



Travaux pratiques

Java Card

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

TP n°1 – Applet et Client simples utilisant les APDU Etendu

TP n°3 – Porte-monnaie électronique simple sécurisé via un PIN

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 et des JavaCard JC10M128C (pour le TP APDU étendu).

Pour charger dans la carte, on utilisera suivant les TP et les besoins soit Eclipse soit le kit de développement de JavaCardOS <https://javacardos.com/tools/>.

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`).

Applet et Client simples utilisant les APDU Étendu

Objectif : Comprendre le fonctionnement des APDU étendu en Java Card mais aussi du côté client selon les protocoles de transports T=0, T=1, ...

Dans ce TP nous allons implémenter une applet Java Card capable de recevoir des APDU étendus. Ensuite nous la simulerons avec un programme client utilisant javax.smartcardio et capable de transmettre ces APDU quel que soit le protocole de transport.

Le protocole sera très simple.

- RECEIVE_LONG_APDU recevra des données du client et les stockera dans un tableau.
- SEND_LONG_APDU renverra au client les données contenues dans le tableau.

RECEIVE_LONG_APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0x80	0x30	0x0	0x0	00xxxx	data	N/A
• The data field contains VERY LONG DATA						
Response APDU						
Optional data		Status word		Meaning of status word		
N/A		0x9000		Successful processing		

SEND_LONG_APDU command						
Command APDU						
CLA	INS	P1	P2	Lc	Data field	Le
0x80	0x32	0x0	0x0	N/A	N/A	00xxxx
Response APDU						
Optional data		Status word		Meaning of status word		
data		0x9000		Successful processing		

Dans un premier temps, une fois l'applet implémenter nous pouvons utiliser un client simple pour envoyer les données :

- en T=1 d'abord ;
- en T=0 ensuite (ici il faudra utiliser les ENVELOPE pour le RECEIVE_LONG_APDU – attention à la CLA !)

Une fois le principe maîtrisé, implémenter un client lisant les commandes APDU dans un fichier texte et les transmettant indépendamment du protocole de transport.

A noter qu'on aurait pu réaliser un chiffrement ou une signature des données dans la carte par exemple.

Pensez à regarder <http://book2s.com/java/src/package/sun/security/smartcardio/channelimpl.html> et à jouer avec :

```
System.setProperty("sun.security.smartcardio.t0GetResponse", "false"); // et true
System.setProperty("sun.security.smartcardio.t1GetResponse", "false"); // et true
```

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 maximale 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.