Secure Electronic Cards in Public Services

Cristian TOMA, Marius POPA, Cătălin BOJA
Department of Economic Informatics, Academy of Economic Studies, Bucharest
Miruna VASILACHE Department of IT R&D Oxygen Vision, Bucharest cristian.toma@ie.ase.ro, marius.popa@ase.ro, catalin.boja@ie.ase.ro, mirunavasilache@gmail.com

The paper presents the electronic wallet solution implemented within a GSM SIM technology for accessing public services. The solution is implemented in the medical field to provide information on the patient's medical history and payment for private medical services. The security issue is a very important one as the patient's history is confidential and the payment has to be safe.
Keywords*: public services, mobile, security, m-application, smart card.*

1 Introduction

The chapter presents concepts that used in the designed solution for the electronic card (wallet) used in medical services. The main issue is to design and implement a solution which helps the hospitals to manage the patient's history information and can help insurance companies to do the billing with hospitals' systems. The paper presents a different solution as the ones existing in the market (in Germany the most known is KVK – Deutsch Krankenversichertenkarte). The medical electronic wallet is a Java card application (called also cardlet) which is running within the mobile SIM. The cardlet communicates with the "external world" via a Java midlet. The GSM – Global System for Mobile Communication includes many technologies for voice and data transmission. For GSM there are few distinctive types of applications: Pull typical applications (Web); Push typical applications (SMS/MMS); SIM Toolkit – Subscriber Identity Module Toolkit applications that are running in the SIM Smart Card using native code or Java Card technology [3], [4], [5], [6]; Native applications (which are running on the top of operating system of the mobile device); Applications written in Java Micro-Edition or in C# for .NET Compact Framework (that are running in proper virtual machines on the mobile device); and Hybrid Applications. Hybrid Applications provide complex services such as SIM Sentry for Multimedia Mobile Content Digital Rights Management, Midlets with Java Smart Card technology solution for mobile banking or electronic purses, Web WAP applications for mobile streaming over RTP and RTSP in GSM networks [3], [4], [5], [6], [7], [8].

In this paper we focus on hybrid applications because the electronic wallet for the medical services is a cardlet that is running in mobile SIM and it communicates via a midlet application with a desktop application using Bluetooth or WiFi technology. The desktop application accesses information from distributed databases using web services over secure communication protocols such as SSL.

The main concept used in GSM for end-user device, the mobile is a two in one computer. The first computer is represented by the SIM – Subscriber Identity Module. Actually, the SIM is a smart card with a microcontroller, three types of memory area (ROM, EEPROM and RAM) and I/O ports for outside communication (usually in half duplex mode). The mobile device itself is the second computer. It also includes a microprocessor, different types of memory areas and an operating system.

The electronic wallet that is running in GSM mobile SIM is a Java card application.

2. Java Card Technology used for medical electronic wallet

A Java card application is an applet which is running in smart card. But often the applet needs to interact with different systems and applications. That's why in specialty litera-

ture a complete application for Java smart card is composed from the java applet which is running on smart card, a host application and back-end application systems which provide to the end-user a service.

With SATSA, a Java midlet can access a Java Card application using either Message-Passing Model or JCRMI model. Our Java application, which is run by the mobile Java Micro-edition virtual machine (which is managed by the mobile device operating system), communicates with our electronic wallet, which is run by the SIM Java Virtual Machine (which is managed by the SIM-smart card operating system), using JSR 177 over Message-Passing model. The communication between host application (mobile Java midlet) and the applets (Java smart card application) from smart card suppose to transmit some APDU – Application Protocol Data Unit from host to CAD – Card Acceptance Device, and then the same bytes strings are sent from the CAD to the card applet. The card applet (has nothing to do with Java applets which are run by Internet browsers) receives those bytes strings, it is parsing the bytes and then will send back following the reverse path: Applet-CAD-Host (see Figure 2). An APDU is composed from standard bytes blocks conform ISO/IEC 7816-3 and 7816-4. Respecting the standards the applet receives directly from CAD, *APDU Commands* and sends back to CAD, *APDU Responses*. The communication between the card reader and the card is physically realized through data link protocol. This protocol is likely data link level protocol from protocol stack ISO/OSI. The link protocol, defined in ISO/IEC7816-4, has four alternatives: T=0 – byte oriented, T=1 – bytes arrays oriented, T=USB – oriented Universal Serial Bus or T=RF – radio wave oriented, Radio Frequencies. The classes from Java Card API and JCRE specifications embed the physical details for APDU communication protocol.

The general template for an ADPU Command is depicted in figure 2.

There are other four specific structures for an APDU Command, but these structures are used only in data link protocol T=0.

APDU Command						
CLA	INS	P1	P2	Lc	Data Field	Le
Header-mandatory				Body-optional		

Fig.2. General structure for an APDU Command.

The explication for the fields from the APDU command is the following:

- CLA – is one byte – 2 hexadecimal digits, and has different predefined values conform standard ISO7816. For instance, between the value 0x00 and 0x19 are values for accessing file system and security operations, from 0x20 to 0x7F are reserved for future using, and from 0x80 to 0x99 can be used for applets' specific instructions implemented by developers but between 0xB0 and 0xCF are specific instructions for all applets and not for a particular one. As matter of fact the most used value for this field is 0x80. We use 0x88.
- INS – is one byte, and the standard defines a specific instruction in the field CLA. For instance, when CLA has the value between 0x00 and 0x09, but INS has the value 0xDC – means card's records update. In personal applications which are installed on the card, the field INS could have predefined values established by developers but according with the standard. For example, the developer chooses for this field the value 0x20 for checking sold amount from card if and only if the CLA field is 0x80;
- P1 – this represents the first parameter for an instruction and it has one byte. This field is used when the developers want to send some parameters to the applet or want to qualify the INS field;
- P2 – this is the second parameter for an instruction and it has one byte. It is used for the same goal like P1;
- Lc – has one byte, it is optional and represents the bytes length for the field Data Field;
- Data Field – is not fixed and has a bytes' length equal with the value from the field's value Lc. In this field data and parameters are stored and they are sent from host application to applet;
- Le – stores the maxim number of bytes that should have Data Field from APDU Re-

sponse (the number of bytes from response could be any value from the range 0 and the value from this field).
Practically a host application sends to the CAD but the CAD sends to the applet the same APDU commands with structures and values which respect the standards.
The structure for an APDU Response is simple and is depicted in the figure 3:

APDU Response		
Data Field	SW1	SW2
Body-optional	Trailer-mandatory	

Fig.3. Structure for an APDU Response.

The fields' explication for APDU Response is the following:
- Data Field – has variable length which is determined by the value of the byte field Le from the APDU Command;
- SW1 – has one byte and represent the status word 1;
- SW2 – has one byte and represent the status word 2.

The fields SW1 and SW2 are parsed and interpreted together, but a communication process is called *complete* if there were no problems (SW1=0x61 and SW2=0x90 or any-0xnn) or if there were only warnings (SW1=0x62 or SW1=0x63 and SW2 contain the warning code). A communication process is called *failed* if there were execution errors (SW1=0x64 or SW1=0x65 and SW2 have the error code for execution) or checking errors (SW1=from 0x67 to 0x6F and SW2 have the code for checking error).
A typical code of a Java midlet that communicates with the cardlet is in table 1:

Table 1. A Java midlet JSR177 code

```
...
try {
// Create an APDUConnection
String url =
"apdu:0;AID=A1.0.0.67.4.7.2F.3.2C.3";
byte[] commandAPDU = new
byte[]{(byte)0x88, (byte)0x20, …};
//debit command

APDUConnection ac = (APDUConnec-
tion)Connector.open(url);

//Send a APDU command and receive a
APDU
//response
byte[] responseAPDU =
ac.exchangeAPDU(commandAPDU);
...
// Close connection.
ac.close();
} catch(IOException e){
    ...
}
...
```

The midlet creates a JSR177 connection with the cardlet. The connection properties are encapsulated by the ***ac*** object. The APDU command is sent from the midlet to the cardlet using the method ***exchangeAPDU()*** of object ***ac***. ***The life cycle of JCVM and of a Java Card applet*** should be understood by anyone involved in Java smart card field. The life duration of JCVM is the same with that of the card. If the power supply of the card stops, the entire content of JCVM is saved in the persistent memory, non-volatile. Everything in the internal memory-RAM-volatile of the card at the moment of the interruption of power is lost. Moreover, the objects created in the Java Card platform are non-volatile, and if it is sometime intended that an octet string should be in the volatile memory because, for instance, it holds only temporary data, we should use the method makeTransientByteArray() of the class *javacard.framework.JCSystem*.
The methods by which the applet life cycle is realized and that the applet should implement are presented in figure 4:

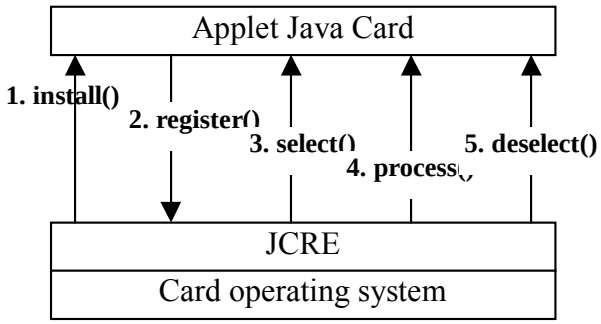

Fig.4. Methods to be implemented by an applet to execute the complete life cycle

Each applet is uniquely identified by an octet string – between 5 and 16, as defined in ISO7816-5. The octet string is named AID –

Application ID in the standard. As well, each applet should extend the abstract Applet class and implement the methods – install(), register(), select(), process(), deselect(), that represent the applet life cycle. The applet life cycle starts immediately after it is downloaded into the card and JCRE forces the execution of the static method Applet.install() and at its turn the applet is registered in JCRE by calling the static method Applet.register(). After the applet is installed and registered, it exists on the card as „not selected", the equivalent denomination being inactive applet. An applet is activated to process APDU Commands only after the host type application sends to JCRE by CAD an APDU command of type SELECT or MANAGE CHANNEL. JCRE complies and notifies the concerned applet by forcing the execution of select() method which the applet implements. After the selection is done, all the APDU commands received from the host type application by CAD of JCRE are sent to the applet by forcing calling the method process() implemented by the applet. The life cycle of the applet ends when the host application intends to select another applet for processing the APDU commands, moment when JCRE notifies the applet by forcing the execution of deselect() method of the applet. A typical code of a Java cardlet that communicates with the Java midlet is in table 2:

Table 2. A Java Card code

```
package
com.sun.javacard.samples.wallet;
import javacard.framework.*;
public class MedicalWallet extends
Applet {

private static byte[] name = new
byte[] {(byte)0x85, (byte)0x03,
(byte)0x49, (byte)0x6F, (byte)0x6E};
private static byte[] surname = new
byte[] {(byte)0x86, (byte)0x07,…}

...
//constructor
private MedicalWallet (byte[] bAr-
ray,short bOffset,byte bLength)
{...}
//Life-cycle methods
   public static void install(...)
{...}
   public void select() {...}
   public void deselect() {...}
   public void process(APDU apdu)
{...}
//private methods
   private void payConsult(APDU ap-
du) {...}
   private void sendMedRec(APDU ap-
du) {...}
   private void verify(APDU apdu)
{...}
   private void getInfo(APDU apdu)
{...}
   private void setInfo(APDU apdu)
{...}
...
}
```

Another important issue in the listing from table 2 is that the personal medical information is stored in ASN.1 – Abstract Syntax Notation (TLV – Tag Length Value). For instance, the variable ***name*** from table 2 is identified by tag 0x85 with 0x03 length chars. The chars represent the string "Ion" in ISO-8859-1 which is the name of the medical patient.

3. The Secure Electronic Card Solution for Medical Services

Once we established the technical details, we can discuss our ***SECMS – Secure Electronic Card for Medical Services*** solution.

The designed ***SECMS – Secure Electronic Card for Medical Services*** solution for an electronic wallet used in medical services will exploit the presented concepts in the previous chapters. The architecture of ***SECMS*** solution is presented in figure 5. In figure 5 the business flow is clear. The patient goes to the doctor for a medical consultation. In order to have a complete view before diagnosis, the doctor may request to see and to update the medical records of the patient. On the patient's mobile device are running two applications (both digitally signed by Health Insurance Company): the Java midlet on the phone and the smart card application 'MedicalWallet' on the SIM. The communication between the 'MedicalWallet' and the Java midlet is JSR177. The medical record information is sent to the doctor application only if the patient inserts the applica-

tion PIN. The medical record information is very sensitive and therefore the communication between the Java midlet and doctor application uses SSL – TCP/IP over WiFi. After the medical consultation the doctor application of our solution called over HTTPS a web service (first implementation was XML-RPC).
The doctor application sends encrypted and authenticated SOAP – Simple Object Access Protocol messages (briefly described in [10]) which will contain:
- the patient SIM IMSI;
- the patient SSN – Social Security Number;
- the patient digitally signed hash with the SIM RSA secret key over first two information plus a nonce generated by the SIM;
- the doctor IBAN and ID card number;
- the money amount to debit.

The web service called EJB from the billing system and send payment order via Web services over HTTPS to the Health Insurance Company of the patient. The Health insurance server application receives over HTTPS the electronic payment order from various billing systems. The Health Insurance Company has two powerful interconnected systems: PKI – Public Key Infrastructure system and the processor application of payment orders called also Payment Gateway. The transfer of money between the banks is not included in SECMS goals. Because the description of each segment of communication in detail needs more pages, we will present some details of communication between patient application and doctor application.
The communication between the patient midlet and doctor application is inspired in terms of security concept from SET – Secure Electronic Transaction [2], [4]. The communication uses dual signature concept. **Dual signature** is an innovative method for resolving the following aspect:
- The patient needs to send the SIM IMSI and SSN included in "owner information" (OI) to the merchant and the "payment information" (PI) to the billing system;
- Ideally, the patient does not want the bank to know the OI and the doctor to know PI;
- However, PI and OI must be linked to resolve disputes if necessary (e.g., the patient can prove that the medical consultation has been paid);

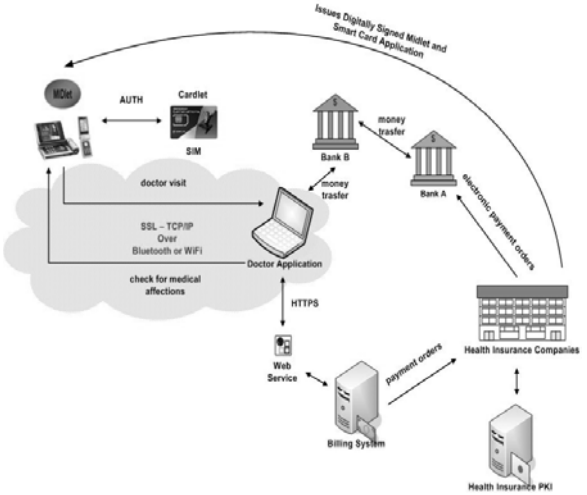

Fig.5. SECMS – Secure Electronic Card for Medical Services Architecture

The steps for **dual signatures** are:
- The message digests – using one of MD5, SHA-1 message digest function – of PI and OI are found : PIMD=MD(PI), OIMD=MD(OI);
- The message digests are combined and the resultant message digest is found: POMD=MD(PIMD +OIMD);
- POMD is encrypted by using the customer's private key to produce the dual signature: DS=E(POMD, Priv_key).

The predicted scenario is based on:
- The doctor application is provided with OI, PIMD and DS. The signature can be verified by computing MD(MD(OI)+PIMD);
- The Health Insurance payment gateway is provided with PI, OIMD and DS. The signature can be verified by computing MD(MD(PI)+OIMD);
- By using dual signature, the patient can provide the linkage between OI and PI;

The reason for which the OI and PI are linked is the dual signature. The procedure for checking the bound established by the dual signature is:
- The doctor only knows OI but not PI;
- Payment gateway only knows PI but not OI;
- Nevertheless if either the OI or the PI is

changed, the dual signature will be changed;
- That means, OI and PI are linked together.

The figure 6 is comprehensive to depict how a *payment request by medical patient* and *verification by doctor application* are being created.

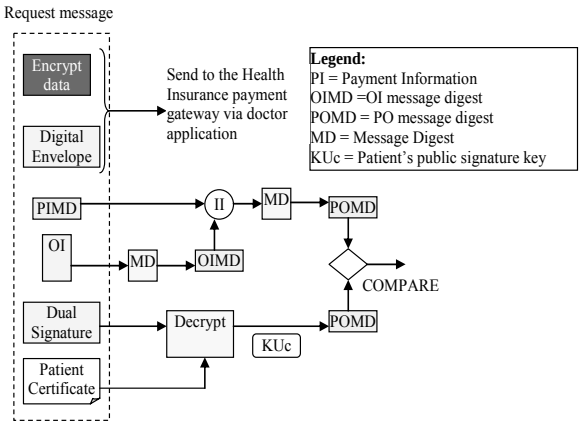

Fig.6. Verification process by doctor application

All the components are loosely coupled because each component of our SECMS can be implemented by different companies as long as they respect the designed specifications. The specifications are in draft form and we expect to be published in third quarter of 2008.

4. Conclusions

The suggested secure architecture is based on concepts like *electronic signature* and *encryption with symmetric keys* (described in detail by [1], [4], [5]), *SSL* – Secure Sockets Layer and *IP tracking* (described in [4]). Based on the analysis that were made in lab using a Nokia N95 device and a NetFront mobile browser it has been highlighted that the difference between processing time and access time is not significant. On average the process of digital signing takes at most three seconds. The project is still in development phase and the SIM application has been tested. This approach is developed to be accessed by a Java Micro-Edition Midlet, defined by JSR177. The solution presented here has many advantages for medical services which are extended on mobile devices because it allows developing secure environments with fewer resources because everything is taking place at software level and it doesn't interfere with the existing infrastructure.

Regarding the impact on mobile business solutions and on mobile services, the authors are continuing this research in two important research contracts that are financed by the Romanian Government through two complex development research projects, module MATNANTECH (Materials and Nanotechnologies) and module AMCSIT of Romanian Excellence Research Program.

References

[1] Rankl and Effing Wolfgang, *Smart Card Handbook, 3/E*, John Wiley & Sons, 2003.

[2] Bruce Schneier, Applied Cryptography 2nd Edition: protocols, algorithms, and source code in C, John Wiley & Sons, Inc. Publishing House, New York 1996.

[3] Douglas Stinson, *Cryptography – Theory and Practice*–2nd Edition, Chapman & Hall/Crc, NY 2002.

[4] William Stallings, *Cryptography and Network Security, 3/E*, Prentice Hall, 2003.

[5] Zhiqun Chen, *Java Card Technology for Smart Cards*, Addison-Wesley, 2004.

[6] Cristian TOMA, "Tutorial on Java Smart Card electronic Wallet Application", Informatics Security Handbook, AES Publishing House, Romania 2006.

[7] Cristian TOMA, Secure architecture used in systems of distributed applications, *The 7-th International Conference on Informatics in Economy*, May 2005, pp. 1132-1138.

[8] Ion IVAN, Cristian TOMA, Catalin BOJA, Marius POPA, "Secure Architecture for the Digital Rights Management of the M-Content", ISP'06 of the WSEAS Conference in Venice, Nov. 2006.

[9] Ion IVAN, Cristian TOMA, Catalin BOJA, Marius POPA, "Secure Platform for Digital Rights Management Distribution", WSEAS Transaction on Computers, 2007.

[10] SOAP Tutorial on www.w3schools.com