzerland, 2004.
- [7] ISO. *Information technology - Biometric data interchange formats - Part 1: Framework*. ISO/IEC 19794-1. International Organization for Standardization, Geneva, Switzerland, 2006.
- [8] ISO. *Information technology - Biometric data interchange formats - Part 2: Finger minutiae data*. ISO/IEC 19794-2. International Organization for Standardization, Geneva, Switzerland, 2005.
- [9] ISO. *Information technology - Biometric data interchange formats - Part 3: Finger pattern spectral*

- data*. ISO/IEC 19794-3. International Organization for Standardization, Geneva, Switzerland, 2006.
- [10] ISO. *Information technology - Biometric data interchange formats - Part 4: Finger image data*. ISO/IEC 19794-4. International Organization for Standardization, Geneva, Switzerland, 2005.
- [11] ISO. *Information technology - Biometric data interchange formats - Part 5: Face image data*. ISO/IEC 19794-5. International Organization for Standardization, Geneva, Switzerland, 2005.
- [12] ISO. *Information technology - Biometric data interchange formats - Part 6: Iris Image Data*. ISO/IEC 19794-6. International Organization for Standardization, Geneva, Switzerland, 2005.
- [13] ISO. *Information technology - Biometric data interchange formats - Part 1: Finger pattern skeletal data*. ISO/IEC 19794-8. International Organization for Standardization, Geneva, Switzerland, 2006.
- [14] ISO. *Information technology - Common Biometric Exchange Formats - Framework - Part 1: Data element Specification*. ISO/IEC 19785-1. International Organization for Standardization, Geneva, Switzerland, 2006.
- [15] ISO. *Information technology - Biometric profiles for interoperability and data interchange - Part 1: Overview of biometric systems and biometric profiles*. ISO/IEC 24713-1. International Organization for Standardization, Geneva, Switzerland, 2006.
- [16] International Civil Aviation Organization.
<http://www.icao.int/>
- [17] *Development and Specifications of Globally Interoperable Biometric Standards for Machine Assisted Identity Confirmation using Machine Readable Travel Documents*, V2.0, International Civil Aviation Organization, 2004.
- [18] *Development of a Logical Data Structure for Optional Capacity Expansion Technologies*, V1.7, International Civil Aviation Organization, 2004.
- [19] *PKI for Machine Readable Travel Documents Offering ICC Read-Only Access*, V1.1, International Civil Aviation Organization, 2004.
- [20] *CWA 14890-1: 2004, Application Interface for smart cards used as Secure Signature Creation Devices*, Version 1 release 9 rev. 2, 22 Dec 2003.
- [21] GPSHELL Framework
http://sourceforge.net/project/showfiles.php?group_id=143343
- [22] Covadis S.A.
<http://www.covadis.ch/>
- [23] ALYA.
<http://www.covadis.ch/Alya.239.0.html>
- [24] Gemalto card TOP GX4.
http://www.gemalto.com/products/top_javacard/download/TOP_GX4_Nov08.pdf
- [25] Java Card Smart Cards
<http://www.wrinkl.de/Javacard/Javacard.html>
- [26] How to write a Java Card applet: A developer's guide.
<http://www.javaworld.com/javaworld/jw-07-1999/jw-07-javacard.html?page=5>
- [27] The ISO 7816 Smart Card Standard: Overview
http://www.cardwerk.com/smartcards/smartcard_standard_ISO7816.aspx
- [28] A Free Implementation of Machine Readable Travel Documents
<http://jmrtid.org/>
- [29] Developing a Java Card Applet
<http://developers.sun.com/mobility/javacard/articles/applet/>
- [30] Unified Modeling Language (UML)
http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/index.htm

- [31] Installing Java Card Applet into real SmartCard
<http://adywicaksono.wordpress.com/2008/01/05/installing-javacard-applet-into-real-smartcard/>
- [32] CCID free software driver
<http://pcsclite.alioth.debian.org/ccid.html>
- [33] Z. Chen. Java Card Technology for Smart Cards. *Architecture and Programmer's Guide*. 2000.
- [34] EasyEclipse
<http://www.easyeclipse.org/site/home/>
- [35] Security and Privacy Issues in E-Passports, Ari Juele, David Molnar, and David Wagner, RSA Labs, 2005.
- [36] Sun Microsystems, Java Card Development Kit
<http://java.sun.com/javacard/devkit/>
- [37] EclipseJCDE plugin (Java Card applets development).
<http://eclipse-jcde.sourceforge.net/>
- [38] EasyEclipse Desktop Java
<http://www.easyeclipse.org/site/distributions/desktop-java.html>