



Travaux pratiques Java Card

TP n°1 – Applet faisant un écho de l'entrée

TP n°2 – Porte monnaie électronique simple sécurisé via un PIN

TP n°3 – Utilisation de la cryptographie : Réalisation d'une applet Java Card et du client

TP n°4 – Réalisation via RMI d'un porte monnaie électronique

Nous utiliserons dans cette série de TP l'environnement Eclipse avec le plugin JCOP déjà activé.

Je rappelle le fondement de Java, il n'y a pas d'arithmétique sur les types autres que numériques (et en Java Card sur les entiers). Pensez aux règles de base de génie logiciel, pas de constantes en dur dans le code, nommer correctement vos variables, etc.

Pour tous les TP, nous utiliserons des JCOP.

Pour charger dans la carte, on utilisera suivant les TP et les besoins soit Eclipse.

Procédure de création d'une application :

Donner un nom à votre projet, sélectionner les profils de votre carte (en général JC2.1.1 et OP2.0.1), à votre package et à votre applet. Donner également un AID à votre package et à l'applet (en le dérivant de celui du package).

Écrire le programme Java correspondant à la spécification. Pour ce faire, créer un projet Java Card (un projet par package). Le nom du projet et/ou des packages et applets n'ont pas de réelle importance c'est plus pour vous y retrouver.

Une fois l'application codée, vous la chargez sur la carte au travers du Run Configuration. Double cliquez sur Java Card Application et sélectionner le lecteur de carte comme destination et en avant « Run ».

À l'aide du shell de JCOP vous pouvez échanger avec la carte via des commandes prédéfinies ou directement vos APDU. Vérifiez le bon fonctionnement de votre application.

Si vous rencontrez des erreurs n'hésitez pas à me les signaler :

damien.sauveron@unilim.fr

Applet faisant un écho de l'entrée

Objectif : Connaissance des outils de gestion des applications (loader, debugger, script), de la chaîne de compilation et de conversion, écriture d'une applet Java Card, définition de l'AID, utilisation de l'API `javacard.framework` et compréhension de la communication carte/lecteur (réception de commande APDU et d'envoi de réponse APDU et de Status Word – SW).

Les spécifications de cette applet sont les suivantes :

- Si les deux premiers octets de la commande APDU sont égaux à 0x80 alors la carte renverra seulement 0x6B 0x00 (pour cela on utilisera la classe `ISOException`).
- Sinon on réalise un écho de l'APDU reçu.

Voici le code minimale d'une applet en Java Card.

```
package fr.cryptis.nullapp;
import javacard.framework.*;
public class NullApp extends Applet
{
    protected NullApp()
    {
        // Insérer ici les initiatisations
    }

    public static void install(byte[] bArray, short bOffset, byte bLength) throws ISOException
    {
        new NullApp().register();
    }

    public void process(APDU apdu) throws ISOException
    {
        // Insérer ici le code métier
    }
}
```

On compilera puis convertira l'applet d'écho obtenue pour ensuite la charger sur la carte. Son installation se fait en deux phases qui sont le chargement et l'instanciation (appel à la méthode statique `install`).

Porte monnaie électronique simple sécurisé via un PIN

Objectif : Connaissance des outils de gestion des applications (loader, debugger, script), de la chaîne de compilation et de conversion, écriture d'une applet Java Card, définition de l'AID, utilisation de l'API `javacard.framework`.

Les spécifications de cette applet sont les suivantes :

- Elle comportera une balance (le solde) exprimé sous forme d'un `short` qui devra toujours rester positive et qui ne dépassera pas la limite maximal du format de stockage.
- Elle comportera un PIN qui devra être vérifié pour les opérations de crédit et de débit. On utilisera pour ce PIN la classe `OwnerPin` qui fait tout le travail de gestion du PIN pour nous et de manière sécurisée. Nous le fixerons à 8 octets la longueur et à 3 le nombre maximal d'essai consécutifs faux. La valeur du PIN sera fixée lors de l'opération d'installation afin de pouvoir le personnaliser pour chaque carte.
- Elle acceptera 4 commandes `GET_BALANCE`, `CREDIT`, `DEBIT` et `VERIFY` :
 - `GET_BALANCE` renverra le solde ;

GET BALANCE APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0x80	0x50	0x0	0x0	N/A	N/A	2
Response APDU						
Optional data		Status word		Meaning of status word		
Balance value		0x9000		Successful processing		

- `VERIFY` permettra de proposer un PIN à vérifier ;

VERIFY APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0x80	0x20	0x0	0x0	Length of the PIN data	PIN data	N/A
• CLA byte denotes the structure of the command • INS byte (0x20) indicates a verify instruction • P1 and P2 are not used, and are both set to 0 • The data field contains the PIN value						
Optional data		Status word		Meaning of status word		
N/A		0x9000		Successful processing		
		0x6300		Verification failed		

- CREDIT permettra de créditer la balance et ne fonctionnera que si le PIN a été vérifié avec succès ;

CREDIT APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0x80	0x30	0x0	0x0	1	Credit amount	N/A
The data field contains the credit amount						
Response APDU						
Optional data		Status word		Meaning of status word		
N/A		0x9000		Successful processing		
		0x6301		PIN verification required		
		0x6A83		Invalid credit amount		
		0x6A84		Exceed the maximum amount		

- DEBIT permettra de débiter la balance et ne fonctionnera que si le PIN a été vérifié avec succès.

DEBIT APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0x80	0x40	0x0	0x0	1	Debit amount	N/A
The data field contains the debit amount						
Response APDU						
Optional data		Status word		Meaning of status word		
N/A		0x9000		Successful processing		
		0x6301		PIN verification required		
		0x6A83		Invalid debit amount		
		0x6A85		Negative balance		

Il suffit donc de programmer cette application simple. Vous trouverez dans la classe `Util` des méthodes intéressantes pour la manipulation de `short` et de tableau de `bytes`. L'interface `ISO7816` dans le paquetage `javacard.framework` contient déjà beaucoup des constantes dont vous aurez besoin pour retourner les erreurs.

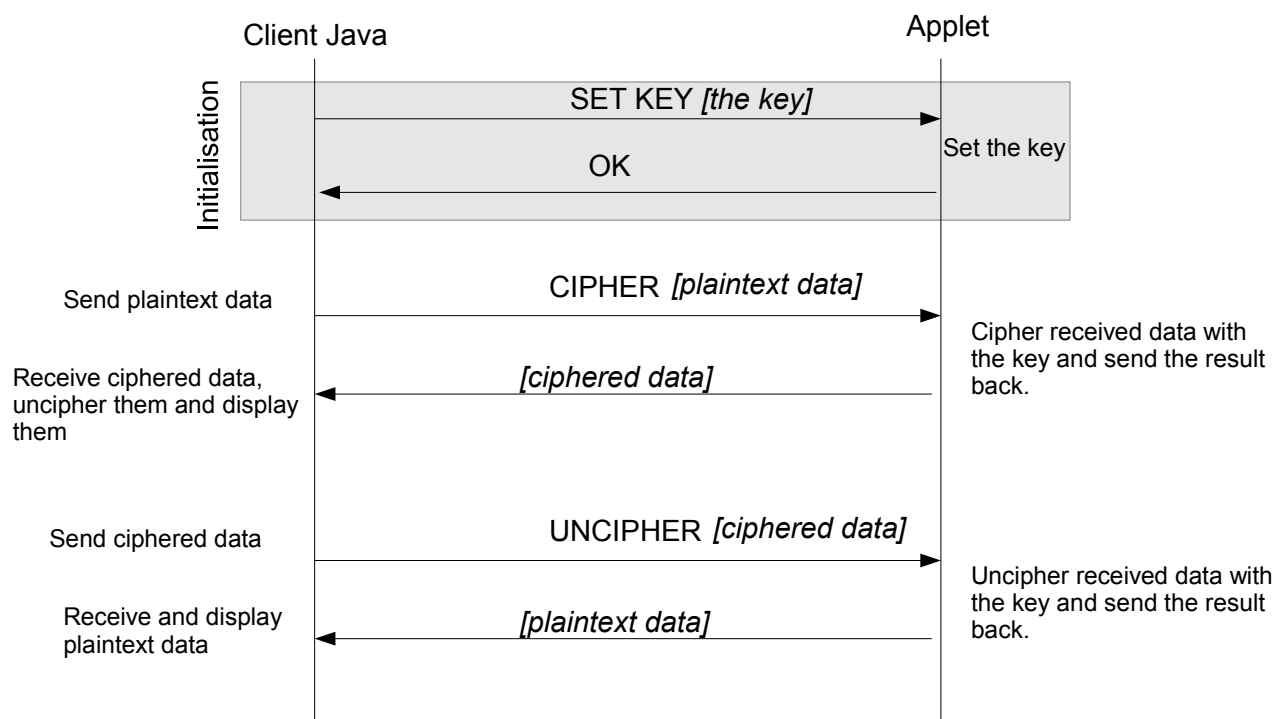
Utilisation de la cryptographie : Réalisation d'une applet Java Card et du client

Objectif : Manipulation d'un algorithme de chiffrement coté Java Card et intégration dans un client Java.

Dans ce TP nous allons utiliser les primitives cryptographiques pour réaliser le chiffrement et le déchiffrement de messages par la carte. Pour cela nous implémenterons un protocole simple dans lequel la carte et le client possèdent une clé symétrique 3 DES de 192 bits qui sera utilisée pour chiffrer et déchiffrer des messages de 8 octets en utilisant l'algorithme ALG_DES_CBC_NOPAD. Cet algorithme utilise la clé de 192 bits et travaille sur des blocs de 64 bits avec un vecteur d'initialisation de même taille que les blocs.

1) Écrire un client Java afin qu'il puisse :

- initialiser l'application avec la clé symétrique utilisée.
- envoyer une donnée de 8 octets en clair à l'applet. Cette donnée sera chiffrée par l'applet et renvoyée au terminal qui la déchiffre et l'affiche.
- envoyer une donnée chiffrée à l'applet qui la déchiffre et retourne la donnée en clair. Cette donnée sera affichée par le terminal.



a) Il faut tout d'abord définir les commandes APDU nécessaires. On peut par exemple utiliser des constantes pour les différentes instructions (respectivement dans l'ordre du schéma 0x20, 0x30 et 0x40).

b) Il faut également importer les packages suivants :

```
import javax.crypto.*;
import javax.crypto.spec.DESedeKeySpec;
import javax.crypto.spec.IvParameterSpec;
```

c) Il faut déclarer un tableau d'octets qui a la taille de la clé et qui contient la valeur de la clé secrète.

d) Il faut déclarer le vecteur d'initialisation initialisé à 0 car l'algorithme du 3DES côté carte repose sur un vecteur d'initialisation qui est toujours égal à 0 à cause de l'espace mémoire limité.

e) Avant les opérations de chiffrement ou de déchiffrement on réalisera les étapes suivantes :

- créer un objet DESedeKeySpec correspondant à la clé secrète qui a été définie,
- créer une fabrique de clés secrètes avec la classe SecretKeyFactory pour l'algorithme 3DES nommé ici "DESede" pour DES Encryption-Decryption-Encryption,
- générer la clé secrète avec la méthode generateSecret(),
- créer un objet de type IvParameterSpec correspondant au vecteur d'initialisation,
- créer un objet de type Cipher en précisant la variante d'algorithme 3DES qui sera utilisé ici, à savoir "DESede/CBC/NoPadding", c'est-à-dire le DESede utilisant un découpage de messages en blocs (Cipher Block Chaining, ici de 64 bits) sans bits de bourrage (No padding).

```
// Prepare key
// Create an array to hold the key
byte[] baKey = { (byte)0x00, (byte)0x01, (byte)0x02, (byte)0x03, (byte)0x04, (byte)0x05, (byte)0x06,
(byte)0x07, (byte)0x00, (byte)0x01, (byte)0x02, (byte)0x03, (byte)0x04, (byte)0x05, (byte)0x06, (byte)0x07,
(byte)0x00, (byte)0x01, (byte)0x02, (byte)0x03, (byte)0x04, (byte)0x05, (byte)0x06, (byte)0x07, };

// Create a DESede key spec from the key
DESedeKeySpec keySpec = new DESedeKeySpec(baKey);

// Get the secret key factory for generating DESede keys
SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DESede");

// Generate a DESede SecretKey object
SecretKey theKey = keyFactory.generateSecret(keySpec);

// Create an initialization vector (necessary for CBC mode)
byte[] baiv = { (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x00,
(byte)0x00, };
IvParameterSpec iv = new IvParameterSpec(baiv);

Cipher cipher = Cipher.getInstance("DESede/CBC/NoPadding");
```

f) Ensuite pour effectuer une opération de chiffrement ou déchiffrement, on initialisera l'objet Cipher en précisant l'opération à effectuer. L'initialisation nécessite 3 paramètres :

- nature de l'opération : la constante Cipher.ENCRYPT_MODE si chiffrement ou Cipher.DECRYPT_MODE si déchiffrement.
- la clé secrète
- et le vecteur d'initialisation.

g) L'opération de chiffrement/déchiffrement proprement dite est réalisée grâce à la méthode doFinal().

2) Écrire l'applet Java Card afin qu'elle puisse traiter les commandes APDU envoyées par le terminal.

a) Il faut importer les packages suivants :

```
import javacard.security.DESEKey;
import javacard.security.KeyBuilder;
import javacardx.crypto.Cipher;
```

b) Il faut créer les différents objets nécessaires dans le constructeur de l'applet, soit :

- un objet DESKey via l'utilisation de KeyBuilder.buildKey() avec comme paramètres KeyBuilder.TYPE_DES, KeyBuilder.LENGTH_DES3_3KEY et false (car on ne souhaite pas que la clé soit chiffrée en mémoire).
- un objet Cipher qui implémente l'algorithme souhaité. Pour préciser qu'il s'agit d'un chiffrement/déchiffrement avec l'algorithme 3DES de type CBC No padding on utilisera la constante Cipher.ALG_DES_CBC_NOPAD et d'un paramètre mis à false (car on ne souhaite pas partager cet instance au travers du firewall).

c) Selon la commande APDU reçu :

- la donnée reçue pourra être utilisée comme clé via la méthode setKey(clé, offset)
- la donnée reçue sera chiffrée ou déchiffrée avant d'être renvoyée au terminal. La méthode init() de l'objet Cipher permet de fixer l'opération à effectuer :
 - chiffrement si le deuxième paramètre est égal à Cipher.MODE_ENCRYPT
 - déchiffrement si le deuxième paramètre est égal à Cipher.MODE_DECRYPT.

Le premier paramètre étant l'objet DESKey correspondant à la clé secrète.

On notera que le chiffrement/déchiffrement est effectivement réalisée à l'aide de la méthode doFinal() de l'objet Cipher.

Réalisation via RMI d'un porte monnaie électronique

Objectif : Apprendre à utiliser Java Card RMI pour écrire des applications client - serveur

Pour ce TP nous utiliserons la configuration JC2.2.1 et GP 2.1.1 pour le projet Java Card.

- 1) En vous inspirant du cours, écrire l'interface `Purse.java` matérialisant les services offerts par l'objet distant.
- 2) Ensuite écrire la classe `PurseImpl.java` qui matérialise l'objet distant. Cette classe doit implémenter l'interface `Purse.java` en complétant chaque méthode définie dans l'interface.
- 3) Écrire la classe `PurseApplet.java` comme défini dans le cours.

Après compilation et conversion, un fichier `.CAP` est généré. On installera l'applet Serveur sur une carte et on la sélectionnera avec le shell JCOP.

Après la sélection, le shell passe en mode « RMI ». On peut appeler les commandes `list`, `method`, `invoke`. On peut tester manuellement l'applet avec ces commandes. Voici des exemples (ou C7FA est la référence de l'objet) :

Remote Object:

```
JC RMI server applet: 00112233440101
JC RMI version: '0202'
INVOKE instruction byte: '38'
Remote ref. ID: 'C7FA'
Hash modifier: 'null'
Class: fr.cryptis.rmiserver.PurseImpl
```

```
method -m getBalance C7FA getBalance "short()"
invoke -r req_result ${getBalance}
/echo ${req_result}
method -m credit C7FA credit "(short)"
invoke ${credit} 200
```

- 4) Afin de tester l'applet `PurseApplet` installée sur la carte, écrire le programme client comme indiqué en cours. Pour communiquer il faudra implémenter la classe `PCSCAccessor.java` pour permettre la connexion au lecteur comme tout client Java. Il faudra penser à rendre accessible `javax/smartcardio/**` dans les « rules ». Le code est donné ci-après.

On pensera à ajouter le projet contenant l'application Java Card dans « Projects » pour pouvoir importer l'interface `Purse`. On pensera aussi à ajouter le package externe « `jcrmiclientframework.jar` » dans les « Libraries ». On trouve ce package dans le kit de développement d'Oracle (version 2.2.2 – ne pas utiliser une version précédente car il y a eu des changements importants ...)

- 5) Pour créer le stub du RemoteObjet, `PurseImpl`, on utilisera `rmic` en ligne de commande.

Mettez vous dans le bin du projet de l'application Carte et tapez :

```
chemin_vers_le_bin_de_votre_jdk\rmic -v1.2
-classpath .;chemin_vers\EclipseNewJCOP\plugins\com.ibm.blue
z.jcop.eclipse_3.1.2\apis\jc221\jc221.jar -d .
nom_complet_du_package.PurseImpl
```

Le stub n'est utile que si on utilise :

```
jcrmi.selectApplet( AID, JCRMICConnect.REF_WITH_CLASS_NAME );
```

```

package fr.cryptis.rmiclient;

import java.io.IOException;
import java.util.List;

import javax.smartcardio.Card;
import javax.smartcardio.CardChannel;
import javax.smartcardio.CardException;
import javax.smartcardio.CardTerminal;
import javax.smartcardio.CommandAPDU;
import javax.smartcardio.ResponseAPDU;
import javax.smartcardio.TerminalFactory;
import com.sun.javacard.clientlib.CardAccessor;

public class PCSCAccessor implements CardAccessor {
    CommandAPDU capdu;
    ResponseAPDU rapdu;
    Card card;
    CardChannel channel;

    public PCSCAccessor(){
        TerminalFactory factory = TerminalFactory.getDefault();
        try {
            List<CardTerminal> terminals = factory.terminals().list();
            System.out.println("Terminals : "+ terminals);

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

            // establish a connection with the card
            card = terminal.connect("");
            System.out.println("card: " + card);

            channel = card.getBasicChannel();
        } catch (CardException e) {
            System.out.println(e.getMessage());
        } catch (Exception e){
            e.printStackTrace();
            System.exit(1);
        } finally{
        }
    }

    public void closeCard() {
        try {
            card.disconnect(true);
        } catch (CardException e) {
            System.out.println(e.getMessage());
        } catch (Exception e){
            e.printStackTrace();
        }
    }

    public byte[] exchangeAPDU(byte[] arg0) throws IOException {
        try{
            byte[] rapdu = channel.transmit(new CommandAPDU(arg0)).getBytes();
            //extraction de SW a la fin de rapdu
            byte[] sw = new byte[2];
            sw[0] = rapdu [rapdu.length-2];
            sw[1] = rapdu [rapdu.length-1];
            //mise du SW en tete de retour et copie du reste de rapdu.
            byte []retour = new byte [rapdu.length];
            retour[0]= sw[0];
            retour[1]= sw[1];
            for (int i = 0; i<rapdu.length-2; i++){
                retour[i+2] = rapdu[i];
            }
            /*
             * elements pour debugger le protocole :

```

```

        *
        */
        System.out.print(">");
        System.out.println(ByteArrayToHexString(arg0));
        String s = ByteArrayToHexString(retour);

        System.out.println("<="+s);
        s = ByteArrayToHexString(sw);
        System.out.println("SW="+s);

        return retour;

    } catch (CardException e) {
        System.out.println(e.getMessage());
        return null;
    }
}

public String toString(){
    return "card accessor PCSC made in France";
}

/**
 * Utility method to convert a byte array to a hexadecimal string.
 *
 * @param bytes Bytes to convert
 * @return String, containing hexadecimal representation.
 */
public static String ByteArrayToHexString(byte[] bytes) {
    final char[] hexArray =
{'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
    char[] hexChars = new char[bytes.length * 2]; // Each byte has two hex characters
    nibbles
    int v;
    for (int j = 0; j < bytes.length; j++) {
        v = bytes[j] & 0xFF; // Cast bytes[j] to int, treating as unsigned value
        nibble
        hexChars[j * 2] = hexArray[v >>> 4]; // Select hex character from upper
        nibble
        hexChars[j * 2 + 1] = hexArray[v & 0x0F]; // Select hex character from lower
        nibble
    }
    return new String(hexChars);
}

/**
 * Utility method to convert a hexadecimal string to a byte string.
 *
 * <p>Behavior with input strings containing non-hexadecimal characters is undefined.
 *
 * @param s String containing hexadecimal characters to convert
 * @return Byte array generated from input
 * @throws java.lang.IllegalArgumentException if input length is incorrect
 */
public static byte[] HexStringToByteArray(String s) throws IllegalArgumentException {
    int len = s.length();
    if (len % 2 == 1) {
        throw new IllegalArgumentException("Hex string must have even number of
characters");
    }
    byte[] data = new byte[len / 2]; // Allocate 1 byte per 2 hex characters
    for (int i = 0; i < len; i += 2) {
        // Convert each character into a integer (base-16), then bit-shift into place
        data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4)
+ Character.digit(s.charAt(i+1), 16));
    }
    return data;
}
}

```