Deployment and Performance Analysis of JavaCards in a Heterogenous Environment

Computer Science Project

done by:
Carolin Latze
latze@iam.unibe.ch

head:
Prof. Dr. Torsten Braun
braun@iam.unibe.ch

assisted by:
Attila Weyland
weyland@iam.unibe.ch

Computer Networks and Distributed Systems
Institute of Computer Science and Applied Mathematics (IAM)
University of Bern
Switzerland

January 2005

SmartCards have become more and more important. Their field of application has increased over time and ranges from authentication infrastructures to electronic purses. Therefore we wanted to evaluate the current state of the art concerning the suitability for daily use, the performance and the functional range of actual SmartCards. To do so, we chose the Schlumberger Cyberflex Access e-gate32K SmartCard and performed our evaluations under Linux and Windows.

Contents

1 Theory

1.1 SmartCards

One of the main applications for SmartCards is dual factor authentication as described in [1]. This means, not only a single password, but a piece of hardware (the SmartCard) and a password identify a person uniquely. Such a system provides more security, since the password alone is not sufficient to gain access and also a stolen SmartCard does not mean access to the resources due to the password. Of course, a SmartCard can store additional keys for mail encryption and signing or to secure network communication. The alternative is to store such keys in a file on the hard-disk protected by a passphrase, which is more unsafe, since SmartCards provide access control and block the card after a certain number of wrong passwords (pins). Another advantage of SmartCards is their mobility, which allows to use the stored keys on any machine whereas a key-file would have to be copied first.

In contrast to standard chipcards, SmartCards have a microprocessor and programmable memory [2]. Due to the missing cpu, non-smart chipcards usually are monofunctional like telefoncards or, if they are intelligent, they offer only limited security by a pin and a counter which limites the trials for the pin.

SmartCards follow with their architecture ISO7816, which defines the card dimensions, the pin layout, allowed signals and voltage as well as the transmission protocol. SmartCards usually consist of a CPU, ROM, RAM, I/O unit and an EEPROM. However, the performance and the amount of memory is rather limited [3], which is why, for example, mainly symmetric cyphers are deployed, since these cyphers use less memory and are very fast. ISO7816 also specifies the communication with the card. Application Protocol Data Units (APDUs) are exchanged using Transmission Protocol Data Units (TPDUs). APDUs contain a method identifier to specify which method should be executed by the card and they could also contain needed parameters and an input string for this method. As written in [4] this protocol is organized after the master-slave model. The Computer always is the master and the SmartCard the slave.

1.1.1 Schlumberger Cryptoflex Cards

Cryptoflex Cards are SmartCards which have a minimal filesystem and support a standard set of commands. They provide cryptographic services to authenticate users and protect information. Cryptoflex Cards by Schlumberger [5] have a crypto-processor, which implements "'security industry functions based on public key cryptography directly on the card, eliminating the risk of sending secret unprotected data across a network"' [5]. Furthermore, these cards support secure key loading, on-card key generation and ciphering of imported/exported data. Schlumberger's Cryptoflex Cards consist of a 32 Kbytes EEPROM and integrate with PC/SC and PKSC#11.

1.1.2 Schlumberger Cyberflex Access Cards

Schlumberger Cyberflex Access Cards have a 32 or 64 Kbytes EEPROM and support cryptograhic services like DES, T-DES, RSA and SHA-1. The 32K cards have a ROM of 96KB and a RAM of 4KB, whereas the 64K cards consist of a 136KB ROM and also a 4KB RAM. Additional these cards consist of a 8 bits microcontroller and a co-processor for RSA and T-DES. They possess a JavaCard VM to execute applets, which can be programmed in JavaCard. A card can contain several applets, which are uniquely identified by their application identifier (AID) as defined in ISO7816. As the card has limited memory and computing power, JavaCard uses only a little subset of the Java programming language. It does not support dynamic class loading, security manager, threads and synchronization, object cloning, finalization, large primitive data types and, and thats the most disadvantageous, it does not support garbage collection.

1.2 Cryptographic Functions

1.2.1 RSA

As written in [2], RSA has been developed by Ronald Rivest, Adi Shamir and Leonard Adleman in 1978 and is based on the fact that there exists no fast algorithm to factorize big numbers into prime numbers. RSA is not an official standard although it is part of many official standards. Nevertheless it is widely-used and accepted and therefore constituted as a de-facto standard for asymmetric ciphers, which means that there are different keys for encryption and decryption whereas in symmetric ciphers, the same key is used for encryption and decryption.

The Algorithm

1. Choose two big prime numbers p and q and calculate the RSA-module $n = pq$.
2. Choose $d \in [0, n-1]$ where d is relative prim to $\phi(n) = (p-1) \cdot (q-1)$. Then choose a prime number d where $max(p, q) < d < \phi(n) - 1$.
3. Choose $e \in [0, n-1], e \cdot d = 1 mod \phi$
4. (e, n) is public key.
5. (d, n) is private key.
6. Encrypt clear text $M \in [0, n-1] : E(M) = M^e mod n$.
7. Decrypt crypto text $C \in [0, n-1] : D(C) = C^d mod n$.

Implementation

As written in the paragraph above, a clear text which should be encrypted using RSA has to be transformed in a numeric representation. Furthermore we need efficient algorithms to calculate big prime numbers, the number e and to exponentiate big numbers.

As RSA uses big numbers which represent the clear text and because of its operations, it is slower than symmetric ciphers. This is why RSA usually is not used to encrypt long clear texts, but to exchange session keys or to transmit signatures.

1.2.2 DES

The Data Encryption Standard (DES) is a symmetric cipher which encrypts or decrypts a 64bit long text using a 64bit long key[1].

Since 1998, DES has no longer a certification as it is unsecure. Its successor is the Advanced Encryption Standard (AES).

The Algorithm

The basic algorithm of DES is to divide an input into two blocks and to process them during several loops. Within one loop, the loop function is applied on only one block and the result is linked with the other block using XOR. Afterwards the two blocks are interchanged and the next loop begins. This procedure is shown in Figure 1.2.

As shown in Figure 1.1, the input is passed through an initial permutation IP, which results in a permuted input. This permuted input passes through 16 identical ciphering loops which use in every loop a new key K_i. These keys are created using a value which is the result of the permutation PC1 on the original key K. The result of the 16 loops passes through the inverted initial permutation.

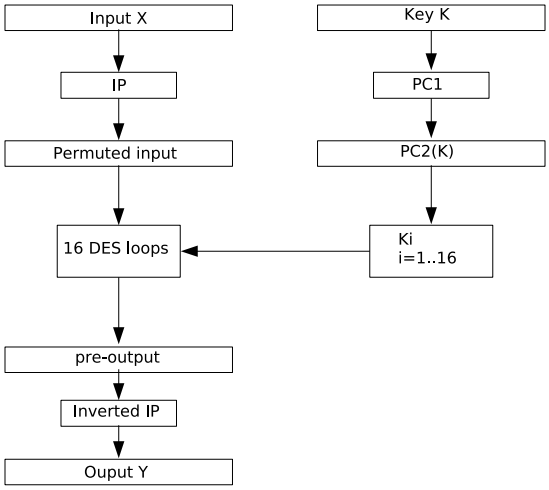


Figure 1.1: DES algorithm

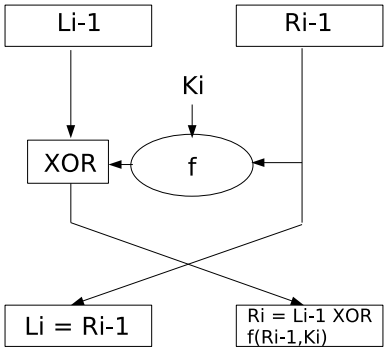


Figure 1.2: One DES Loop

In Figure 1.3 we see the loop function. The input block R_{i-1} is expanded duplicating several bits (expansion E). Afterwards the expanded block $E(R_{i-1})$ is linked with K_i using XOR which results in A. Then A is divided into 8 blocks A_j ($j = 1..8$) containing 6 bits and then transformed into 8 blocks S_k ($k = 1..8$) containing 4 bits using a nonlinear function. This

[1]The final key consists only of 56 bit. 8 bit are parity bits.

procedure results in 8 4 bits blocks B_k which will be merged into B. In the end, B will be permuted (permutation P), which results in $f(R_{i-1}, K_i)$.

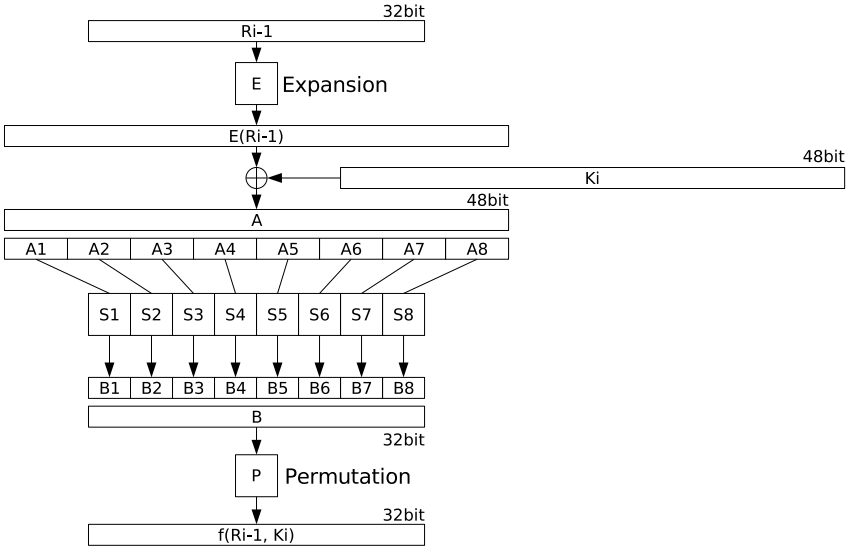


Figure 1.3: DES function

Implementation

DES is a cipher which can be easily implemented as hardware ciphering, because of the required operations and its low complexity.

1.2.3 Triple DES

Triple DES is a DES extension which is more secure. Triple DES uses two keys. As shown in Figure 1.4, the first key is used to encrypt a clear text, the second key to decrypt this encrypted text and finally the first key to encrypt this decrypted text.

1.2.4 SHA-1

The Secure Hash Algorithm SHA-1 is part of the Secure Hash Standard which has been published in 1993. SHA-1 creates 160 bit hash values and uses a block length of 512 bits.

The Algorithm

As shown in Figure 1.5 an input given to a SHA-1 hash function has to be splitted several times. In the first step, the input has to be padded to a multiple of 512 bits. Therefore a special padding function is used:

- the first bit after the original message is 1

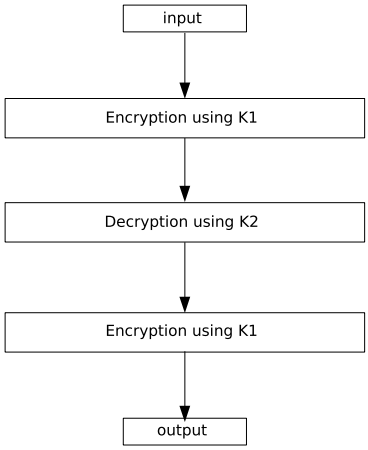


Figure 1.4: Triple DES algorithm

- the input is padded with 0 except for the last 64 bits
- the last 64 bits contain the length of the original message

Afterwards the padded input is divided into n blocks M_i of 512 bits, which are divided into 16 blocks W_j of 32 bits. As SHA-1 needs 80 blocks of 32 bits from every block of 512 bits, 64 blocks more have to be created as follows:
$W_t = W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}$
Thereafter these 80 blocks pass through 4 loops composed of 20 steps. The work-flow of one loop is shown in Figure 1.6, where S_x stands for "'shift left of x bits"' The blocks $A - E$ are initialized before the first loop starts. Afterwards three of these five blocks pass function f which is defined as:

$$f_k(x,y,z) = \begin{cases} 0x5a827999 + ((x \wedge y) \vee (\neg(x) \wedge z)) & 0 \leq i \leq 19 \\ 0x6ed9eba + (x \oplus y \oplus z) & 20 \leq i \leq 39 \\ 0x8f1bbcdc + ((x \wedge y) \vee (x \wedge z) \vee (y \wedge z)) & 40 \leq i \leq 59 \\ 0xca62c1d6 + (x \oplus y \oplus z) & 60 \leq i \leq 79 \end{cases}$$

Then, f_k, W_j and the other two variables, where one is shifted left of 5 bits, are summed up. All the blocks of 512 bits pass through this step. At the end, the variables $A - E$ contain the 160 bit hash value.

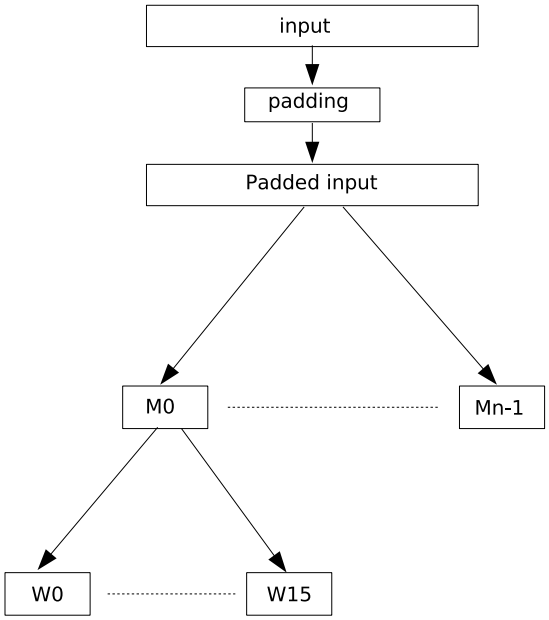


Figure 1.5: SHA-1 inputsplitting

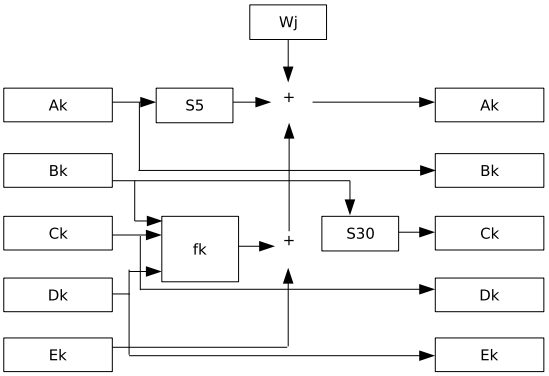


Figure 1.6: SHA-1

2 Deployment

2.1 Requirements

We wanted a flexible and current test environment to perform our analysis. Therefore we chose a programmable SmartCard, which has to sign and verify messages as well as store data securely. We tested the usability of the card in a Linux and in a Windows environment.

2.2 Installation

2.2.1 Linux Environment

Overview

The Smartcards communicate with the computer using the MuscleCard-Framework. However, to communicate with Cyberflex-cards they have to be initialized with a Java-Applet, which cannot be done using this framework as it is not provided. Therefore an additional AppletLoader is required, which is available as an open-source program.

MuscleCard PKCS#11 Framework

The MuscleCard-Framework provides two plugins, the MuscleCard-Plugin and the Cryptoflex-Plugin. The first one should be used to access Cyberflex Cards whereas the second one is responsible for Cryptoflex Cards. PCSC-Lite, which is expected by the framework, provides a daemon, which in turn is responsible for the interaction between the card and the computer.

The MuscleCard-Framework expects pcsclite-1.1.1 or higher. We used pcsc-lite-1.2.9-beta6-.orig downloaded from [6].

Installation steps:

1. `./configure`
2. `make`; `make install`; `make check` (to verify your installation)

If this was successful you can start pcscd with the -f (operate in foreground) and -d (debug) option. This should produce the output in Listing 2.1.

Listing 2.1: Output of pcscd -f -d stdout

```
sakigake:~# pcscd -f -d stdout
pcscdaemon.c:238:main pcscd set to foreground with debug send to stderr
pcscdaemon.c:257:main debug messages send to stdout
```

```
pcscdaemon.c:440:main pcsc-lite 1.2.9-beta6 daemon ready.
hotplug_linux.c:105:HPReadBundleValues Cannot open PC/SC drivers directory: /usr/
    local/pcsc/drivers
hotplug_linux.c:106:HPReadBundleValues Disabling USB support for pcscd.
```

Now MuscleCard-Framework can be installed which is provided by [7]. This version has to be patched with a patch provided by [7]. We only need the MCardPlugin and the CFlexPlugin from this framework. To install them perform the following steps:

1. Patch the framework: `cat muscleframework_1.1.5-4.diff.gz | patch -p0`
2. MCardPlugin:
   a) `./configure`
   b) `make`; `make install`; `make check`
   c) `./installBundle` (This will install the driver for the MuscleCard)
3. CFlexPlugin:
   a) `./configure`
   b) `make`; `make install`; `make check`
   c) `./installBundle` (This will install the driver for the CryptoflexCard)

USB-Driver

Then install the USB-Driver to access the card in the USB-reader. To do so, you have to install libusb-dev and libusb on your system. After doing that, download ifd-egate-0.05-patched.tar.gz from [8] and install it:

1. `make`; `make install`

Now, starting pcscd should produce the output in Listing 2.2.

Listing 2.2: Output of pcscd -f -d stdout

```
sakigake:~# pcscd -f -d stdout
pcscdaemon.c:238:main pcscd set to foreground with debug send to stderr
pcscdaemon.c:257:main debug messages send to stdout
pcscdaemon.c:440:main pcsc-lite 1.2.9-beta6 daemon ready.
readerfactory.c:1055: RFInitializeReader Attempting startup of E-Gate 00 00.
readerfactory.c:897:RFBindFunctions Loading IFD Handler 2.0
Card ATR: 3B 95 18 40 FF 62 01 02 01 04
```

To be sure, if everything works fine just run 'pcsc-lite-1.2.9-beta6.orig/src/testpcsc', which will write the output in Listing 2.3 (Note: You have to start pcscd first!).

Listing 2.3: Output of testpcsc

```
sakigake:~# /usr/local/pcsc-lite-1.2.9-beta6.orig/src/testpcsc
MUSCLE PC/SC Lite Test Program
Testing SCardEstablishContext    : Command successful.
Testing SCardGetStatusChange
Please insert a working reader   : Command successful.
Testing SCardListReaderGroups    : Command successful.
Group 01: SCard\$DefaultReaders
Testing SCardListReaders         : Command successful.
Reader 01: E-Gate 00 00
Waiting for card insertion       : Command successful.
Testing SCardConnect             : Command successful.
Testing SCardControl             : Transaction failed. (don't panic)
Testing SCardGetAttrib           : Transaction failed. (don't panic)
Testing SCardSetAttrib           : Transaction failed. (don't panic)
Testing SCardStatus              : Command successful.
Current Reader Name              : E-Gate 00 00
Current Reader State             : 0x0034
Current Reader Protocol          : T=0
Current Reader ATR Size          : 10 bytes
Current Reader ATR Value         : 3B 75 94 00 00 62 02 02 03 01
Testing SCardDisconnect          : Command successful.
Testing SCardReleaseContext      : Command successful.
PC/SC Test Completed Successfully !
sakigake:~#
```

MuscleTool

Muscletools is used to communicate with the card and to execute actions on it like format etc. It can be downloaded from [9] including additional patches. To install muscletools the following commands are required:

1. `zcat muscletools_0.9.2-3.diff.gz | patch -p0` (to patch it)
2. `make`; `make install`
3. edit /etc/ld.so.conf: insert a new line: /usr/local/lib (this should be the path, where libmusclecard.* and libpcsclite.* are located)
4. `ldconfig`

AppletLoader

If the MuscleCard-Framework is working, we need to install an AppletLoader. We used the IAC_Client provided by [10]. First install pkg-config and libssl-dev and extend the environment

variable PKG_CONFIG_PATH with the path where libpcsclite.pc is located. After doing that, install the AppletLoader:

1. `make`; `make install`

Change to the directory iacms/AppletLoaderClient/Software and start the AppletLoader using the following command: ./IAC_Client home.identityalliance.com 60000

Usage

You have to start the different tools in a special oder:

1. start pcscd
2. start bundletool and chose the bundle which belongs to your card
3. start muscletool and connect to your card

Problems

We had difficulties with Cyberflex cards under Linux. These cards need to be initialized with a JavaCard-applet, which should be done by the applet-loader. But the IAC_Client is unable to load an applet on the card. It can only list the card's content and delete applets on the card. We did not find a single applet-loader which can load applets on our cards. This is the reason why it was not possible for us to perform our tests with the Cyberflex-cards in an open-source Linux environment.

As written in several discussion forums, older versions of the Cyberflex Access Cards work with these tools.

2.2.2 Windows Environment

Overview

Schlumberger SDK, which must be purchased, provides building SmartCard-enabled PC applications on Windows. It contains the Program File Generator to convert class-files into ijc-files which can be installed on the Cyberflex Access egate32K card. It also allows to communicate with the card, e.g. to instantiate an applet and to communicate with this instance. Furthermore this SDK provides a middleware to communicate with the applet directly using your own program. As JavaCard is a subset of Java, a Java Runtime Environment (JRE) is needed. To develop the applets, it is recommended to use an Integrated Development Environment like Eclipse.

JRE

The Program File Generator which is integrated in the SDK only works properly if JRE 1.3 is used. Download it from [11] and install it running the installer. Ignore the warnings from Sun that this is not the latest version.

CLA	INS	P1	P2	Lc	Data	Le

Figure 2.1: Command APDU

Data	SW1	SW2

Figure 2.2: Response APDU

Eclipse

Because of the old JRE, an old version of Eclipse has to be used. Version 2.1 works fine. Download the zip-archive from [12] and unzip it. No installation is needed. To develop the applet, create a new Java Project and add Path_to_SDK\Schlumberger\Smart Cards and Terminals\Cyberflex Access Kits\v4\Classlibrary\jc_api_212.jar. To develop a client application for your applet also add Path_to_SDK\Schlumberger\Smart Cards and Terminals\Cyberflex Access Kits\v4\Middleware Interfaces\jiop\slbJIop.jar.

Cyberflex Access Software Development Kit

Install the SDK using the installer.

Usage

Develop the client application and the applet in Eclipse and use the SDK to install and instantiate the applet on the card. To install and instantiate the applet, use the Program File Generator which is integrated in the SDK to generate an ijc-file. Afterwards connect to the card, click on "Load Files and Libraries" using the right mouse button and choose "New → Program File". An window will open where you have to specify the ijc-file and the package's AID, which will be explained later. Then click on the program file on the card using the right mouse button and choose "New → Instance". In the new window the applet's AID and the needed memory have to be specified. Afterwards the client can be executed with Eclipse.

Note: Cyberflex cards which have been initialized using the SDK are inaccessible under Linux.

2.3 JavaCard Applet

Applets, which should be executed on the Cyberflex Access egate32K card are programmed with JavaCard. All these Applets should extend the javacard.framework.Applet class and import javacard.framework. If the card should provide cryptographic function, javacard.security and javacard.crypto are also needed.

A host communicates with the applet using APDUs. There are Command APDUs and Response APDUs, which are shown in Figure 2.1, respectively 2.2.

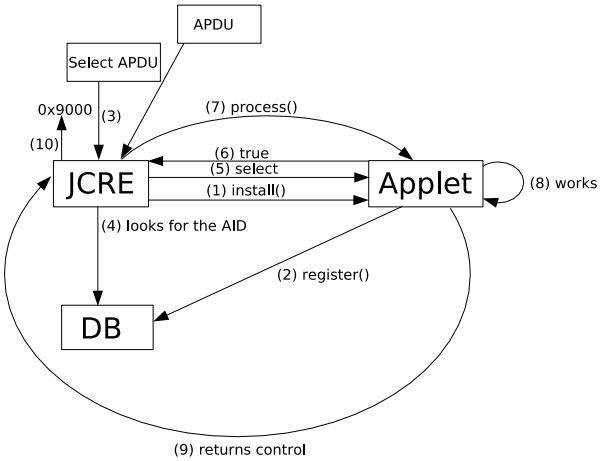


Figure 2.3: Control Flow

As described in [4] the Mandatory Header of a Command APDU consists of a CLA-, INS-, P1- and P2-byte. The CLA-byte is the Class byte, which is used to identify an application on the card. INS stands for Instruction byte. This byte indicates the method to execute. P1 and P2 are parameter bytes, which are not used in our applet.

Lc denotes the number of bytes in the data-field and Le denotes the expected number of bytes in the response APDU's data-field.

The SW1- and SW2-byte in the response APDU denote the card's processing status.

In our cryptographic applet, we needed to specify the CLA-byte and several INS-bytes, which are declared as constants. Additionally we have a constant to limit the pin trials and we introduced two additional error-codes to simplify debugging. Finally, we define a static byte array containing the applet's pin.

The applet's constructor should include every memory allocation which is needed during the execution of this applet.

Additional each applet and each package has its own Application Identifier (AID). With this AID, the package and the applet are identified uniquely. An AID is a sequence between 5 and 16 bytes, whereas the first 5 bytes are the National Registered Application Provider (RID) and the second 11 bytes are the Proprietary Application Identifier Extension (PIX). The RID's allocation is done by the ISO whereas the PIX are allocated by the company which developes an applet.

Furthermore an applet has to implement the install-, select- and process-method.

To understand the necessity of these methods, one has to analyse the control flow. As illustrated in Figure 2.3 the install-method is the first method which is called (1). Install() creates a new instance of the applet. At the end of this process, the applet registers itself (2). To use the applet, it has to be selected. The selection process is started by a select APDU, which is send by the user (3). The JavaCard Runtime Environment (JCRE) looks for the applet's AID in its database (4) and then calls the applet's select method (5). If this method returns true the applet is ready for usage (6).

If a user wants to use a service provided by the applet, he has to send a command APDU. If an APDU arrives at the JCRE, it calls the applet's process method (7). Dependent upon the INS

byte the process method calls the adequate method (8). After executing this method, control is returned to the JCRE (9) which sends a response APDU to the user containing an appropriate SW, e.g. 0x9000 which stands for OK (10).

Our applet provides the following cryptographic services (see Listing A.1):

- Encoding RSA using a 1024 bit private key
- Decoding RSA using a 1024 bit public key
- Encoding and Decoding Triple DES
- Encoding and Decoding DES
- Signing and Verifying SHA-1
- Signing and Ciphering a message using SHA-1 and RSA and vice versa

Problems And Constraints

During the implementation some problems and constraints arose.

- First, the INS byte is not allowed to be equal to 0xC0. There is no apparent reason, but it does not work.
- Second, the usage of the keyword ”this” will cause problems during the generation of the ijc-file.
- Third, to assign the same AID to the same applet every time it is newly generated, the SDK has to be closed before each generation.

2.4 Client Application

A client application has to import slb.iop which is included in the Schlumberger SDK (see Listing B.1). This package provides methods to connect to the card and to exchange ADPUs with it.

The Schlumberger Smart Card Middleware allows client programming using C++, COM, PKCS #11, Java and CSP. We decided to use Java as it seemed to be the most straightforward solution. Figure 2.4 shows the host software architecture and the location of the Java Layer.

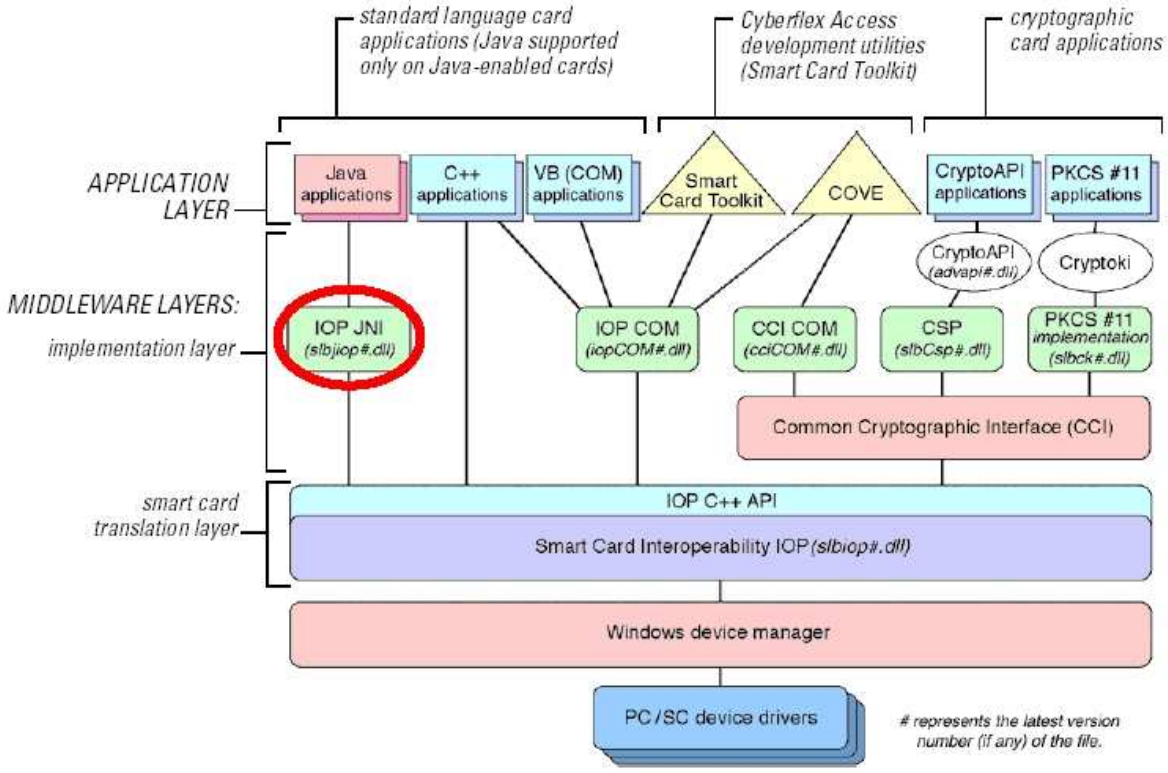


Figure 2.4: Schlumberger's Middleware from [13]

3 Measurements

3.1 Scenario

As we wanted to measure the general peformance of the card, we had to set up several scenarios as listed below:

Scenario 1 (Symmetric Ciphers) The encoding and decoding of the symmetric ciphers like DES and T-DES have been tested using a 8 bytes long input string which was generated randomly (see Listing 3.1). This has been done 50 times for encoding and decoding.

Listing 3.1: Example of a method to measure DES

```
private static void measureDES(MyProjectClient myClient, int count){
  FileWriter des_results = null;
  String inputString ;
  try {
    // create file
     des_results = new FileWriter(" des_results . txt ");
     des_results . write ("#Encoding DES\n");
    long start , stop , dummyStart, dummyStop, dummyTime;
    // generate input
     inputString = generateInput (16) ;
    // measure communication time
    dummyStart = System.currentTimeMillis () ;
    myClient.dummy(inputString);
    dummyStop = System.currentTimeMillis();
    dummyTime = dummyStop − dummyStart;
    // measure des time
    for (int i=0;i<count;i++){
         start = System. currentTimeMillis () ;
        myClient.desEncode( inputString ) ;
        stop = System. currentTimeMillis () ;
         des_results . write ( i+" "+(stop−start−dummyTime)+" ms\n");
    }
     des_results . write ("#Decoding DES\n");
     inputString = generateInput (16) ;
    dummyStart = System.currentTimeMillis () ;
    myClient.dummy(inputString);
    dummyStop = System.currentTimeMillis();
```

```
    dummyTime = dummyStop − dummyStart;
    for (int i=0;i<count;i++){
        start = System.currentTimeMillis();
        myClient.desDecode(inputString);
        stop = System.currentTimeMillis();
        des_results.write(i+" "+(stop−start−dummyTime)+" ms\n");
    }
  }
  catch (IOException e){
    System.out.println("DES: Unable to create file.");
  }
  finally {
    try{
        if (des_results!=null) des_results.close();
    }
    catch (IOException e){}
  }
}
```

Scenario 2 (Asymmetric Ciphers, Signing and Verifying) In this scenario we had to distinguish between encoding and decoding. This is caused by the need of a valid input for the decoding and the flexible length of the input for the encoding.

1. Encoding: We tested RSA, signing SHA-1 and signing and ciphering a message with the input length of 2, 4, 6, 8, 10, 12, 14, 16 and 18 bytes 50 times each. The input strings were generated randomly for each iteration.
2. Decoding: RSA, verifying SHA-1 and decoding and verifying a message require a valid input, which leads to need of generating an input and then decoding it (see Listing 3.2). This is also the reason why we tested these three algorithms only with one input length 50 times. As the JavaCard does not provide a verify method for SHA-1 we had to implement it by ourselfs. In our implementation the given input is divided into the original message and the signature. Afterwards a new signature based on the original message is calculated and then compared with the given signature. The absence of this method is probably caused by the assumption that a signature is only verified by a client and never by the card. Therefore the card is only intended to generate the signatures. The verifying led to another problem. Due to the guidelines by Schlumberger we had to specify the number of bytes the original message contains in our client. Therefore we could test it only with one input length.

Listing 3.2: Example of a method to measure SHA-1

```
private static void measureSHA(MyProjectClient myClient, int maxInputLength, int
    count){
        FileWriter sha_results = null;
```

```
String inputString;
try {
        // create file
        sha_results = new FileWriter("sha_results.txt");
        sha_results.write("#Calculating sha\n");
        long start, stop, dummyStart, dummyStop, dummyTime;
        for (int j=2; j<maxInputLength; j+=2){
                // generate input
                inputString = generateInput(j);
                // measure communication time
                dummyStart = System.currentTimeMillis();
                myClient.dummy(inputString);
                dummyStop = System.currentTimeMillis();
                dummyTime = dummyStop - dummyStart;
                sha_results.write("#input-length = "+j+"\n");
                // measure sha time
                for (int i=0;i<count;i++){
                        start = System.currentTimeMillis();
                        myClient.shadigest(inputString);
                        stop = System.currentTimeMillis();
                        sha_results.write(i+" "+(stop-start-dummyTime)+"
                            ms\n");
                }
        }
        sha_results.write("#Verifying sha\n");
        String input = generateInput(4);
        inputString = myClient.shadigest(input);
        dummyStart = System.currentTimeMillis();
        myClient.dummy(inputString);
        dummyStop = System.currentTimeMillis();
        dummyTime = dummyStop - dummyStart;
        for (int i=0;i<count;i++){
                start = System.currentTimeMillis();
                myClient.shaverify(inputString);
                stop = System.currentTimeMillis();
                sha_results.write(i+" "+(stop-start-dummyTime)+" ms\n");
        }
}
catch (IOException e){
        System.out.println("sha: Unable to create file.");
}
finally {
        try{
                if (sha_results!=null) sha_results.close();
```

```
                }
                catch (IOException e){}
        }
}
```

The client we used for our measurements has been written in Java. In order to ensure that we measure only the time needed to decode or encode, the applet was extended by a dummy method, which returns a received APDU immediately. A new class was introduced, which measures the time between sending and receiving an APDU. The results are written into files, one for each ciphering.

As the card has limited RAM, we need to reset the card after each measurement.

We also had to extend the client by a new class “Measure”, which contains a measure method for each cipher.

The card would also support signing and ciphering APDUs, but we did not test this as it was not needed for our puroses and it reduces the performance.

3.2 Results

3.2.1 Cryptographic Functions

Symmetric Cipher

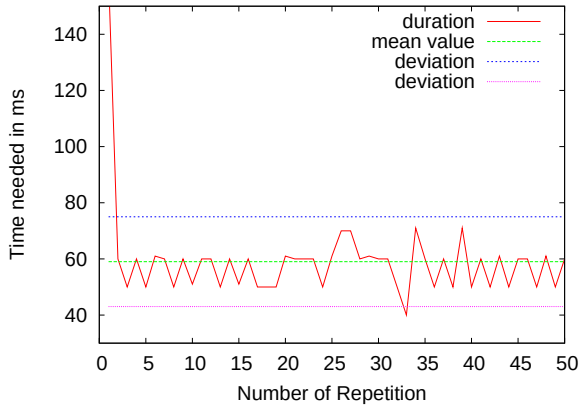


Figure 3.1: Encoding DES

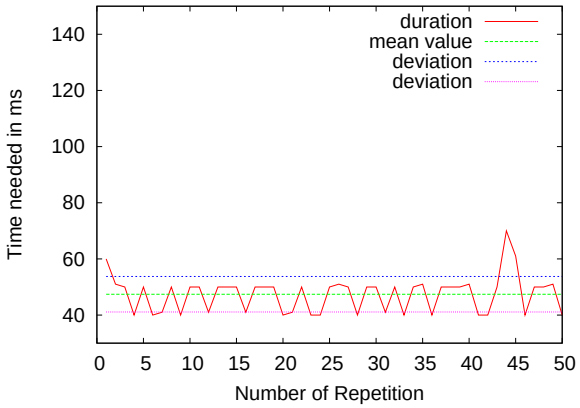


Figure 3.2: Decoding DES

DES As shown in Figure 3.1 and 3.2, DES is a very fast cipher algorithm (see also Table 3.4 and 3.5). The first encryption and decryption needs a little bit more time than the others. A possible reason is that the card has to be initialized for this ciphering. The longer decoding time of the 43th cycle is probably a result of the internal memory management of the card.

Triple DES Encoding and Decoding Triple DES needs nearly the same time, but encoding Triple DES has a higher deviation as can be seen in Figure 3.3 as well as in Table 3.4 and 3.5.

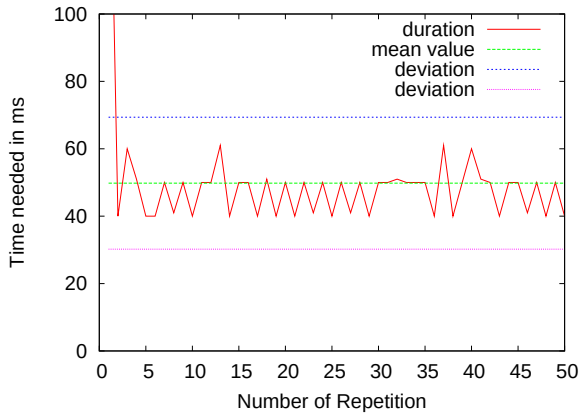


Figure 3.3: Encoding Triple DES

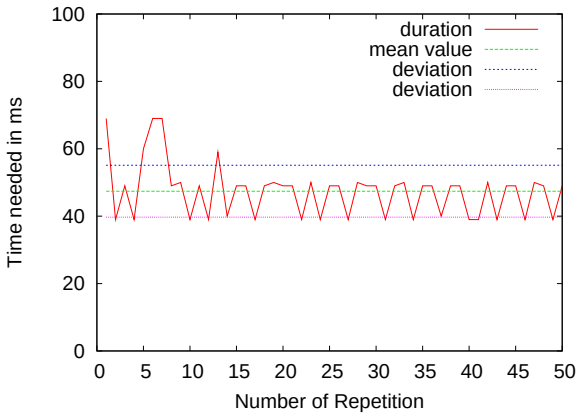


Figure 3.4: Decoding Triple DES

This is caused by the very slow initialization. Another distinctive feature is that early loops require much more time, than later cycles, which might a sign for optimization on the card.

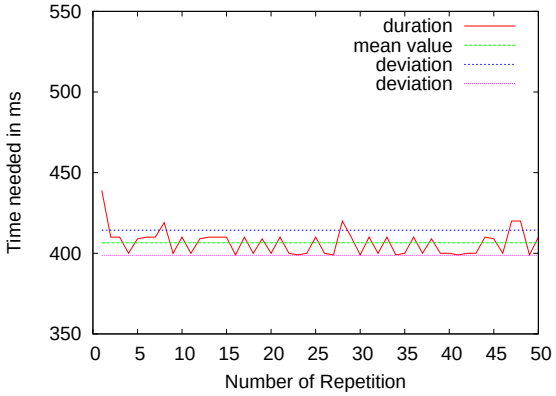
Asymmetric Cipher

RSA Encoding RSA needs nearly the same time for different input lengths as can be seen in Table 3.1. The test with input-length 2 was the first from all RSA-tests. The card had to be initialized which explains the slower first run. All tests have shown that the performance of the SmartCard using RSA ist very slow. It needs about 400ms single RSA encryption, as illustrated in the Figures 3.5(a) - 3.5(d). As shown in Figure 3.6 and Table 3.1, decoding RSA is much more faster than encoding RSA.

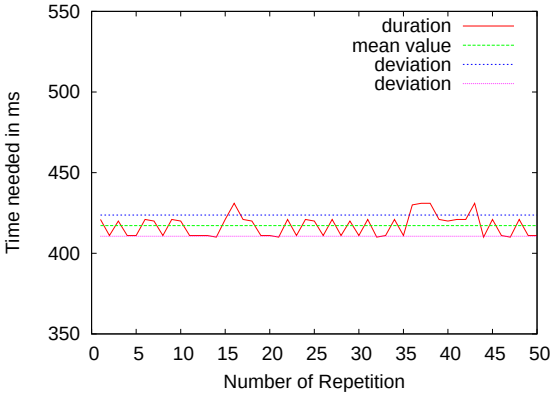
Message Digest

SHA-1 Figures 3.7(a) - 3.7(d) as well as Table 3.2 show the SHA-1 signing algorithm requires nearly the same time for different input lengths. As shown in Figure 3.8 and Table 3.2 the verifying of a SHA-1 digest takes more time than the signing. This is caused by the implementation of the verify method.

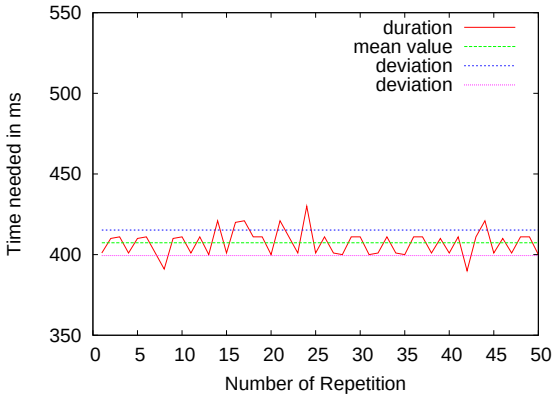
Message Signing And Ciphering In the last test we signed a message using SHA-1 and encrypted it using RSA. Afterwards we decrypted it with RSA and verified the SHA-1 signature. Figures 3.9(a) - 3.9(d) and Table 3.3 illustrate that the input-length has no influence on the time needed. Another distinctive feature is that the extreme values are maximal values, never minimal values. Decoding and verifying a message takes much more time than vice versa as shown in Figure 3.10 as well as in Table 3.3. This is caused by the implementation of the SHA-1 verify method.



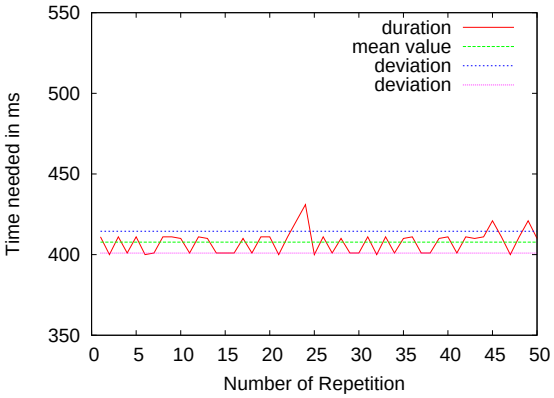
(a) input-length = 2



(b) input-length = 8



(c) input-length = 14



(d) input-length = 18

Figure 3.5: Encoding RSA

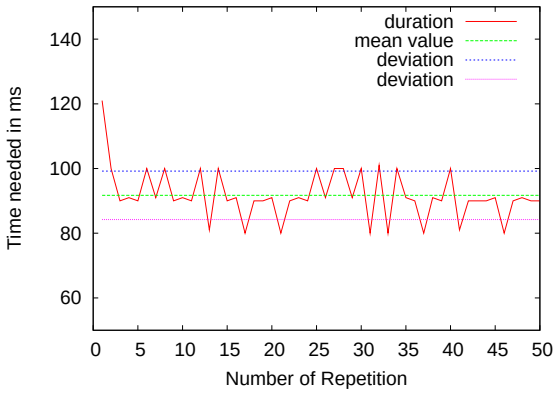
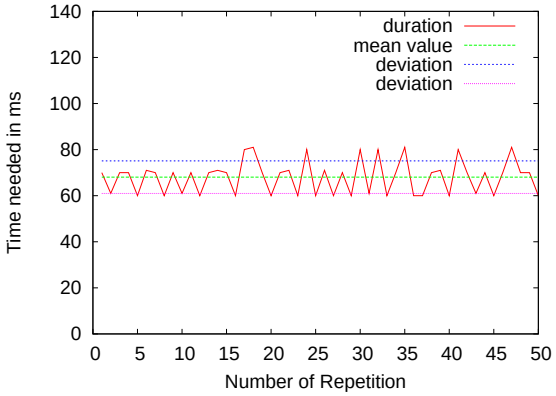


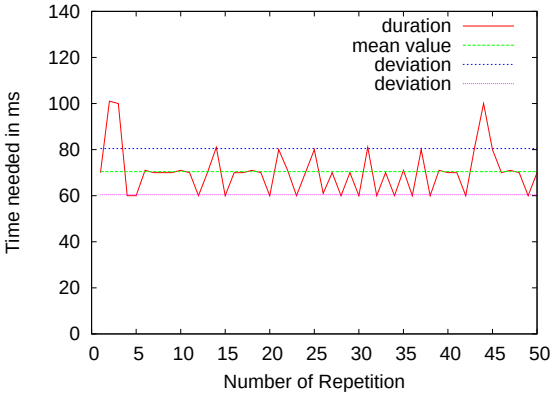
Figure 3.6: Decoding RSA

Type/ input length	Mean Value (ms)	Deviation (ms)
enc/2	406.52	7.83
enc/8	417.14	6.62
enc/14	407.32	7.92
enc/18	407.72	6.78
dec	91.72	7.57

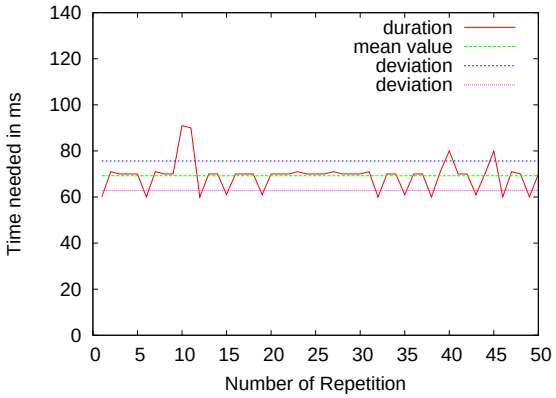
Table 3.1: Encoding and Decoding RSA



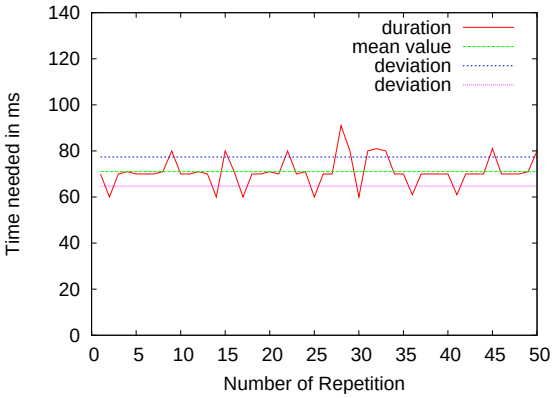
(a) input-length = 2



(b) input-length = 8



(c) input-length = 14



(d) input-length = 18

Figure 3.7: SHA-1 signing

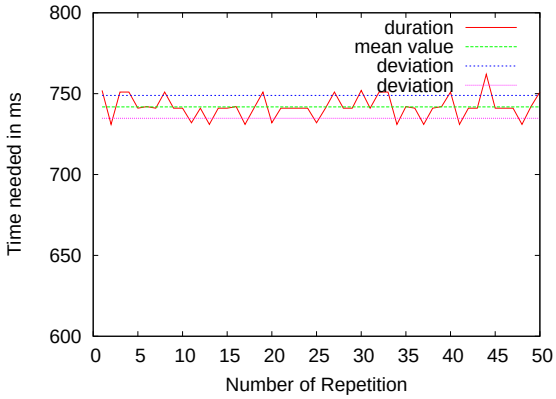
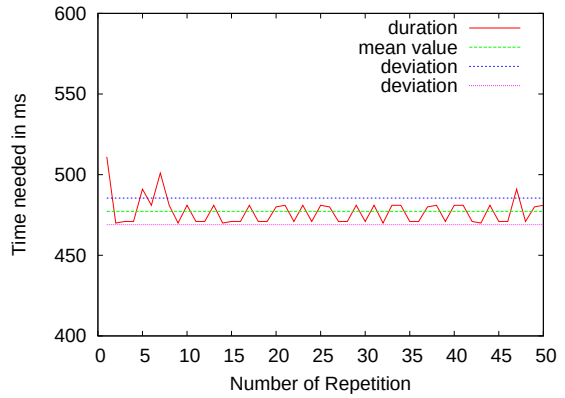


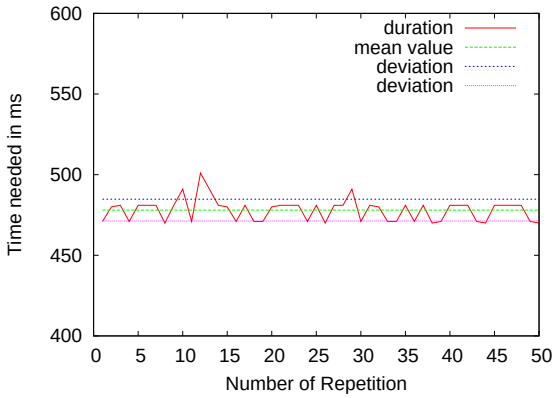
Figure 3.8: SHA-1 verifying

Type/ input length	Mean Value (ms)	Deviation (ms)
enc/2	68.04	7.07
enc/8	70.44	10.10
enc/14	69.24	6.39
enc/18	71.04	6.32
dec	741.82	7.14

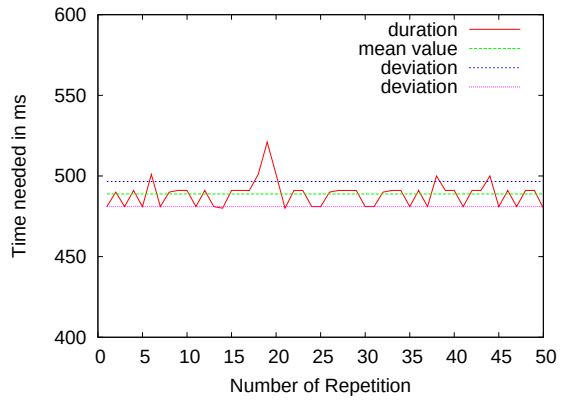
Table 3.2: Signing and Verifying SHA-1



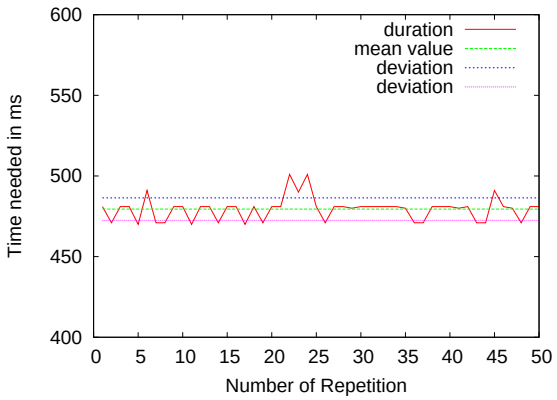
(a) input-length = 2



(b) input-length = 8



(c) input-length = 14



(d) input-length = 18

Figure 3.9: Message Signing and Ciphering

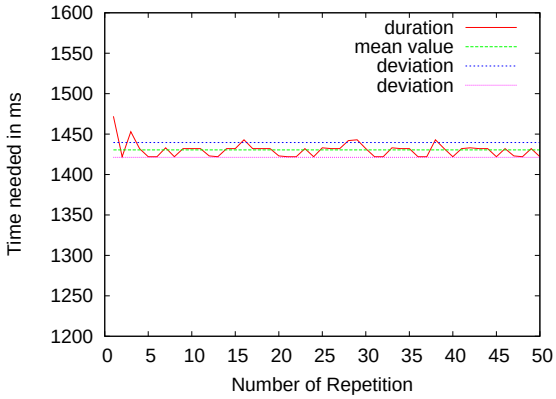


Figure 3.10: Message decoding and verifying

Type/ input length	Mean Value (ms)	Deviation (ms)
enc/2	477.22	8.24
enc/8	478.02	6.80
enc/14	488.82	7.73
enc/18	479.44	6.97
dec	1430.42	9.29

Table 3.3: Message Signing and Ciphering

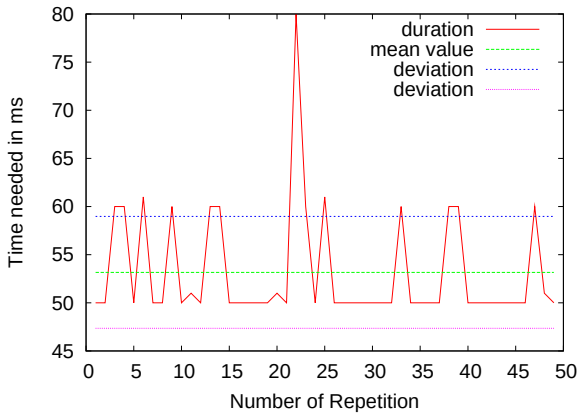


Figure 3.11: Time needed to connect to card

3.2.2 Communication Protocol

Connecting To Card

The time needed to connect to a card is the time without verifying the pin. ”connect to card” concerns only the time needed to establish a secure channel. This includes exchanging the AUTH, MAC and KEK keys which are provided by Schlumberger. The AUTH key is the authentication and encryption key and is used for mutual authentication and for double encryption with the KEK key. The MAC key is the message authentication code key and the KEK key is the key encryption key which is used to encrypt key data. As illustrated in Figure 3.11 it is quite fast as we have a mean value of 52.1 ms and a deviation of 5.85 ms. Possible extreme values are higher than the mean value.

Verifying The PIN

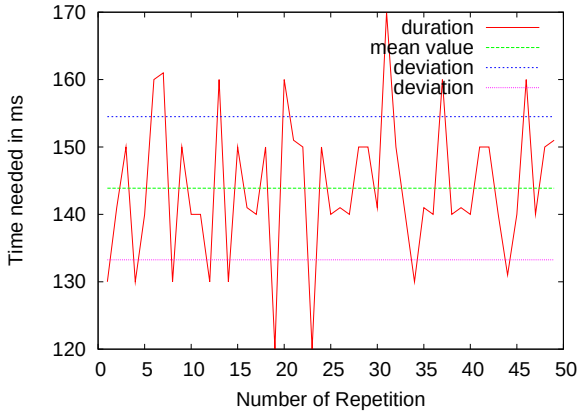


Figure 3.12: Time needed to verify a pin

When a secure channel is established, the pin has to be verified to access the applet. This

takes more time (mean value = 141 ms) than the establishment of a secure channel as can be seen in Figure 3.12. Furthermore there are considerable fluctuations as we have a deviation of 10.9 ms.

Communication With The Card

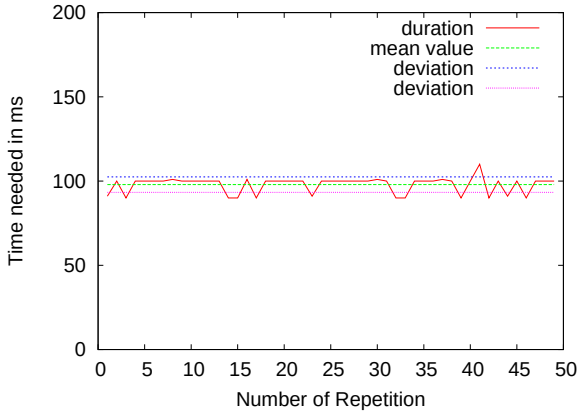


Figure 3.13: Time needed to send an APDU and receive the same APDU

In Figure 3.13 we see the time needed to send an APDU to the card and receive it back. The card only receives this APDU and sends it back immediately. This method was used to measure only the ciphering in the first measurements. This Figure shows, that the communication with the card is very slow, because only sending and receiving an APDU causes a mean value of 95.94 ms (deviation = 4.94 ms).

Resetting The Card

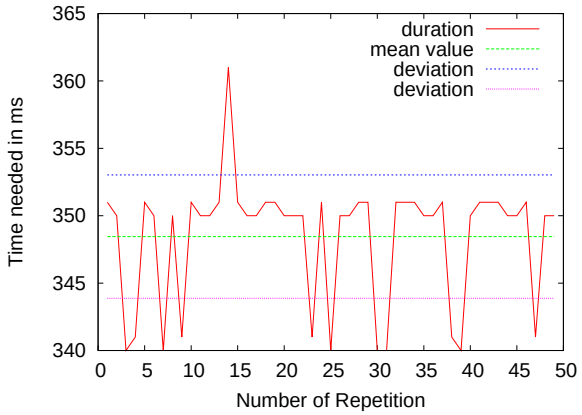


Figure 3.14: Time needed to reset the card

As the card does not support garbage collection and there is no possibility to manually free memory, a full reset of the card has to be done, to free the RAM after every type of ciphering.

Otherwise the card crashes. As shown in Figure 3.14 the reset is extremely slow. We have a mean value of 341.48 ms (deviation = 8.25 ms). That is as long as a RSA encryption.

3.2.3 Comparison

Encodings

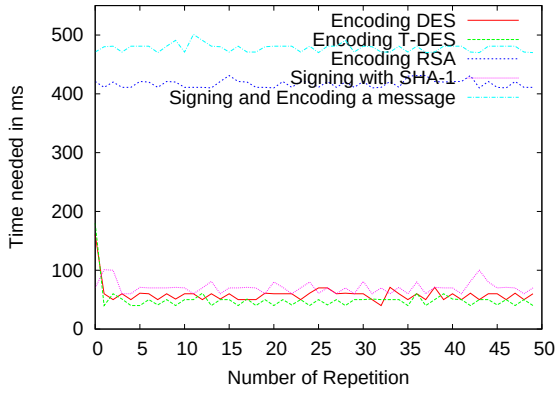


Figure 3.15: Comparison between different ciphers

Algorithm	Mean Value (ms)	Deviation (ms)
DES	59.02	16.14
T-DES	49.80	19.78
Message	478.02	6.80
RSA	417.14	6.62
SHA-1	70.44	10.10

Table 3.4: Comparison of Encodings and Signing

The comparison of all the ciphers shows that RSA behaves differently than the others. Figure 3.15 shows encodings of 8 bytes. DES, Triple DES and SHA-1 need less than 100ms, whereas RSA needs more than 400ms. As expected, signing and encoding a message requires the most time, because it was encoded using RSA after signing it with SHA-1 (see also Table 3.4).

Decodings

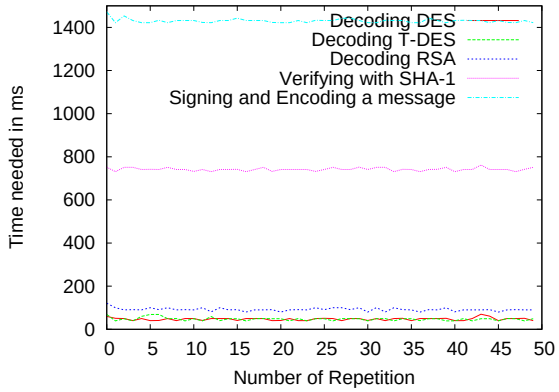


Figure 3.16: Comparison between different ciphers

Algorithm	Mean Value (ms)	Deviation (ms)
DES	47.42	6.39
T-DES	47.40	7.77
Message	1430.42	9.29
RSA	91.72	7.57
SHA-1	741.82	7.14

Table 3.5: Comparison of Decodings and Verifying

As shown in Figure 3.16 decoding DES, Triple DES and RSA is very fast, whereas verifying SHA-1 needs much more time than signing, which is caused by our implementation of this method. Decoding and verifying message needs the most time for the same reasons (Table 3.5).

4 Conclusions

4.1 Issues

There were several problems while programming the applet. These problems were mainly caused by bad documentation. The tutorials just presented the API and the samples were sparse documented. Another problem were the runtime error messages on the card. If the applet caused a JavaCard runtime error, the message was ”6F00” which means ”JVM error has no specified diagnosis”. The error appeared if the applet had not enough memory or if there was an error in our code, so it did not help to find the error. Therefore it was very exhausting to debug the code. Furthermore there was the error ”6124” which is not documented but means, that we forgot to specify the length of the response APDU.

Allocating memory for an applet was another problem because one simply does not know the memory consumption of the JVM for a specific applet. Even in the few examples in the documentation, the programmers obviously did not know the required memory, since the proposed value did not work. As a search in the Internet revealed, we were not alone with this problem. However, the most significant flaw of the card was the missing garbage collection or a mechanism to manually free memory, requiring a periodic reset of the card.

The last problem was the different number representations on the card and in Java. The card stores one hex digit in 32 bit whereas Java uses Unicode, which means that Java stores one hex digit in 16 bit.

4.2 Summary

As seen in the measurements, the card performance is weak. Obviously the CPU resources of the SmartCard are rather limited and of course Java causes additional overhead. This could be a problem in daily use. Another open question is the general stability of the card software. It was not possible to complete the following operations without errors, as can be seen below.

1. encoding RSA with 9 different input lengths, each 50 times, no error
2. decoding RSA 50 times, no error
3. encoding DES 50 times, no error
4. decoding DES 50 times, no error
5. encoding Triple DES 50 times, no error
6. decoding Triple DES 50 times, no error

7. signing SHA-1 with 9 different input lenghts, each 50 times, no error
8. verifying SHA-1 46 times (afterwards 3 error messages)
9. signing and encoding a message with 9 different input lengths, each 50 times
10. decoding and verifying a message caused error messages

To prevent these errors, we have to reset the card after each ciphering, which consumes a lot of time.

The Cybeflex Access egate32K cards support RSA, Triple DES and DES, which is considered insecure. However, the cards are not compliant to the newest JavaCard version 2.2.1 which includes AES, the successor of DES, and dates from November 2003. Furthermore they are not platform independent as they are only usable under Windows. This is why they have a very restricted field of application that is supporting RSA, Triple DES and SHA-1 in a Windows Environment.

Bibliography

[1] O. Kirch, “Smart Cards on Linux,” in *Proceedings of 10th International Linux System Technology Conference*, Saarbrücken, Germany, Oct. 2003.

[2] C.Eckert, *IT-Sicherheit. Konzepte-Verfahren-Protokolle*. oldenbourg-verlag, 2003, ch. 10.3, pp. 387–397.

[3] K. Schmeh, *Safer Net. Kryptografie im Internet und Intranet*. dpunkt.verlag, 1998, pp. 173–181.

[4] Z. Chen. (1999, July) How to write a Java Card applet: A developer’s guide. [Online]. Available: http://www.javaworld.com/javaworld/jw-07-1999/jw-07-javacard_p.html

[5] Axalto. (2004, Oct.) Cryptoflex smart card - The smart card for e-transactions and trusted services. [Online]. Available: http://www.axalto.com/infosec/cryptoflex.asp

[6] L. R. David Corcoran. (2004, May) PSCS Lite. [Online]. Available: http://dennou-k.gaia.h.kyoto-u.ac.jp/library/Linux/debian/pool/main/p/pcsc-lite/

[7] D. Cocoran and C. Osgood. (2004, May) The MuscleCard Framework. [Online]. Available: http://packages.qa.debian.org/m/muscleframework.html

[8] P. E. Chaskiel Grundman. (2003, Feb.) SmartCard Reader Driver for the Schlumberger e-gate adapter for e-gate compatible cards for use with pcsc-lite. [Online]. Available: http://www.luusa.org/~wbx/sc/ifd-egate-0.05-patched.tar.gz

[9] D. Corcoran. (2003, Sept.) MuscleTool. [Online]. Available: http://debian.fapeal.br/debian/pool/main/m/muscletools/

[10] I. Alliance. (2004, Aug.) Client Middleware for Java Cards. [Online]. Available: http://www.identityalliance.com/products.html

[11] S. Microsystems. (2004, Oct.) Java Runtime Environment. [Online]. Available: http://java.sun.com/j2se/1.3/download.html

[12] eclipse.org. (2003, Mar.) Eclipse SDk. [Online]. Available: http://www.eclipse.org

[13] *Guide to Schlumberger Smart Card Middleware*.

A JavaCard Applet Code

Listing A.1: JavaCard Applet Code

```
package csProject ;

// this class provides methods to encrypt
// input data and send back the encrypted
// data

import javacard . framework.*;
import javacard . security .*;
import javacardx . crypto .*;

public class MyProject extends Applet{

        // constants
        final static byte MY_PROJECT_CLA = (byte)0x90;
        final byte PIN_CHECK = (byte)0x10;
        final byte RSA = (byte)0x20;
        final byte DES3 = (byte)0x30;
        final byte DES = (byte)0x40;
        final byte SHA = (byte)0x50;
        final byte RSA_D = (byte)0x60;
        final byte DES3_D = (byte)0x70;
        final byte DES_D = (byte)0x80;
        final byte SHA_V = (byte)0x90;
        final byte SIGN_TEXT = (byte)0xA0;
        final byte VERIFY_TEXT = (byte)0xB0;
        final byte DUMMY = (byte)0xD0;

        final byte PIN_TRY_LIMIT = (byte)0x03;
        final byte MAX_PIN_SIZE = (byte)0x04;
        final short WRONG_PIN = (short)0x6BBB;
        final short WRONG_SHA = (short)0x6CCC;
        static byte[] PIN_STRING =
                {
                        (byte)0x87,(byte)0x65,(byte)0x43,(byte)0x21
                };
```

```
//seeds for the ciphers
static byte[] singleDESKey =
        {
                (byte)0x83,(byte)0x25,(byte)0xAF,(byte)0x49,(byte)0xE4,(
                    byte)0xAB,(byte)0x6F,(byte)0x17
        };
static byte[] tripleDESKey =
        {
                (byte)0x83,(byte)0x25,(byte)0xAF,(byte)0x49,(byte)0xE4,(
                    byte)0xAB,(byte)0x6F,(byte)0x17,
                (byte)0x38, (byte)0x12, (byte)0xA4, (byte)0x19, (byte)0xC6,
                    (byte)0x3B, (byte)0xE7, (byte)0x71
        };

static byte[] rsaPrivateKey = //Note this is 338 bytes long and is a 1024
    bit CRT formatted Private key
                {
                (byte)0xC2, (byte)0x01, (byte)0x05,
                (byte)0xC2, (byte)0x41, (byte)0x00,
                (byte)0xC5, (byte)0xD7, (byte)0x20, (byte)0x28, (byte)0
                    xFA, (byte)0xA7, (byte)0x91, (byte)0x55,
                (byte)0x23, (byte)0xE2, (byte)0x0D, (byte)0xE4, (byte)0
                    x28, (byte)0x7C, (byte)0x65, (byte)0xB7,
                (byte)0x18, (byte)0x59, (byte)0xD9, (byte)0x0D, (byte)0
                    xBA, (byte)0xE7, (byte)0xCF, (byte)0x6A,
                (byte)0xF1, (byte)0xE3, (byte)0x10, (byte)0xC3, (byte)0
                    x7E, (byte)0x48, (byte)0x0D, (byte)0xBC,
                (byte)0x76, (byte)0x7B, (byte)0x04, (byte)0x86, (byte)0
                    xB6, (byte)0x7F, (byte)0xCD, (byte)0x6C,
                (byte)0x84, (byte)0x1A, (byte)0x0B, (byte)0x86, (byte)0
                    xB9, (byte)0x96, (byte)0xAA, (byte)0x83,
                (byte)0x68, (byte)0x63, (byte)0x3C, (byte)0x4D, (byte)0
                    x43, (byte)0x84, (byte)0xB8, (byte)0x6D,
                (byte)0x48, (byte)0xAA, (byte)0xC4, (byte)0xC9, (byte)0
                    x1B, (byte)0x50, (byte)0x47, (byte)0x49,
                (byte)0xC2, (byte)0x41, (byte)0x00,
                (byte)0xC9, (byte)0x5E, (byte)0x4A, (byte)0x08, (byte)0
                    x0E, (byte)0xDC, (byte)0xA5, (byte)0x45,
                (byte)0x90, (byte)0x1C, (byte)0x52, (byte)0xF8, (byte)0
                    x3E, (byte)0xB0, (byte)0x6B, (byte)0x8F,
                (byte)0xCF, (byte)0xEA, (byte)0xA8, (byte)0xF5, (byte)0
                    x1E, (byte)0xAD, (byte)0xD3, (byte)0x82,
                (byte)0x52, (byte)0x30, (byte)0x43, (byte)0x04, (byte)0
                    x9C, (byte)0x25, (byte)0x63, (byte)0x87,
```

(**byte**)0x37, (**byte**)0x20, (**byte**)0x64, (**byte**)0x3F, (**byte**)0
x16, (**byte**)0xB0, (**byte**)0x50, (**byte**)0xCC,
(**byte**)0x38, (**byte**)0x06, (**byte**)0x1B, (**byte**)0xDF, (**byte**)0
xE6, (**byte**)0x78, (**byte**)0xC3, (**byte**)0x99,
(**byte**)0xF7, (**byte**)0xC4, (**byte**)0x3F, (**byte**)0x95, (**byte**)0
x81, (**byte**)0xE0, (**byte**)0x77, (**byte**)0x5C,
(**byte**)0xA3, (**byte**)0x78, (**byte**)0xB8, (**byte**)0x9C, (**byte**)0
x8D, (**byte**)0x9B, (**byte**)0x46, (**byte**)0x5D,
(**byte**)0xC2, (**byte**)0x41, (**byte**)0x00,
(**byte**)0x7F, (**byte**)0xDE, (**byte**)0x17, (**byte**)0x36, (**byte**)0
xDA, (**byte**)0x18, (**byte**)0x1E, (**byte**)0xEF,
(**byte**)0xD0, (**byte**)0x50, (**byte**)0xBD, (**byte**)0x2C, (**byte**)0
x57, (**byte**)0xBE, (**byte**)0x10, (**byte**)0x7B,
(**byte**)0x08, (**byte**)0x7D, (**byte**)0xC6, (**byte**)0xC7, (**byte**)0
xF5, (**byte**)0x76, (**byte**)0xA2, (**byte**)0x59,
(**byte**)0x35, (**byte**)0x7D, (**byte**)0xEA, (**byte**)0xB0, (**byte**)0
x73, (**byte**)0xE6, (**byte**)0x51, (**byte**)0xEB,
(**byte**)0xD3, (**byte**)0x07, (**byte**)0xDD, (**byte**)0x73, (**byte**)0
x56, (**byte**)0x8B, (**byte**)0x76, (**byte**)0x46,
(**byte**)0x3F, (**byte**)0xAE, (**byte**)0x0A, (**byte**)0xB9, (**byte**)0
xA3, (**byte**)0xE3, (**byte**)0x0D, (**byte**)0x71,
(**byte**)0xFB, (**byte**)0xFE, (**byte**)0x55, (**byte**)0x02, (**byte**)0
xF6, (**byte**)0xB6, (**byte**)0x4D, (**byte**)0xC2,
(**byte**)0x79, (**byte**)0x64, (**byte**)0xFD, (**byte**)0x28, (**byte**)0
xBB, (**byte**)0x13, (**byte**)0x23, (**byte**)0xB2,
(**byte**)0xC2, (**byte**)0x41, (**byte**)0x00,
(**byte**)0xA3, (**byte**)0x8D, (**byte**)0xA3, (**byte**)0xED, (**byte**)0
x9C, (**byte**)0xC2, (**byte**)0x18, (**byte**)0xB8,
(**byte**)0x9D, (**byte**)0x10, (**byte**)0x8D, (**byte**)0x51, (**byte**)0
x58, (**byte**)0x52, (**byte**)0xF6, (**byte**)0xB7,
(**byte**)0xB5, (**byte**)0xEE, (**byte**)0xD9, (**byte**)0x2C, (**byte**)0
xAB, (**byte**)0x9E, (**byte**)0x65, (**byte**)0xEF,
(**byte**)0xD0, (**byte**)0x86, (**byte**)0x59, (**byte**)0xDE, (**byte**)0
x73, (**byte**)0xB0, (**byte**)0x57, (**byte**)0x82,
(**byte**)0xBD, (**byte**)0x24, (**byte**)0x17, (**byte**)0xEA, (**byte**)0
xD2, (**byte**)0x46, (**byte**)0xB7, (**byte**)0x69,
(**byte**)0x85, (**byte**)0x90, (**byte**)0x0E, (**byte**)0x85, (**byte**)0
x53, (**byte**)0x3A, (**byte**)0x06, (**byte**)0x3E,
(**byte**)0xDA, (**byte**)0x76, (**byte**)0x67, (**byte**)0x6C, (**byte**)0
xAC, (**byte**)0x6B, (**byte**)0xB5, (**byte**)0x17,
(**byte**)0xCB, (**byte**)0x62, (**byte**)0x39, (**byte**)0x8A, (**byte**)0
xD4, (**byte**)0x04, (**byte**)0xBA, (**byte**)0xD9,
(**byte**)0xC2, (**byte**)0x41, (**byte**)0x00,

```
                    (byte)0x44, (byte)0x03, (byte)0x03, (byte)0xB0, (byte)0
                        x1B, (byte)0x0C, (byte)0xED, (byte)0x09,
                    (byte)0x44, (byte)0xB6, (byte)0x3C, (byte)0x53, (byte)0
                        xBA, (byte)0x20, (byte)0xAE, (byte)0x03,
                    (byte)0xA1, (byte)0xAE, (byte)0xD9, (byte)0x28, (byte)0
                        x09, (byte)0x17, (byte)0x9E, (byte)0xC3,
                    (byte)0x7A, (byte)0x6C, (byte)0xF0, (byte)0x85, (byte)0
                        xC3, (byte)0x13, (byte)0x61, (byte)0xBD,
                    (byte)0x4E, (byte)0xA2, (byte)0x33, (byte)0x19, (byte)0
                        x97, (byte)0xD9, (byte)0x2F, (byte)0x40,
                    (byte)0xFA, (byte)0x7F, (byte)0x1D, (byte)0xB5, (byte)0
                        x0E, (byte)0xCB, (byte)0xA5, (byte)0x0D,
                    (byte)0x00, (byte)0xC1, (byte)0x18, (byte)0xD4, (byte)0
                        xAF, (byte)0x4C, (byte)0x18, (byte)0x24,
                    (byte)0x82, (byte)0xD6, (byte)0x08, (byte)0x4C, (byte)0
                        x60, (byte)0x0B, (byte)0x9C, (byte)0xC5
                    };

static byte[] rsaPublicKey = //Note this is 140 bytes long and is a modulus
    exponent formatted 1024 bit Public key
            {
            (byte)0xC1, (byte)0x01, (byte)0x05,
            (byte)0xC0, (byte)0x81, (byte)0x00,
            (byte)0x9B, (byte)0x9E, (byte)0xC6, (byte)0x74, (byte)0x65, (byte
                )0x5A, (byte)0xBC, (byte)0xBA,
            (byte)0xC6, (byte)0xB0, (byte)0x5D, (byte)0x3C, (byte)0x24, (byte
                )0x3C, (byte)0x90, (byte)0x59,
            (byte)0x7D, (byte)0x5B, (byte)0x6D, (byte)0x03, (byte)0x7B, (byte
                )0xAD, (byte)0x9A, (byte)0xB4,
            (byte)0x58, (byte)0xFE, (byte)0x80, (byte)0x22, (byte)0xEC, (byte
                )0xF0, (byte)0xAB, (byte)0x84,
            (byte)0xF9, (byte)0x07, (byte)0x84, (byte)0x91, (byte)0xE2, (byte
                )0x08, (byte)0xA6, (byte)0x9B,
            (byte)0xED, (byte)0xD7, (byte)0x45, (byte)0xC9, (byte)0xDD, (byte
                )0xBF, (byte)0xF8, (byte)0x7A,
            (byte)0x8B, (byte)0xE1, (byte)0x17, (byte)0xCD, (byte)0x3D, (byte
                )0xD3, (byte)0x6F, (byte)0xB2,
            (byte)0x59, (byte)0x0C, (byte)0x0E, (byte)0x47, (byte)0x0B, (byte
                )0x8E, (byte)0x83, (byte)0xC5,
            (byte)0x0F, (byte)0xAF, (byte)0xD8, (byte)0x21, (byte)0x0C, (byte
                )0xD1, (byte)0x0B, (byte)0xB4,
            (byte)0x24, (byte)0x8D, (byte)0x66, (byte)0xAC, (byte)0x93, (byte
                )0xA3, (byte)0xE4, (byte)0x61,
```

```
            (byte)0xEF, (byte)0x26, (byte)0x50, (byte)0x1C, (byte)0x30, (byte
                )0xED, (byte)0x73, (byte)0xF1,
            (byte)0x92, (byte)0xD8, (byte)0x2C, (byte)0xC6, (byte)0x38, (byte
                )0xD4, (byte)0x6D, (byte)0x81,
            (byte)0x48, (byte)0x2B, (byte)0xCC, (byte)0x42, (byte)0xF8, (byte
                )0x60, (byte)0x61, (byte)0xAD,
            (byte)0x8D, (byte)0x7F, (byte)0x8D, (byte)0x6D, (byte)0x87, (byte
                )0xBF, (byte)0x7D, (byte)0x1C,
            (byte)0x61, (byte)0x2B, (byte)0xC0, (byte)0x42, (byte)0x47, (byte
                )0xDB, (byte)0xDD, (byte)0xC9,
            (byte)0x3F, (byte)0x07, (byte)0x23, (byte)0xE1, (byte)0x3D, (byte
                )0xDA, (byte)0xDB, (byte)0x85,
            (byte)0xC0, (byte)0x04, (byte)0x00,
            (byte)0x01, (byte)0x00, (byte)0x01,
            };

    // instance variables
    OwnerPIN pin;
    Cipher cipherDES;
    Cipher cipherRSA;
    DESKey deskey;
    DESKey des3key;
    RSAPrivateCrtKey rsa_PrivateCrtKey ;
    RSAPublicKey rsa_PublicKey;
    MessageDigest messageDigest;

    // constructor
    private MyProject(byte buffer [], short offset ,byte length){
        pin = new OwnerPIN(PIN_TRY_LIMIT,MAX_PIN_SIZE);
        pin.update(PIN_STRING,(short)0,(byte)4);
        deskey = (DESKey)KeyBuilder.buildKey(KeyBuilder.TYPE_DES,
            KeyBuilder.LENGTH_DES,false);
        des3key = (DESKey)KeyBuilder.buildKey(KeyBuilder.TYPE_DES,
            KeyBuilder.LENGTH_DES3_2KEY,false);
        rsa_PrivateCrtKey = (RSAPrivateCrtKey)KeyBuilder.buildKey(
            KeyBuilder.TYPE_RSA_CRT_PRIVATE,KeyBuilder.
            LENGTH_RSA_1024,false);
        rsa_PublicKey = (RSAPublicKey)KeyBuilder.buildKey(KeyBuilder.
            TYPE_RSA_PUBLIC,KeyBuilder.LENGTH_RSA_1024,false);
        cipherDES = Cipher. getInstance (Cipher.ALG_DES_ECB_NOPAD,false);
        cipherRSA = Cipher. getInstance (Cipher.ALG_RSA_PKCS1, false);
        messageDigest = MessageDigest. getInstance (MessageDigest.ALG_SHA,
            false);
        if ( buffer [ offset ] == 0) {
```

```
                                        register ();
            }
            else {
                                        register (buffer, (short)(offset +1) ,(byte)(
                                            buffer[offset ]));
            }
}

// install method
public static void install (byte buffer [], short offset ,byte length){
            new MyProject(buffer, offset , length);
}

// select method
public boolean select (){
            pin.reset ();
            return true;
}

// process method
public void process(APDU apdu){
            byte apduBuffer[] = apdu.getBuffer ();
            if (selectingApplet ()){
                        ISOException.throwIt(ISO7816.SW_NO_ERROR);
            }
            if (apduBuffer[ISO7816.OFFSET_CLA] != MY_PROJECT_CLA){
                        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
            }
            byte ins = apduBuffer[ISO7816.OFFSET_INS];
            switch (ins){
                        case PIN_CHECK:         pinCheck(apdu); return;
                        case RSA:                       RSAencode(apdu); return;
                        case DES3:                      DES3encode(apdu); return;
                        case DES:                       DESencode(apdu); return;
                        case SHA:                       SHAdigest(apdu); return;
                        case RSA_D:                     RSAdecode(apdu); return;
                        case DES3_D:            DES3decode(apdu); return;
                        case DES_D:                     DESdecode(apdu); return;
                        case SHA_V:                     SHAverify(apdu); return;
                        case SIGN_TEXT:         signText(apdu); return;
                        case VERIFY_TEXT:       verifyText (apdu);return;
                        case DUMMY:                     dummyMethod(apdu);return;
                        default:                ISOException.throwIt(ISO7816.
                            SW_INS_NOT_SUPPORTED);
```

```
        }
}

private void pinCheck(APDU apdu){
        byte apduBuffer[] = apdu.getBuffer();
        short byteRead = (short)(apdu.setIncomingAndReceive());
        if(byteRead != 4) ISOException.throwIt(ISO7816.
            SW_WRONG_LENGTH);
        if(pin.check(apduBuffer,(short)ISO7816.OFFSET_CDATA,(byte)4) ==
            false) ISOException.throwIt(WRONG_PIN);
}

private void RSAencode(APDU apdu){
        byte apduBuffer[] = apdu.getBuffer();
        if( !pin.isValidated() )ISOException.throwIt(ISO7816.
            SW_SECURITY_STATUS_NOT_SATISFIED);
        short byteRead = (short)(apdu.setIncomingAndReceive());
// Create Cipher
        rsa_PrivateCrtKey.setQ(rsaPrivateKey, (short) 6, (short)64);
        rsa_PrivateCrtKey.setP(rsaPrivateKey, (short)73, (short)64);
        rsa_PrivateCrtKey.setPQ(rsaPrivateKey, (short)140, (short)64);
        rsa_PrivateCrtKey.setDQ1(rsaPrivateKey, (short)207, (short)64);
        rsa_PrivateCrtKey.setDP1(rsaPrivateKey, (short)274, (short)64);
        cipherRSA.init(rsa_PrivateCrtKey, Cipher.MODE_ENCRYPT);
        short outbytes = cipherRSA.doFinal(apduBuffer,(short)ISO7816.
            OFFSET_CDATA, byteRead, apduBuffer, (short)ISO7816.
            OFFSET_CDATA);
// Send results
        apdu.setOutgoing();
        apdu.setOutgoingLength((short) outbytes);
        apdu.sendBytesLong(apduBuffer, (short)ISO7816.OFFSET_CDATA, (
            short)outbytes);
}

private void DESencode(APDU apdu){
        byte apduBuffer[] = apdu.getBuffer();
        if (!pin.isValidated()) ISOException.throwIt(ISO7816.
            SW_SECURITY_STATUS_NOT_SATISFIED);
        short byteRead = (short)(apdu.setIncomingAndReceive());
        if (byteRead != 8) ISOException.throwIt(ISO7816.
            SW_WRONG_LENGTH);
        // create cipher
        deskey.setKey(singleDESKey,(short)0);
        cipherDES.init(deskey,Cipher.MODE_ENCRYPT);
```

```
        short outbytes = cipherDES.doFinal(apduBuffer,(short)ISO7816.
            OFFSET_CDATA,byteRead,apduBuffer,(short)ISO7816.
            OFFSET_CDATA);
        //send results
        apdu.setOutgoing();
        apdu.setOutgoingLength(outbytes);
        apdu.sendBytesLong(apduBuffer,(short)ISO7816.OFFSET_CDATA,(short
            )outbytes);
    }

    private void DES3encode(APDU apdu){
        byte apduBuffer[] = apdu.getBuffer();
        if (!pin.isValidated())ISOException.throwIt(ISO7816.
            SW_SECURITY_STATUS_NOT_SATISFIED);
        short byteRead = (short)(apdu.setIncomingAndReceive());
        if (byteRead != 8)ISOException.throwIt(ISO7816.
            SW_WRONG_LENGTH);
        // create cipher
        des3key.setKey(tripleDESKey,(short)0);
        cipherDES.init(des3key,Cipher.MODE_ENCRYPT);
        short outbytes = cipherDES.doFinal(apduBuffer,(short)ISO7816.
            OFFSET_CDATA,byteRead,apduBuffer,(short)ISO7816.
            OFFSET_CDATA);
        //send results
        apdu.setOutgoing();
        apdu.setOutgoingLength((short)outbytes);
        apdu.sendBytesLong(apduBuffer,(short)ISO7816.OFFSET_CDATA,(short
            )outbytes);
    }

    private void SHAdigest(APDU apdu){
        byte apduBuffer[] = apdu.getBuffer();
        if (!pin.isValidated())ISOException.throwIt(ISO7816.
            SW_SECURITY_STATUS_NOT_SATISFIED);
        short byteRead = (short)(apdu.setIncomingAndReceive());
        // create signature
        messageDigest.reset();
        short outbytes = messageDigest.doFinal(apduBuffer,(short)ISO7816.
            OFFSET_CDATA,(short)byteRead,apduBuffer,(short)(ISO7816.
            OFFSET_CDATA+byteRead));
        //send results
        apdu.setOutgoing();
        apdu.setOutgoingLength((short)(byteRead+outbytes));
```

```
            apdu.sendBytesLong(apduBuffer,(short)ISO7816.OFFSET_CDATA,(short
                )(outbytes+byteRead));
    }

    private void RSAdecode(APDU apdu){
            byte apduBuffer[] = apdu.getBuffer();
            if ( ! pin.isValidated() )ISOException.throwIt (ISO7816.
                SW_SECURITY_STATUS_NOT_SATISFIED);
            short byteRead = (short)(apdu.setIncomingAndReceive());
            // create cipher
            rsa_PublicKey.setModulus(rsaPublicKey, (short)6, (short)128);
            rsa_PublicKey.setExponent(rsaPublicKey, (short)137, (short)3);
            cipherRSA.init(rsa_PublicKey, Cipher.MODE_DECRYPT);
            short outbytes = cipherRSA.doFinal(apduBuffer,(short)ISO7816.
                OFFSET_CDATA, byteRead, apduBuffer, (short)ISO7816.
                OFFSET_CDATA);
        // Send results
            apdu.setOutgoing();
            apdu.setOutgoingLength((short) outbytes);
            apdu.sendBytesLong(apduBuffer, (short)ISO7816.OFFSET_CDATA, (
                short)outbytes);
    }

    private void DES3decode(APDU apdu){
            byte apduBuffer[] = apdu.getBuffer();
            if (!pin.isValidated())ISOException.throwIt(ISO7816.
                SW_SECURITY_STATUS_NOT_SATISFIED);
            short byteRead = apdu.setIncomingAndReceive();
            if (byteRead != 8)ISOException.throwIt(ISO7816.
                SW_WRONG_LENGTH);
            // create cipher
            des3key.setKey(tripleDESKey,(short)0);
            cipherDES.init(des3key,Cipher.MODE_DECRYPT);
            short outbytes = cipherDES.doFinal(apduBuffer,(short)ISO7816.
                OFFSET_CDATA,byteRead,apduBuffer,(short)ISO7816.
                OFFSET_CDATA);
            //send results
            apdu.setOutgoing();
            apdu.setOutgoingLength((short)outbytes);
            apdu.sendBytesLong(apduBuffer,(short)ISO7816.OFFSET_CDATA,(short
                )outbytes);
    }

    private void DESdecode(APDU apdu){
```

```
        byte apduBuffer[] = apdu.getBuffer();
        if (!pin.isValidated())ISOException.throwIt(ISO7816.
            SW_SECURITY_STATUS_NOT_SATISFIED);
        short byteRead = apdu.setIncomingAndReceive();
        if (byteRead != 8)ISOException.throwIt(ISO7816.
            SW_WRONG_LENGTH);
        // create cipher
        deskey.setKey(singleDESKey,(short)0);
        cipherDES.init(deskey,Cipher.MODE_DECRYPT);
        short outbytes = cipherDES.doFinal(apduBuffer,(short)ISO7816.
            OFFSET_CDATA,byteRead,apduBuffer,(short)ISO7816.
            OFFSET_CDATA);
        //send results
        apdu.setOutgoing();
        apdu.setOutgoingLength((short)outbytes);
        apdu.sendBytesLong(apduBuffer,(short)ISO7816.OFFSET_CDATA,(short
            )outbytes);
    }

    private void SHAverify(APDU apdu){
        byte apduBuffer[] = apdu.getBuffer();
        if (!pin.isValidated())ISOException.throwIt(ISO7816.
            SW_SECURITY_STATUS_NOT_SATISFIED);
        short byteRead = apdu.setIncomingAndReceive();
        byte in_sha[] = new byte[(byte)20];
        byte text[] = new byte[(byte)(byteRead-20)];
        byte out_sha[] = new byte[(byte)20];
        for (short i=0;i<(short)(byteRead-20); i++){
            text[i]=apduBuffer[(short)(ISO7816.OFFSET_CDATA+i)];
        }
        short j=0;
        for (short i=(short)(byteRead-20);i<byteRead;i++){
            in_sha[j++] = apduBuffer[(short)(ISO7816.OFFSET_CDATA+i
                )];
        }
        messageDigest.reset();
        messageDigest.doFinal(text,(short)0,(short)(byteRead-20),out_sha,(
            short)0);
        boolean same_sha=true;
        for (short i=0;i<(short)20;i++){
            if (in_sha[i]!=out_sha[i])same_sha=false;
        }
        if (!same_sha)ISOException.throwIt(WRONG_SHA);
    }
```

```
private void signText(APDU apdu){
        byte apduBuffer[] = apdu.getBuffer();
        if (!pin.isValidated())ISOException.throwIt(ISO7816.
            SW_SECURITY_STATUS_NOT_SATISFIED);
        short byteRead = (short)(apdu.setIncomingAndReceive());
        // create signature
        messageDigest.reset();
        short digest = messageDigest.doFinal(apduBuffer,(short)ISO7816.
            OFFSET_CDATA,(short)byteRead,apduBuffer,(short)(ISO7816.
            OFFSET_CDATA+byteRead));
        //Create Cipher
        rsa_PrivateCrtKey.setQ(rsaPrivateKey, (short) 6, (short)64);
        rsa_PrivateCrtKey.setP(rsaPrivateKey, (short)73, (short)64);
        rsa_PrivateCrtKey.setPQ(rsaPrivateKey, (short)140, (short)64);
        rsa_PrivateCrtKey.setDQ1(rsaPrivateKey, (short)207, (short)64);
        rsa_PrivateCrtKey.setDP1(rsaPrivateKey, (short)274, (short)64);
        cipherRSA.init(rsa_PrivateCrtKey, Cipher.MODE_ENCRYPT);
        short outbytes = cipherRSA.doFinal(apduBuffer,(short)ISO7816.
            OFFSET_CDATA, (short)(byteRead+digest), apduBuffer, (short)
            ISO7816.OFFSET_CDATA);
        // Send results
        apdu.setOutgoing();
        apdu.setOutgoingLength((short)outbytes);
        apdu.sendBytesLong(apduBuffer, (short)ISO7816.OFFSET_CDATA, (
            short)outbytes);
}

private void verifyText(APDU apdu){
        byte apduBuffer[] = apdu.getBuffer();
        if (!pin.isValidated())ISOException.throwIt(ISO7816.
            SW_SECURITY_STATUS_NOT_SATISFIED);
        short byteRead = (short)(apdu.setIncomingAndReceive());
        // create cipher
        rsa_PublicKey.setModulus(rsaPublicKey, (short)6, (short)128);
        rsa_PublicKey.setExponent(rsaPublicKey, (short)137, (short)3);
        cipherRSA.init(rsa_PublicKey, Cipher.MODE_DECRYPT);
        short outbytes = cipherRSA.doFinal(apduBuffer,(short)ISO7816.
            OFFSET_CDATA, byteRead, apduBuffer, (short)ISO7816.
            OFFSET_CDATA);
        // create signature
        byte encipheredBuffer[]=new byte[(byte)outbytes];
        for (short i=0;i<outbytes;i++){
```

```
                encipheredBuffer[i]=apduBuffer[(short)(ISO7816.
                    OFFSET_CDATA+i)];
            }
            byteRead = (short) encipheredBuffer.length;
            byte in_sha[] = new byte[(byte)20];
            byte text[] = new byte[(byte)(byteRead-20)];
            byte out_sha[] = new byte[(byte)20];
            for (short i=0;i<(short)(byteRead-20);i++){
                text[i]=encipheredBuffer[i];
            }
            short j=0;
            for (short i=(short)(byteRead-20);i<byteRead;i++){
                in_sha[j++] = encipheredBuffer[i];
            }
            messageDigest.reset();
            messageDigest.doFinal(text,(short)0,(short)(byteRead-20),out_sha,(
                short)0);
            boolean same_sha=true;
            for (short i=0;i<(short)20;i++){
                if (in_sha[i]!=out_sha[i])same_sha=false;
            }
            if (!same_sha)ISOException.throwIt(WRONG_SHA);
        }

        private void dummyMethod(APDU apdu){
            byte apduBuffer[] = apdu.getBuffer();
            if (!pin.isValidated())ISOException.throwIt(ISO7816.
                SW_SECURITY_STATUS_NOT_SATISFIED);
            short byteRead = apdu.setIncomingAndReceive();
            apdu.setOutgoing();
            apdu.setOutgoingLength(byteRead);
            apdu.sendBytesLong(apduBuffer,(short)ISO7816.OFFSET_CDATA,(short
                )byteRead);
        }
}
```

B Client Application Code

Listing B.1: Client Application Code

```
/*
 * Created on 17.11.2004
 *
 */
package myMeasurements;

import slb.iop.*;

/**
 * @author Carolin Latze (01-129-436)
 *
 * This is a client for the JavaCard Applet.
 *
 */
public class MyProjectClient {

        private boolean connected;
        private IOP sIOP;
        private SmartCard iopCard;
        private short aid [];
        private boolean readyForTests;

        public MyProjectClient(short myAID[]){
                this.connected = false;
                this.sIOP = new IOP();
                this.iopCard = new SmartCard();
                this.aid = new short[myAID.length];
                for (int i=0; i<myAID.length; i++){
                        this.aid[i] = myAID[i];
                }
                this.readyForTests = false;
        }

        public void listMyReaders(){
                String [] readers = this.sIOP.ListReaders();
                for (int i=0;i<readers.length;i++){
```

```
                System.out.println(readers[i]+"\n");
            }
        }

        public void connectToCard(int readerNr){
            String s;
            if (!this.connected) this.iopCard = new SmartCard();
            String[] listReaders = this.sIOP.ListReaders();
            if (listReaders.length > 0) {
                boolean result = this.sIOP.Connect(this.iopCard, listReaders
                    [readerNr], false);
                if (result){
                    this.connected = true;
                    System.out.println("Successfully connected to card
                        .\n");
                }
                else {
                    System.out.println("Unable to connect to card.\n");
                }
            }
            else {
                this.connected = false;
                this.iopCard = new SmartCard();
                System.out.println("No reader available \n");
            }
        }

        public void verifyPIN(String pin){
            int cla, ins, p1, p2;
            int iArray[] = new int[4];
            String error, pinSubstring;
            if (!this.connected){
                System.out.println("Connect to card first .\n");
                return;
            }
            System.out.println("Selecting Applet ....\ n");
            try {
                boolean result = this.iopCard.SelectAID(this.aid);
                if (!result){
                    error = this.iopCard.GetErrorMessage();
                    System.out.println("SelectAID failed for the
                        following reason: "+error+"\n");
                    return;
                }
```

```
}
catch (slbException e){
        error = e.getMessage();
        System.out.println("SelectAID failed for the following
            reason: "+error+"\n");
        return;
}
System.out.println("Applet selected.\n");
System.out.println("Verifying PIN ....\n");
try {
        cla = 0x90;
        ins = 0x10;
        p1 = 0;
        p2 = 0;
        int length = pin.length();
        if (length != 8){
                System.out.println("Pin-length must be 8.\n");
        }
        else {
                for (int i=0; i<4; i++){
                        pinSubstring = pin.substring(i*2,(i*2)+2);
                        iArray[i] = Integer.valueOf(pinSubstring,
                            16).intValue();
                }
                this.iopCard.SendCardAPDU(cla,ins,p1,p2,iArray,0);
                int iErrorCode = this.iopCard.GetLastErrorCode();
                if (iErrorCode != 0x9000){
                        if (iErrorCode == 0x6300){
                                System.out.println("Wrong PIN.\n");
                        }
                        else {
                                error = this.iopCard.
                                    GetErrorMessage();
                                System.out.println("SendCardAPDU
                                    failed for the following reason: "
                                    +error+"\n");
                        }
                }
                else {
                        System.out.println("PIN verified.\n");
                        this.readyForTests = true;
                        System.out.println("Ready for tests!\n");
                }
        }
```

```
        }
        catch (slbException e){
                error = e.getMessage();
                System.out.println("VerifyPIN failed for the following
                    reason: "+error+"\n");
        }

}

public void reset(){
        try {
                this.iopCard.ResetCard();
        }
        catch (slbException e){
                System.out.println(e.getMessage());
        }
}

public String rsaEncode(String input){
        int cla, ins, p1, p2;
        int iArray[];
        short sArray[];
        String error, inputSubstring;
        if (!this.connected){
                System.out.println("Connect to card first.\n");
                return null;
        }
        if ((input.length()%2)!=0){
                System.out.println("Input-length has to be even.\n");
                return null;
        }
        try {
                cla = 0x90;
                ins = 0x20;
                p1 = 0;
                p2 = 0;
                int length = input.length();
                if (length < 1){
                        System.out.println("Input has to be longer than 0.\
                            n");
                        return null;
                }
                else {
                        iArray = new int[input.length() /2];
```

```
                    for (int i=0; i<input.length() /2; i++){
                        inputSubstring = input.substring(i*2,(i*2)
                            +2);
                        iArray[i] = Integer.valueOf( inputSubstring,
                            16).intValue();
                    }
                    sArray = this.iopCard.SendCardAPDU(cla,ins,p1,p2,
                        iArray,128);
                    // output is 128 bytes long
                    int iErrorCode = this.iopCard.GetLastErrorCode();
                    if (iErrorCode != 0x9000){
                        error = this.iopCard.GetErrorMessage();
                        System.out.println("SendCardAPDU failed for
                            the following reason: "+error+"\n");
                        return null;
                    }
                    else {
                        byte sBytes[] = new byte[sArray.length];
                        for (int i=0; i<sArray.length;i++){
                            sBytes[i] = (byte)sArray[i];
                        }
                        String result = byteArrayToHexString(sBytes
                            );
                        return result;
                    }
                }
            }
            catch (slbException e){
                error = e.getMessage();
                System.out.println("rsaEncode failed for the following
                    reason: "+error+"\n");
                return null;
            }
        }

        public void rsaDecode(String input){
            int cla, ins, p1, p2;
            int iArray[];
            short sArray[];
            String error, inputSubstring;
            if (!this.connected){
                System.out.println("Connect to card first.\n");
                return;
            }
```

```
        if (input.length()!=256){
            System.out.println("Input-length has to be 256.\n");
            return;
        }
        try {
            cla = 0x90;
            ins = 0x60;
            p1 = 0;
            p2 = 0;
            iArray = new int[input.length() /2];
            for (int i=0; i<input.length() /2; i++){
                inputSubstring = input.substring(i*2,(i*2)+2);
                iArray[i] = Integer.valueOf(inputSubstring, 16).
                    intValue();
            }
            sArray = this.iopCard.SendCardAPDU(cla,ins,p1,p2,iArray,5);
                //you should know, how many digits to you expect
            int iErrorCode = this.iopCard.GetLastErrorCode();
            if (iErrorCode != 0x9000){
                error = this.iopCard.GetErrorMessage();
                System.out.println("SendCardAPDU failed for the
                    following reason: "+error+"\n");
            }
            else {
                byte sBytes[] = new byte[sArray.length];
                for (int i=0; i<sArray.length;i++){
                    sBytes[i] = (byte)sArray[i];
                }
                String result = byteArrayToHexString(sBytes);
            }
        }
        catch (slbException e){
            error = e.getMessage();
            System.out.println("rsaDecode failed for the following
                reason: "+error+"\n");
        }
    }

    public void des3Encode(String input){
        int cla, ins, p1, p2;
        int iArray[];
        short sArray[];
        String error, inputSubstring;
        byte sBytes[] = null;
```

```
            if (!this.connected){
                System.out.println("Connect to card first.\n");
                return;
            }
            if (input.length()!=16){
                System.out.println("Input-length has to be 16.\n");
                return;
            }
            try {
                cla = 0x90;
                ins = 0x30;
                p1 = 0;
                p2 = 0;
                iArray = new int[input.length()/2];
                for (int i=0; i<input.length()/2; i++){
                    inputSubstring = input.substring(i*2,(i*2)+2);
                    iArray[i] = Integer.valueOf(inputSubstring, 16).
                        intValue();
                }
                sArray = this.iopCard.SendCardAPDU(cla,ins,p1,p2,iArray,8);
                int iErrorCode = this.iopCard.GetLastErrorCode();
                if (iErrorCode != 0x9000){
                    error = this.iopCard.GetErrorMessage();
                    System.out.println("SendCardAPDU failed for the
                        following reason: "+error+"\n");
                }
                else {
                    sBytes = new byte[sArray.length];
                    for (int i=0; i<sArray.length;i++){
                        sBytes[i] = (byte)sArray[i];
                    }
                    String result = byteArrayToHexString(sBytes);
                }
            }
            catch (slbException e){
                error = e.getMessage();
                System.out.println("des3Encode failed for the following
                    reason: "+error+"\n");
            }
        }

        public void des3Decode(String input){
            int cla, ins, p1, p2;
            int iArray[];
```

```
            short sArray[];
            String error, inputSubstring;
            if (!this.connected){
                System.out.println("Connect to card first.\n");
                return;
            }
            if (input.length()!=16){
                System.out.println("Input-length has to be 16.\n");
                return;
            }
            try {
                cla = 0x90;
                ins = 0x70;
                p1 = 0;
                p2 = 0;
                iArray = new int[input.length()/2];
                for (int i=0; i<input.length()/2; i++){
                    inputSubstring = input.substring(i*2,(i*2)+2);
                    iArray[i] = Integer.valueOf(inputSubstring, 16).
                        intValue();
                }
                sArray = this.iopCard.SendCardAPDU(cla,ins,p1,p2,iArray,8);
                int iErrorCode = this.iopCard.GetLastErrorCode();
                if (iErrorCode != 0x9000){
                    error = this.iopCard.GetErrorMessage();
                    System.out.println("SendCardAPDU failed for the
                        following reason: "+error+"\n");
                }
                else {
                    byte sBytes[] = new byte[sArray.length];
                    for (int i=0; i<sArray.length;i++){
                        sBytes[i] = (byte)sArray[i];
                    }
                    String result = byteArrayToHexString(sBytes);
                }
            }
            catch (slbException e){
                error = e.getMessage();
                System.out.println("des3Decode failed for the following
                    reason: "+error+"\n");
            }
        }

        public void desEncode(String input){
```

```
int cla, ins, p1, p2;
int iArray [];
short sArray [];
String error, inputSubstring;
byte sBytes[] = null;
if (! this.connected){
        System.out.println ("Connect to card first .\n");
        return;
}
if (input.length()!=16){
        System.out.println ("Input-length has to be 16.\n");
        return;
}
try {
        cla = 0x90;
        ins = 0x40;
        p1 = 0;
        p2 = 0;
        iArray = new int[input.length() /2];
        for (int i=0; i<input.length() /2; i++){
                inputSubstring = input.substring(i*2,(i*2)+2);
                iArray[i] = Integer.valueOf(inputSubstring, 16).
                    intValue();
        }
        sArray = this.iopCard.SendCardAPDU(cla,ins,p1,p2,iArray,8);
        int iErrorCode = this.iopCard.GetLastErrorCode();
        if (iErrorCode != 0x9000){
                error = this.iopCard.GetErrorMessage();
                System.out.println ("SendCardAPDU failed for the
                    following reason: "+error+"\n");
        }
        else {
                sBytes = new byte[sArray.length];
                for (int i=0; i<sArray.length;i++){
                        sBytes[i] = (byte)sArray[i];
                }
                String result = byteArrayToHexString(sBytes);
        }
}
catch (slbException e){
        error = e.getMessage();
        System.out.println ("desEncode failed for the following
            reason: "+error+"\n");
}
```

```
}

public void desDecode(String input){
        int cla, ins, p1, p2;
        int iArray [];
        short sArray [];
        String error, inputSubstring;
        if (! this.connected){
                System.out.println ("Connect to card first .\n");
                return;
        }
        if (input.length()!=16){
                System.out.println ("Input-length has to be 16.\n");
                return;
        }
        try {
                cla = 0x90;
                ins = 0x80;
                p1 = 0;
                p2 = 0;
                iArray = new int[input.length() /2];
                for (int i=0; i<input.length() /2; i++){
                        inputSubstring = input.substring (i*2,(i*2)+2);
                        iArray[i] = Integer.valueOf( inputSubstring , 16).
                            intValue ();
                }
                sArray = this.iopCard.SendCardAPDU(cla,ins,p1,p2,iArray,8);
                int iErrorCode = this.iopCard.GetLastErrorCode();
                if (iErrorCode != 0x9000){
                        error = this.iopCard.GetErrorMessage();
                        System.out.println ("SendCardAPDU failed for the
                            following reason: "+error+"\n");
                }
                else {
                        byte sBytes[] = new byte[sArray.length];
                        for (int i=0; i<sArray.length;i++){
                                sBytes[i] = (byte)sArray[i];
                        }
                        String result = byteArrayToHexString(sBytes);
                }
        }
        catch (slbException e){
                error = e.getMessage();
```

```
                System.out.println("desDecode failed for the following
                    reason: "+error+"\n");
            }
        }

        public String shadigest(String input){
            int cla, ins, p1, p2;
            int iArray[];
            short sArray[];
            String error, inputSubstring;
            byte sBytes[] = null;
            if (!this.connected){
                System.out.println("Connect to card first.\n");
                return null;
            }
            if ((input.length()%2)!=0){
                System.out.println("Input-length has to be even.\n");
                return null;
            }
            try {
                cla = 0x90;
                ins = 0x50;
                p1 = 0;
                p2 = 0;
                iArray = new int[input.length()/2];
                for (int i=0; i<input.length()/2; i++){
                    inputSubstring = input.substring(i*2,(i*2)+2);
                    iArray[i] = Integer.valueOf(inputSubstring, 16).
                        intValue();
                }
                sArray = this.iopCard.SendCardAPDU(cla,ins,p1,p2,iArray,
                    input.length()/2+20);
                int iErrorCode = this.iopCard.GetLastErrorCode();
                if (iErrorCode != 0x9000){
                    error = this.iopCard.GetErrorMessage();
                    System.out.println("SendCardAPDU failed for the
                        following reason: "+error+"\n");
                    return null;
                }
                else {
                    sBytes = new byte[sArray.length];
                    for (int i=0; i<sArray.length;i++){
                        sBytes[i] = (byte)sArray[i];
                    }
```

```
                String result = byteArrayToHexString(sBytes);
                return result;
            }
        }
        catch (slbException e){
            error = e.getMessage();
            System.out.println("shadigest failed for the following
                reason: "+error+"\n");
            return null;
        }
    }

    public void shaverify(String input){
        int cla, ins, p1, p2;
        int iArray[];
        short sArray[];
        String error, inputSubstring;
        byte sBytes[] = null;
        if (!this.connected){
            System.out.println("Connect to card first.\n");
            return;
        }
        if ((input.length()%2)!=0){
            System.out.println("Input-length has to be even.\n");
            return;
        }
        try {
            cla = 0x90;
            ins = 0x90;
            p1 = 0;
            p2 = 0;
            iArray = new int[input.length()/2];
            for (int i=0; i<input.length()/2; i++){
                inputSubstring = input.substring(i*2,(i*2)+2);
                iArray[i] = Integer.valueOf(inputSubstring, 16).
                    intValue();
            }
            this.iopCard.SendCardAPDU(cla,ins,p1,p2,iArray,0);
            int iErrorCode = this.iopCard.GetLastErrorCode();
            if (iErrorCode != 0x9000){
                error = this.iopCard.GetErrorMessage();
                System.out.println("SendCardAPDU failed for the
                    following reason: "+error+"\n");
            }
```

```
                else {
                }
        }
        catch (slbException e){
                error = e.getMessage();
                System.out.println("shaverify failed for the following
                    reason: "+error+"\n");
        }
}

public String signMessage(String input){
        int cla, ins, p1, p2;
        int iArray [];
        short sArray [];
        String error, inputSubstring;
        byte sBytes[] = null;
        if (!this.connected){
                System.out.println("Connect to card first .\n");
                return null;
        }
        if ((input.length()%2)!=0){
                System.out.println("Input-length has to be even.\n");
                return null;
        }
        try {
                cla = 0x90;
                ins = 0xA0;
                p1 = 0;
                p2 = 0;
                iArray = new int[input.length() /2];
                for (int i=0; i<input.length() /2; i++){
                        inputSubstring = input.substring(i*2,(i*2)+2);
                        iArray[i] = Integer.valueOf(inputSubstring, 16).
                            intValue();
                }
                sArray = this.iopCard.SendCardAPDU(cla,ins,p1,p2,iArray
                    ,128);
                int iErrorCode = this.iopCard.GetLastErrorCode();
                if (iErrorCode != 0x9000){
                        error = this.iopCard.GetErrorMessage();
                        System.out.println("SendCardAPDU failed for the
                            following reason: "+error+"\n");
                        return null;
                }
```

```
                else {
                        sBytes = new byte[sArray.length];
                        for (int i=0; i<sArray.length;i++){
                                sBytes[i] = (byte)sArray[i];
                        }
                        String result = byteArrayToHexString(sBytes);
                        return result;
                }
        }
        catch (slbException e){
                error = e.getMessage();
                System.out.println("signMessage failed for the following
                    reason: "+error+"\n");
                return null;
        }
}

public void verifyMessage(String input){
        int cla, ins, p1, p2;
        int iArray[];
        short sArray[];
        String error, inputSubstring;
        byte sBytes[] = null;
        if (!this.connected){
                System.out.println("Connect to card first.\n");
                return;
        }
        if ((input.length()%2)!=0){
                System.out.println("Input-length has to be even.\n");
                return;
        }
        try {
                cla = 0x90;
                ins = 0xB0;
                p1 = 0;
                p2 = 0;
                iArray = new int[input.length()/2];
                for (int i=0; i<input.length()/2; i++){
                        inputSubstring = input.substring(i*2,(i*2)+2);
                        iArray[i] = Integer.valueOf(inputSubstring, 16).
                            intValue();
                }
                this.iopCard.SendCardAPDU(cla,ins,p1,p2,iArray,0);
                int iErrorCode = this.iopCard.GetLastErrorCode();
```

```
                if (iErrorCode != 0x9000){
                        error = this.iopCard.GetErrorMessage();
                        System.out.println("SendCardAPDU failed for the
                            following reason: "+error+"\n");
                }
                else {
                }
        }
        catch (slbException e){
                error = e.getMessage();
                System.out.println("verifyMessage failed for the following
                    reason: "+error+"\n");
        }
}

public void dummy(String input){
        int cla, ins, p1, p2;
        int iArray [];
        short sArray [];
        String error, inputSubstring;
        byte sBytes[] = null;
        if (! this.connected){
                System.out.println("Connect to card first.\n");
                return;
        }
        if ((input.length()%2)!=0){
                System.out.println("Input-length has to be even.\n");
                return;
        }
        try {
                cla = 0x90;
                ins = 0xD0;
                p1 = 0;
                p2 = 0;
                iArray = new int[input.length() /2];
                for (int i=0; i<input.length() /2; i++){
                        inputSubstring = input.substring(i*2,(i*2)+2);
                        iArray[i] = Integer.valueOf(inputSubstring, 16).
                            intValue();
                }
                sArray = this.iopCard.SendCardAPDU(cla,ins,p1,p2,iArray,
                    input.length() /2);
                int iErrorCode = this.iopCard.GetLastErrorCode();
                if (iErrorCode != 0x9000){
```

```
                        error = this.iopCard.GetErrorMessage();
                        System.out.println("SendCardAPDU failed for the
                            following reason: "+error+"\n");
                    }
                    else {
                        sBytes = new byte[sArray.length];
                        for (int i=0; i<sArray.length;i++){
                            sBytes[i] = (byte)sArray[i];
                        }
                        String result = byteArrayToHexString(sBytes);
                    }
                }
                catch (slbException e){
                    error = e.getMessage();
                    System.out.println("dummy failed for the following reason:
                        "+error+"\n");
                }
        }

        // helper methods
        private String byteArrayToHexString(byte[] byteArray){
                String strArray = new String();
                strArray = "";
                for (int x=0; x < byteArray.length; x++) {
                    int b = ((int)byteArray[x] & 0x000000ff);
                    if (b < 16){
                        strArray = strArray + "0" + Integer.toHexString(b).
                            toUpperCase();
                    }
                    else{
                        strArray = strArray + Integer.toHexString(b).
                            toUpperCase();
                    }
                }
                return strArray;
        }

}
```