Travaux Dirigés
Programmation JavaCARD
ESIGETEL
2004-2005

Objectifs

Découvrir l'outil de développement Cyberflex, programmer une Applet JavaCard, programmer un client avec l'interface JIOP.

1 Préambule :Environnement de développement

1.1 Installation des SDK nécessaires

1.1.1 Java Card 2.2.1

La carte à puce utilise une API Java particulière. Nous téléchargerons le Java Card 2.1.2 Development Kit, qui contient cette API, sa documentation, et des outils divers que nous n'utiliserons pas (génération d'images de type CAP pour les cartes compatibles, émulation Ce téléchargement est uniquement nécessaire si l'on veut avoir la javadoc de l'API de la

1.1.2 jdk 1.3.2_6

Il est nécessaire pour compiler le .class utilisé par le Program File Generator.

1.1.3 Schlumberger Cyberflex Access SDK 4.3 : Drivers

L'installation du kit de développement de Schlumberger est nécessaire pour transférer notre application sur la carte à puce. Le kit fourni le driver Windows pour le lecteur. A noter que certains fabricants de lecteurs de carte à puce fournissent un driver sous la forme d'un fichier ".jar". Dans ce cas, le driver est utilisable sur n'importe quel système d'exploitation. Ici nous ne pourront développer que sous Windows avec le driver fourni.

1.2 Vérification Schlumberger Cyberflex Access SDK 4.3 : Outil de Diagnostic

Le kit de développement de Schlumberger fourni un petit utilitaire permettant de vérifier l'état du driver PC/SC (un standard que suit le driver de Schlumberger) :

- si le driver du lecteur est bien installé, dans ce cas, le lecteur figure dans la branche
- si le service de "lecteur de carte à puce" est bien démarré *(panneau de configuration, outils d'administration, services, carte à puce)*. Dans ce cas, les notifications d'insertion et de désinsertion sont bien transmises au driver. Il en résulte l'apparition d'informations sur la carte insérée (Branche connected readers, puis en cliquant sur le lecteur qui accueille la carte).

Cet outil se trouve par défaut ici :

```
"C:\Program Files\Schlumberger\Smart\_Cards\_and\_Terminals\
Smart\_Card\_Readers\Tools\pcscdiag.exe"
```

2 Exercice 1 : Création d'une applet pour la carte à puce

Pour cette applet, vous implanterez une fonctionnalité très simple d'écho des données envoyées. L'applet réceptionne le buffer APDU dans un tableau de 256 octets, puis elle renvoie les données.

2.1 Déclaration de "package"

```
package fr.esigetel.rcm;
```

2.2 et inclusions

```
import javacard.framework.APDU;
import javacard.framework.Applet;
import javacard.framework.ISO7816;
import javacard.framework.ISOException;
import javacard.framework.Util;
```

2.3 la classe CryptApplet

```
public class CryptApplet extends Applet {
```

2.4 les déclarations de variables

```
% private byte[] cryptBytes;
% private short cryptOffset = (short) 0;
% private static final short LENGTH_ECHO_BYTES = 256;
final static byte CRYPT_CLA = (byte) 0x80;
final static byte CRYPT = (byte) 0x20;
final static byte DECRYPT = (byte) 0x30;
% private short i = (short) 0;
```

2.4.1 le constructeur

```
protected CryptApplet() {
cryptBytes = new byte[LENGTH_ECHO_BYTES];
register();
}
```

2.4.2 La méthode d'installation de l'applet

```
public static void install(byte[] bArray, short bOffset, byte bLength) {
new CryptApplet();
}
```

2.4.3 La méthode de traitement des operations (Servant)

```
public void process(APDU apdu) {

byte buffer[] = apdu.getBuffer();
if (selectingApplet()) {
ISOException.throwIt(ISO7816.SW_NO_ERROR);
}
byte ins = buffer[ISO7816.OFFSET_INS];
// appel de la methode specifiee par INS
switch (ins) {
case CRYPT :
crypt(apdu);
break;
case DECRYPT :
decrypt(apdu);
break;
default :
ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
}
}
```

Le Decryptage consiste a ajouter 1 a chaque byte

```
public void crypt(APDU apdu) {
```

1. lecture du buffer apdu : *méthode getBuffer() de la classe apdu*
2. on récupère le nombre d'octets dans le buffer : *méthode setIncomingAndReceive() de la classe apdu*
3. cryptage : incrément de 1 de chaque valeur du buffer
4. transfert du buffer crypté
   (a) on dit au system que le traitement des données est terminé et que l'on va préparer un buffer de retour *méthode setOutgoing() de la classe apdu*
   (b) définir la taille des données dans le buffer de retour : *méthode setOutgoingLength() de la classe apdu*
   (c) envoi du buffer de données *méthode sendBytes() de la classe apdu*

Le Decryptage consiste a retirer 1 a chaque byte

2.5 Génération du fichier "bytecode" .class

Vous compilerez le code JAVA avec l'option "-g" comme cela est recommandé par Sun, de plus il est vivement déconseillé d'utiliser l'option -0 qui optimise la vitesse d'exécution, car dans le cas de la javacard, cela ne peut qu'avoir un effet néfaste. Le compilateur javac est celui du jdk 1.3 (toute autre version étant à exclure sous peine de bug du logiciel de génération de Schlumberger). Le fichier ".jar" javacard doit être dans le classpath, vous mettrez celui fourni par Schlumberger plutôt que celui de Sun.

2.6 Adaptation du bytecode .class au système de Schlumberger

Vous utiliserez le programme fourni par Schlumberger, et non celui fourni par Sun (converter). En effet les fichiers générés par converter (CAP, EXP, JCA) ne peuvent pas être chargés sur la carte. Cet outil créé donc un fichier dans un format spécifique.

Avec un fichier issu d'une compilation effectuée avec le jdk 1.3 la conversion s'effectue correctement.

3 Exercice 3 : Développement d'un client

Envoi de requêtes APDU via le lecteur de carte

Le lecteur de carte est fourni avec un driver PC/SC pour Windows. Pour communiquer avec ce driver, il y a deux solutions, une ouverte (l'interface *OpenCARD*, et une autre propriétaire fournie par Schlumberger. "Slb".

3.1 Environnement

Pour utiliser l'API propriétaire Schlumberger, il est nécessaire d'inclure les fichier ".jars" et les bibliothèques natives suivantes :

- slbJIop.jar

3.2 Listeners d'insertion/suppression de carte

Il est possible d'être averti de l'ajout ou de la suppression de la carte ou du lecteur. Ceci se fait très simplement par le biais d'un Listener (design pattern observateur). Pour cela, il faut créer un objet IOP et appeler la méthode *iop.addIOPListener(IOPListener listener)*

En fournissant notre implémentation de l'interface *«IOPListener»*. Cette interface se présente comme suit :

4 le client

```
public class Client {
static public void main(String args[]){
}
```

5 les importations du client

```
import slb.iop.*;
```

5.1 la méthode main

Vous allez créer un objet IOP qui va etre l'interface entre l'application cliente et la carte. Un

```
slb.iop.IOP sIOP = new slb.iop.IOP();
```

Vous allez demander la liste des lecteurs et l'afficher

```
String[] listReaders = sIOP.listReaders();
```

Vous allez créer un objet *Listener* qui va gérer les évènements qui se produisent de type carte et lecteur de carte Pour cela vous allez créer la classe *RcmIOP* qui implante les méthodes de l'interface *IOPListener*.

5.2 la classe RcmIOP

```
class RCMIop implements slb.iop.IOPListener {
public void CardInserted(slb.iop.IOPEvent event){
public void CardRemoved(slb.iop.IOPEvent event){
public void ReaderInserted(slb.iop.IOPEvent event){
public void ReaderRemoved(slb.iop.IOPEvent event){
}
```

Implantation des méthodes ...

1. détection de l'insertion d'un lecteur

   ```
   public void ReaderInserted(IOPEvent event) {
   System.out.println("Lecteur connecté");
   }
   ```

2. retrait du lecteur

   ```
   public void ReaderRemoved(IOPEvent event) {
   System.out.println("Lecteur déconnecté");
   }
   ```

3. insertion de carte

   ```
   public void CardInserted(IOPEvent event) {
   System.out.println("Carte connectée");
   connected = iop.Connect(smartCard, event.getReaderName(), false);
   if (connected) {
   }
   ```

4. retrait de carte

   ```
   public void CardRemoved(IOPEvent event) {
   System.out.println("Carte déconnectée");
   ```

5.3 création et enregistrement du listener

```
RCMIop rcml = new RCMIop();
sIOP.addIOPListener(rcml);
```

Vous allez créer un objet SmartCard qui va vous permettre de communiquer avec la carte (envoi et réception de trames apdu). Vous pourrez dans un deuxième temps créer une classe RCMCard qui va implanter les méthodes de l'interface *SmartCardListener*, cette classe va détecter tous les évènement qui vont changer l'état de la carte (transmission de données vers et depuis la carte).

```
slb.iop.SmartCard maCard = new slb.iop.SmartCard();
RCMCard rcm = new RCMCard();

class RCMCard implements slb.iop.SmartCardListener {
}
```

5.4 Connexion de la carte

```
if (listReaders.length> 0) {
result = sIOP.Connect(maCard, listReaders[1], false);
if (result) {
System.out.println ("Carte connectee");
}
}
```

5.5 Ouverture d'un canal sécurisé

```
try {
System.out.println ("Transaction EstablishSecureChannel");
maCard.BeginTransaction();
short[] MACKey;
short[] autKey;
short[] KEKKey;
MACKey = stringToShortArray("404142434445464748494A4B4C4D4E4F");
autKey = stringToShortArray("404142434445464748494A4B4C4D4E4F");
KEKKey = stringToShortArray("404142434445464748494A4B4C4D4E4F");
connected = maCard.EstablishSecureChannel(MACKey, autKey, KEKKey);
if (connected) {
System.out.println ("Canal securise cree");
}
maCard.EndTransaction();
}
catch(slb.iop.slbException e){ System.out.println ("Pb d ouverture de canal");}
```

5.6 Sélection de l'applet

```
CLA = (int) Integer.parseInt("00", 16);
INS = (int) Integer.parseInt("A4", 16);
P1 = (int) Integer.parseInt("04", 16);
```

```
P2 = (int) Integer.parseInt("00", 16);
    body=stringToIntArray("11223344556677");
LE = (int) Integer.parseInt("00", 16);
try {
System.out.println ("Applet Selection");
maCard.BeginTransaction();
maCard.SendCardAPDU(CLA,INS,P1,P2,body,LE);
System.out.println ("Applet selected");
maCard.EndTransaction();
}
catch(slb.iop.slbException e){ System.out.println ("Pb de Selection");}
```

5.7 Envoi d'une trame APDU

5.7.1 Trame Verify PIN

```
System.out.println ("Envoi de l'APDU VerifyPIN");
CLA = (int) Integer.parseInt("80", 16);
INS = (int) Integer.parseInt("40", 16);
P1 = (int) Integer.parseInt("00", 16);
P2 = (int) Integer.parseInt("00", 16);
    body=stringToIntArray("00000000");
LE = (int) Integer.parseInt("00", 16);

try {
maCard.BeginTransaction();
maCard.SendCardAPDU(CLA,INS,P1,P2,body,LE);
System.out.println ("Verify PIN ok");
maCard.EndTransaction();
}
catch(slb.iop.slbException e){ System.out.println ("Pb de VerifyPIN");e.printStackTrace();}
```

5.7.2 Envoi d'une trame View Balance

5.7.3 Envoi d'une trame Deposit

5.7.4 Envoi d'une trame View Balance