Secure Watemarking Method with Smart Card

Hafid Mammass*
MMS, Faculty of Sciences,IbnZohr University
Agadir, Morocco
Email: * hmammass {at} gmail.com

Fattehallah Ghadi& Mohamed Elhajji
MMS, Faculty of Sciences,IbnZohr University
Agadir, Morocco

Abstract—***we treat in this paper a secure authentication method using watermarking, smart card and cryptography. It can serve to control logical and physical access and the network access and to prove the card owner identity.This method is very secure because the high level of security of smart card, the method of watermarking and detection and the complexity of the algorithm RSA and DES aim to remove the most known attacks.***

Keywords-component; ***Smart card, cryptography, RSA, symmetric and asymmetric algorithms, watermarking, detection*****, DCT, DCT inverse**

I. INTRODUCTION

The smartcard are world widely spread in the banking field, the health field, the ticketing field, and several another domains because they can store confidential data and handling a dynamic or static authentication scheme and compute the last ones symmetric and asymmetric algorithms.

The watermarking consists of adding a message to video, image, audio signal or digital document to prove who the owner is and then protect the copyright.

In this paper, we store into the card, the keys whose served to embed the ciphered message into the image (of the card owner) and in the step of detection, the tool asks the card to send it the secret keys stored during the watermarking step and use them to extract the ciphered message from the image and asks the card to decrypt the message and then compare the extracted message and the original message.

We can install an applet Java Card into the card in the EEPROM after the step of manufacturing, with this functionality we can install as we want and modify the applet in Java Card language which is a subset of java language but with a minimum of features because of the restricted resources on the card.

With the technology Java Card introduced by Sun Microsystem, the last ones smartcard are multi-application (see Fig. 1) and we can install several applets into the same card, hence, we can have an electronic purse applet, a ticketing applet and a payment applet. The Java Card technology assumes that an applet cannot access to data of another applet because a firewall protects every applet and the JVCM (Java Card Virtual Machine) offers a context to each applet. But, it is possible to access to another context for the applets belonging to the same package.

II. SMART CARD ARCHITECTURE

The central processing unit in the most smart card is a micro-controller 8-bit and we use a set of instructions Motorola 6805 or Intel 8051 with a 5 MHZ clock and the most recent cards have microcontroller 16-bit or 32 bits and processors with RISC architecture (Reduced Instruction Set Computer) are now available.

The ROM (Read Only Memory) is 16Ko to 64Ko, the EEPROM (Electrically Erasable Programmable Read Only Memory) is 64 Ko and RAM is 256 bytes to 1 Ko and the CPU has the capacity to execute the instructions from the ROM or the EEPROM and it is capable to realize all that microcomputer is capable of doing.

The Smart Card have 8 contacts whose are used to achieve to communication between the card and external world and the most important is the I/O contact which is used to transfer the commands data between the Smart Card and the external world in the half-duplex mode.

The operating system is burned in the ROM and persists in this memory area until the end of life of the smartcard.

The RAM contains the transient objects (temporary) and the intermediates calculations realized by the card.

The EEPROM contains the applets and their attributes, hence, the applets are loadable in this zone during all the life cycle of the card.

The smart cards used in safety or security applications often have a coprocessor, a cryptographic coprocessor is a specific integrated circuit which realizes calculations, particularly the modular arithmetic and a big number calculations , these calculations are necessary to cryptographic operations as the RSA Algorithm and the inclusion of a cryptographic coprocessor influences the cost of a smart card manufacturing.

The ROM is used to store the operating system of the card and no power is required to maintain data in this area, however, she cannot be modified after the manufacturing of

the card, some ROM contain the operating system and permanent data of supported applications.

The EEPROM as the ROM can maintain data in absence of energy, the difference is that the contents can be modified during the use of the card, so, it is used to store the data and it is for the card as the equivalent of the hard disk for the microcomputer.

The users applications can be loaded in this area after manufacturing of the card and the main parameters of the EEPROM are the number of write cycles during the life cycle, the period of retention and the access time.

In most of the cards, the EEPROM can accept in most 100.000 write cycles and hold data during 10 years.

The reading of an EEPROM is as well fast as the reading of the RAM but the writing of an EEPROM is 1000 times as slow as a RAM.

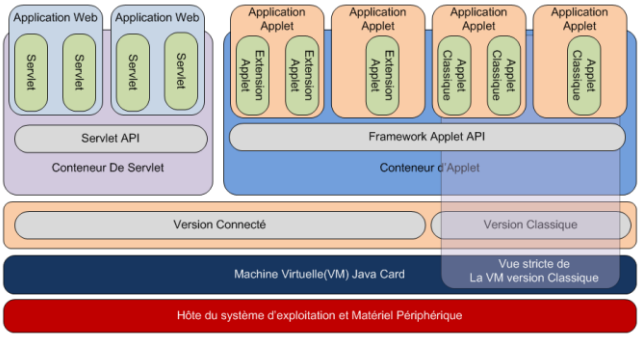

Fig1. Smartcard Architecture

III. APDU PROTOCOL (ISO7816)

The Application Data Unit Protocol is the standard ISO7816 which defines the structure of the commands exchanged between the card and the external world as well as the predefined answers sent by the card in case of error what allows a more appropriate management of the exceptions (see TABLE I).

TABLE I. APDU SELECTION

class	instruction	Param1	Param2	Data Length	Data	Length expected
CLA	INS	P1	P2	LC	DATA	LE

CLA : class of instruction 0x90

INS : code of instruction as defined in the applet

P1: first parameter

P2 : second parameter

LC : length of data in bytes

DATA: array of bytes

LE: length expected of the card answer

Example:SELECT command(see TABLE II)

This command allows to select an application installed on the card via its AID (Application Identifier) so after this selection, the application can be requested by the terminal to execute the services proposed by the applet and by the application.

TABLE II. SELECT COMMAND

CLA	INS	P1	P2	LC	DATA	LE
0x90	A04	4		Len.AID	AID	0x00

The answer is defined as followed (see TABLE III)

TABLE III. CARD RESPONSE

DATA	SW1	SW2

SW1 SW2 = 0x 9000 if the command is successful

IV. WATERMARKING METHOD

We use a DCT transform and IDCT transform for embedding the message in the dominant blocks as [3] proposed with Waveletand we use three parameters seed1 and seed2 for randomization of the image data and opacity to adjust the brightness of image, they are the secret keys for this step because during the step of detection (see Fig.2) we can read the three parameters from the card and use them to randomize and adjust the image result.

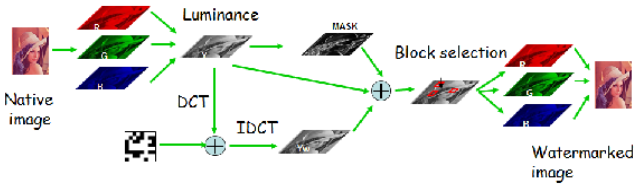

Fig2. Watermarking Method

The formula of DCT-II

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \quad (1)$$

The DCT-III is the inverse of DCT-II

$$X_k = \frac{1}{2}x_0 + \sum_{n=1}^{N-1} x_n \cos\left[\frac{\pi}{N}n\left(k+\frac{1}{2}\right)\right] \quad (2)$$

V. JAVACARD TECHNOLOGY

A. JavaCard Language

Because of its limited resources in memory, the Java Card platform supports only care-fully chosen, customized subset of the features of Java language but the Java Card preserves the object-oriented capabilities of the Java programming language, hence, the Java Card language supports small primitive data types: boolean, byte, short, one-dimensional arrays, Java packages, classes, interface and exceptions, Java object-oriented features: inheritance, virtual methods, overloading and dynamic object creation, access scope, and binding rules and the type's int and integer are optional.

The Java Card language don't supports: large primitive data types: long, double, float, characters and strings, multidimensional arrays, dynamic class loading, security manager, garbage collection and finalization, threads, Object serialization, Object cloning [1],[2].

B. JavaCard Virtual Machine

The Java Card is compiled and interpreted language because the first step generates a byte code which is compatible with all platforms and this step is realized off-card by the converter which generates a CAP file and an export file and the second step is executed by the interpreter which load and execute the CAP file on the card (see Fig. 3).

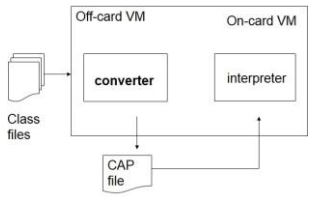

Fig 3.JavaCard Virtual Machine

C. The Converter

By opposition, the Java Virtual Machine, which processes one class at time, the conversion unit of the converter is a package, class files are produced by a Java Compiler from the source code, then the converter treats all the class files of the package and converts them to CAP file.

The converter (see Fig. 4) verifies that the load images of the Java classes are well formed; hence, it checks the Java Card language subset violations, it performs static variables initialization, it resolves symbolic references to classes, methods and fields, it optimizes bytecode and it allocates storage and creates virtual machine data structures

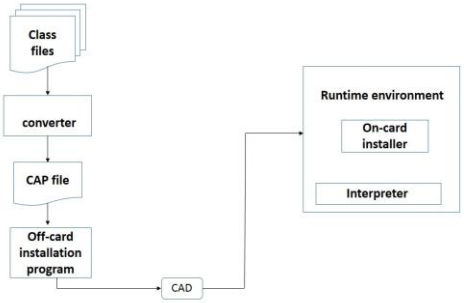

Fig 4. The Converter

D. Applet Implemenation

The methods install, select, deselect, and process are applet entry point methods. They are invoked by the JCRE at the appropriate state of applet creation and execution. The base Applet class gives only the default behavior for these methods. An applet must have to override some of all of these methods to implement its functions.

All the applet are derived from the class javacard.framework.Applet and to load an applet, the off-card installer takes the CAP file and transforms it into a sequence of APDU commands, which carry the CAP file content, by exchanging the APDU commands with the off-card installation program, the on-card installer writes the CAP file into the card's persistent memory and links the classes in the CAP file with other classes that reside on card.

The last step during an applet installation, the installer creates an applet instance and registers the instance with the JCRE (Java Card Runtime Environment) by invoking the install methodpublic static void install(byte [] bArray, short offset, byte length), then, Every applet must implement the method install and in this method, the applet must call the applet's constructor to create an applet instance.

After the applet is initialized and registered with the JCRE, it can be selected and run.

The applet can register itself with the JCRE by using the default AID found in the CAP file.

E. Applet Registration

We use two methods to register an applet with the JCRE

protected final void register();

protected final void register(byte [] bArray, short bOffset, byte bLength)

The first register method without arguments registers the applet with the JCRE using the default AID from the CAP file. The second register method with arguments registers the applet instance with the JCRE using the AID specified in the argument bArray

F. Applet Selection

An applet remains in a suspended state until it is explicitly selected, an applet selection occurs when the JCRE receives the SELECT APDU with the application AID and then the JCRE informs the applet of its selection by invoking its select method, then, the applet returns true if it is ready to receive incoming APDU via its process method.

G. Applet Deselection

Before a new applet is selected, the JCRE deactivates the current applet by invoking its deselect method and during the processing of deselect method, the applet prepares itself to go "of-stage" and to allow another applet to execute

H. Applet Behaviour

At the reception of an APDU, the JCRE calls the current applet's process method, the behavior of this method is related to the function requested in the APDU.

On receiving an APDU, the process method analyze the APDU header and determines which function is requested and to will be executed.

To retrieve an APDU Buffer, the method process must call the method apdu.getBuffer

To read data into the APDU buffer, the applet invokes the setIncommingAndReceive.

public short setIncomingAndRReceive() throws APDUException

This method sets the JCRE in the data-receiving mode and requests the JCRE to receive the incoming command data bytes starting at the offset ISO7816.OFFSET_DATA in the APDU and returns the number of bytes read.

To receive long command data, the method SetIncomingAndReceive is not sufficient then we use the receiveBytes APDU method

public short receiveBytes (short bOff) throws APDUException

I. Applet Response

After the execution of the function requested in the incoming APDU, the applet can send the answer to the host, because of the half-duplex mode, the applet must call the setOutGoing method to indicate that it wants to send out response data and this method don't send any data, it sets only the data transfer mode and it returns the number of response data bytes (LE) expected by the host.

public void SetOutGoing() throws APDUException

After the call of SetOutGoing method, the applet calls the method SetOutGoingLength to indicate to the host the number of bytes the applet will send to it.

public void setOutGoingLength(short length) throws APDU Exception

At the last step, the applet calls the sendBytes method to send out the response data

psublic void sendBytes (short bOff, short len) throws APDUException

VI. Solution Overview

A. Card Authentication

The tattooer asks the card to send it, the RSA public key of the card, then it sends to the card a challenge generated randomly and the card generates a signature RSA with its private key and send the signature to the tattooer which verifies it with the public key already retrieved from the card [4],[5],[6].

Before this card authentication, the tattooer is not able to insert a watermark into the image and to store into the card, the keys which are used in this process.

The command for retrieving the RSA public key (see TABLE IV)

TABLE IV . APDU GET PUBLIC RSA KEY

CLA	Instruction	P1	P2	LC	Data	LE
0x90	INS_GET_PUBLIC_ KEY	0x00	0x00	LC	challenge	0x00

And the algorithm RSA is described below:

n = p*q , p et q are big prime integer with similar lengths

$\phi = (p-1)*(q-1) \bmod N$

$ed = 1 \pmod{\phi}$

$C = M^e \pmod N$

$M = C^d \pmod N$

For each card, we generate a couple of RSA public key and RSA private key and we use them in the applet to sign message and to encrypt and decrypt data and we use a key length of 1024 bits.

To sign signature we use the object signature as followed

Signature signature;

Signature = Signature.getInstance(Signature.ALG_RSA_SHA1_PKCS1, false);

And we use the method init :

public void init (Key theKey, byte theMode);

MODE_SIGN or MODE_VERIFY

Signature.init(privateKey,Signature.MODE_SIGN)

signature.update(challenge, (short)0, (short) (challenge.length));

signature.sign(challenge,(short)0,(short)challenge,sig_buffer, (short)0)

To verify the signature, we use the verify method

signature.init(publicKey,Signature.MODE_VERIFY)

signature.update(s1,(short)s1.length);

signature.verify(s2,(short)0,(short)(s2.length), sig_bufffer, sig_offset,sig_length);

To encrypt data, we use the init, update, and doFinal methods

Cipher cipher;

cipher = Cipher.getInstance (Cipher.ALG_DES_CBC_NO_PAD, false);

cipher.init(desKey, Cipher.MODE_ENCRYPT);

And we use cipher.update and cipher.doFinal.

To decrypt data, we use the init, update, and doFinal methods

Cipher cipher;

Cipher = Cipher.getInstance (Cipher.ALG_DES_CBC_NO_PAD, false);

Cipher.init(desKey, Cipher.MODE_DECRYPT);

And we use cipher.update and cipher.doFinal

B. Storage of the keys

We send to the card the commands INS_PUT_SEED1 (see TABLE V), INS_PUT_SEED2 (see TABLE VI), INS_PUT_OPACITY (see TABLE VII)and INS_PUT_MESSAGE (see TABLE VIII) to store the seed1, seed2, opacity and the message on the CARD.

TABLE V. APDU PUT SEED1

CLA	instruction	P1	P2	LC	Data
0x90	INS_PUT_SEED1	0x00	0x00	LC	SEED1

TABLE VI. APDU PUT SEED2

CLA	instruction	P1	P2	LC	Data
0x90	INS_PUT_SEED2	0x00	0x00	LC	SEED2

TABLE VII. APDU PUT OPACITY

CLA	instruction	P1	P2	LC	Data
0x90	INS_PUT_OPACITY	0x00	0x00	LC	OPACITY

TABLE VIII. APDU PUT CIPHERED MESSAGE

CLA	instruction	P1	P2	LC	Data
0x90	INS_PUT_MESSAGE	0x00	0x00	LC	MESSAGE

Before embedding the message into the image, the tool asks the card to cipher the message with DES algorithm and secret key stored into the card (see TABLE IX).

TABLE IX. APDUENCRYPT MESSAGE

CLA	instruction	P1	P2	LC	Data
0x90	INS_DES_ENCRYPT	0x00	0x00	LC	Message

VII. DETECTION METHOD

We send to the card the commands INS_GET_SEED1 (see TABLE IX), INS_GET_SEED2 (see TABLE X), INS_GET_OPACITY (see TABLE XI) and INS_GET_MESSAGE (see TABLE VIII) to read the seed1, seed2, opacity and the message from the CARD.

The tool uses the parameters seed1, seed2 and opacity which have just been readfrom the card and having served in the process of watermarking to extract the message by using the discrete transform in cosine (DCT) and its inverse, decrypts the message (see TABLE XIV) and compares the extracted message, and the original to authenticate the cardholder.

TABLE X. APDU GET SEED1

CLA	Instruction	P1	P2	LE
0x90	INS_GET_SEED1	0x00	0x00	LE

TABLE XI. APDU GET SEED2

CLA	Instruction	P1		P2	LE
0x90	INS_GET_SEED2	0x00		0x00	LE

TABLE XII. APDU GET OPACITY

CLA	Instruction	P1		P2	LE
0x90	INS_GET_OPACITY	0x00		0x00	LE

TABLE XIII. APDU GET MESSAGE

CLA	Instruction	P1	P2	LE
0x90	INS_GET_MESSAGE	0x00	0x00	LE

TABLE XIV.APDU DECRYPT MESSAGE

CLA	Instruction	P1	P2	LC	Data	LE
0x90	INS_DES_DEC RYPT	0x00	0x00	LC	Message	0x00

The schema of the detection is described below (see Fig. 5)

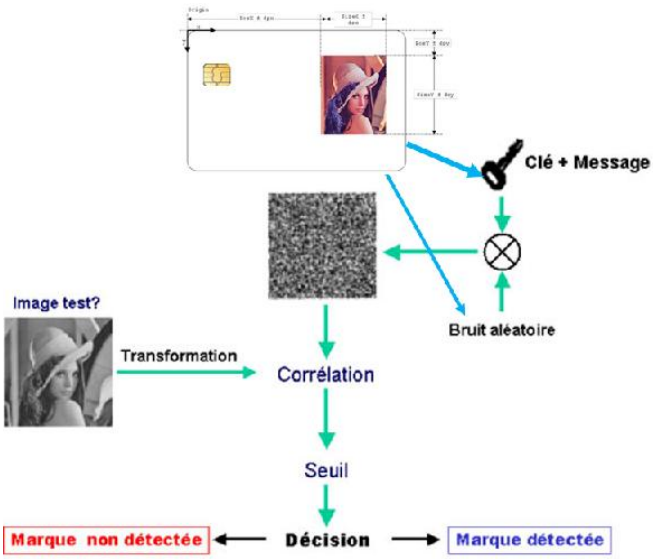

Fig5. Schema of Detection

VIII. CONCLUSION

The use of watermarking and the cryptography with Smart Card can provides a multiple methods to secure the process of watermarking and detection and handling a strong card authentication and remove the most known attacks because of the high level of security supplied by Smart Card and the last one of symmetric and asymmetric algorithms like DES and RSA.

This method can be used to control the physical, logical and network access and to prove the identity of the Smart Card owner and to prevents the cases of card theft and card loss.

We can add to watermarking and Smart Card the use of biometry to insert instead a simple message a characteristic biometric vector calculated in the step of enrolment and by defining a distance and by calculating in the step of detection the distance between the original image and the watermarked image and decide that the two images are similar according the distance and the parameterized thresholds.

REFERENCES

[1] Z. Chen, "Java Card™Technology for Smart CardsArchitecture and Programmer's Guide",Pearson Educationpp. 1-359 (2000)

[2] V.Hassler, "Java Card for E-Payment Applications"Artech House Publisher , pp. 1-361, November 2001

[3] *M. El Hajji, H. Douzi, D. Mammass, R. Harba and F. Ros,"*New Image Watermarking Algorithm Based on Mixed Scales Wavelet", Journal of Electronic Imaging , Vol. 21, 013003 , 2012.

[4] D. Pointcheval and P. Nguyen, 1st Edition.(2010), XIII, Softcover, ISBN: 978-3-642-13012-0, "13th International Conference on Practice and theory in Public Key Cryptography", PKC 201d0,pp. 519, May 26-28, Paris.

[5] Jean-Sebastien.Coron, "Cryptanalyses and Security Proofs of Public-Key Schemes", PhD Thesis 2001 at EcoleNormaleSupérieure , Paris, (2001).

[6] A. Menezes, P. van Oorschot and S. Vanstone, "Handbook of Applied Cryptography", CRC Press**, (**1996**).**

[7] H. Mammass, "Implementation of Smart Card Personalization Software",ICMCS'11 - IEEE cosponsored Conference, 2nd International Conference on Multimedia Computing and Systems, Ouarzazate, Morocco.

[8] C. Delannoy (EYROLLES), "Programmer en Java 5ème édition Java 5 et 6" pp. 787.