```
package application_misc;

/*
*Utilisation des Java Card (U)SIM pour la sécurisation des communications
et des données dans les applications sur téléphones mobiles

 Cette applet est chargée de recevoir une commande d'une midlet
qui permette de récupérer une information confidentielle
à transmettre par SMS. L'applet procède aux opérations de
chiffrement et de déchiffrement des informations sensibles
à communiquer.

Code de l'applet (listing 5)

serge.chaumette at labri.fr
jonathan.ouoba at labri.fr
*/

/*
* Imported packages
*/
import uicc.toolkit.*;
import uicc.access.*;
import javacard.framework.*;
import javacard.security.*;
import javacardx.crypto.*;


public class Applet_Misc extends javacard.framework.Applet implements
        ToolkitInterface, uicc.toolkit.ToolkitConstants {
    // Mandatory variables
    private ToolkitRegistry reg;
    final byte DCS_8_BIT_DATA = 0x04;
    private FileView uiccView;

    private byte[] confidentialInfo = {(byte) 0x61, (byte) 0x62, (byte) 0x63, (byte) 0x64,
            (byte) 0x31, (byte) 0x32, (byte) 0x33, (byte) 0x34 };

    private byte[] user3DESCipheringKeyValue = {
              (byte) 0xCA, (byte) 0xCA, (byte) 0xCA, (byte) 0xCA, (byte) 0xCA,
              (byte) 0xCA, (byte) 0xCA, (byte) 0xCA,
              (byte) 0x2D, (byte) 0x2D, (byte) 0x2D, (byte) 0x2D, (byte) 0x2D,
              (byte) 0x2D, (byte) 0x2D, (byte) 0x2D,
              (byte) 0xCA, (byte) 0xCA, (byte) 0xCA, (byte) 0xCA, (byte) 0xCA,
              (byte) 0xCA, (byte) 0xCA, (byte) 0xCA
          };

    private final static byte INS_CRYPT = (byte) 0xC1;
    private final static byte INS_DECRYPT = (byte) 0xC3;

    //Define 3DES key objects
    private static Key user3DESCipheringKey;

    // Define Cipher object
    private static Cipher cipher3DESEnc;
    private static Cipher cipher3DESDec;
```

```
    // buffer to retrieve apdu data
    private byte[] apduData;
    private byte[] apduDataEncrypted;

    private byte[] tempBuffer;

    private short nbData, nbPad=64, nbPadL=11;

    /**
     * Constructor of the applet
     */
    public Applet_Misc(byte[] bArray, short bOffset, byte bLength) {
        // Register this applet
        register(bArray, (short) (bOffset + 1), (byte) bArray[bOffset]);

        // Get the reference of the applet ToolkitRegistry object
        reg = ToolkitRegistrySystem.getEntry();

        tempBuffer = JCSystem.makeTransientByteArray((short) 155,JCSystem.CLEAR_ON_RESET);

        // Get a reference to a FileView object on the UICC file system
        uiccView = UICCSystem.getTheUICCView(JCSystem.CLEAR_ON_RESET);

        apduData = new byte[255];
        apduDataEncrypted = new byte[64];

        //Create and initialize the applet's crypto objects
        initCrypto();

    }

    /**
     * Method called by the JCRE at the installation of the applet
     * @param bArray the byte array containing the AID bytes
     * @param bOffset the start of AID bytes in bArray
     * @param bLength the length of the AID bytes in bArray
     */
    public static void install(byte[] bArray, short bOffset, byte bLength) {
        // Create the Java SIM toolkit applet
        Applet_Misc StkCommandsExampleApplet = new Applet_Misc(bArray, bOffset,
                bLength);
    }

    /**
     * Method called by the GSM Framework.
     */
    public Shareable getShareableInterfaceObject(AID clientAID, byte parameter) {

        if ((parameter == (byte) 0x01) && (clientAID == null)) {
            return ((Shareable) this);
        }
        return null;
    }

    /**
     * Method called by the SIM Toolkit Framework
     * @param event the byte representation of the event triggered
```

```
     */
    public void processToolkit(short event) {
        if (event == EVENT_PROACTIVE_HANDLER_AVAILABLE){
            reg.clearEvent(EVENT_PROACTIVE_HANDLER_AVAILABLE);
            sendSMS(apduDataEncrypted);
        }
    }

    /**
     * Method called by the JCRE, once selected
     * @param apdu the incoming APDU object
     */
    public void process(APDU apdu) {
        // ignore the applet select command dispached to the process
        if (selectingApplet()) {
            return;
        }

        byte[] apduBuffer = apdu.getBuffer();

        apdu.setIncomingAndReceive();

        nbData = (short) (apduBuffer[ISO7816.OFFSET_LC] & 0x00FF);

        //Util.arrayCopyNonAtomic ( apduBuffer, (short) ISO7816.OFFSET_CDATA, apduData,  (short) 0x00, nbData  ) ;

        switch ((byte)apduBuffer[ISO7816.OFFSET_INS]) {

            // apdu command containing encryption instruction
            case INS_CRYPT:
                nbData = (short) 8;

                Util.arrayCopyNonAtomic ( confidentialInfo, (short) 0x00, apduData,  (short)
0x00, nbData  ) ;

                padUserData(apduData,nbData,(short)8);
                nbData = (short)(nbData + 1);
                short inter = (short)(nbData%8);
                inter = (short)(8-inter);
                nbData = (short)(nbData+inter);

                // Cipher user data + signature starting from offset 2 in smsUserData[] (i.e., after the TP-UDL field)
                // and overwrite clear data with ciphered data
                cipher3DESEnc.doFinal(apduData, (short) 0x00, nbData, apduDataEncrypted, (
short) 0x00);

                //reg.setEvent(EVENT_PROACTIVE_HANDLER_AVAILABLE);

                apdu.setOutgoing();
                apdu.setOutgoingLength(nbData);
                apdu.sendBytesLong(apduDataEncrypted,(short)0x00,nbData);
                break;

            // apdu command containing decryption instruction
            case INS_DECRYPT:
                nbData = (short) (apduBuffer[ISO7816.OFFSET_LC] & 0x00FF);
```

```
                Util.arrayCopyNonAtomic ( apduBuffer,  ( short ) ISO7816.OFFSET_CDATA,
apduData,  ( short ) 0x00, nbData  ) ;

                cipher3DESDec.doFinal(apduData, (short) 0x00, nbData, apduDataEncrypted, (
short) 0x00);

                apdu.setOutgoing();
                apdu.setOutgoingLength(nbData);
                apdu.sendBytesLong(apduDataEncrypted,(short)0x00,nbData);

                break;

            default:
                // The INS code is not supported
                ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
                break;

        }


    }

    // 3DES objects initialization
    private void initCrypto() {

        // Create 3DES key objects
        user3DESCipheringKey = KeyBuilder.buildKey(KeyBuilder.TYPE_DES,
                                                   KeyBuilder.LENGTH_DES3_3KEY, false);

        // Initialize the secret3DESCipheringKey with a key value
        //((DESKey) (user3DESCipheringKey)).setKey(user3DESCipheringKeyValue, (short) 0);

        ((DESKey) (user3DESCipheringKey)).setKey(user3DESCipheringKeyValue,
                (short) 0);

        // Create the cipher3DES object
        cipher3DESEnc = Cipher.getInstance(Cipher.ALG_DES_CBC_NOPAD, false);
        cipher3DESDec = Cipher.getInstance(Cipher.ALG_DES_CBC_NOPAD, false);

        //Initialize cipher3DES object with the user's ciphering key
        cipher3DESEnc.init(user3DESCipheringKey, Cipher.MODE_ENCRYPT);
        cipher3DESDec.init(user3DESCipheringKey, Cipher.MODE_DECRYPT);

        return;

     }

    // padding data to fit the requirement for encryption and decryption
    private void padUserData (byte[] smsUserData, short len, short pad) {

        // Retrieve the length of the User Data in bytes
        short length = len;

        // Systematically add padding "marker" character
        appendUserData(smsUserData, (byte)0x40, length);
        length++;
        // If User Data length not a multiple of pad
```

```
        if ( (length % pad) != (byte)0 ) {
            // Pad with '0' until User Data length is a multiple of pad
            while ((length % pad) != 0) {
                appendUserData(smsUserData, (byte)0x00, length);
                length++;
            }
            return;
        }

        return;

     }

    private void appendUserData (byte[] smsUserData, byte inputData, short offset) {

        // Set offset in User Data for copying the data
        // Copy data into smsUserData[] at offset
        smsUserData[offset] = inputData;
        return;

    }

    private void sendSMS(byte[] smsUserData) {
        ProactiveHandler proHdlr = ProactiveHandlerSystem.getTheHandler();

        //proHdlr.init((byte)0x13, (byte)0x00, (byte)DEV_ID_NETWORK);
        //proHdlr.appendTLV(TAG_ALPHA_IDENTIFIER,mySmsByteBuffer,(short)0,(short)0);
        //proHdlr.appendTLV(TAG_SMS_TPDU,mySmsByteBuffer,(short)0,(short)mySmsByteBuffer.length);
        //proHdlr.send();

        proHdlr.init( (byte) 19, (byte) 0x01, DEV_ID_NETWORK);

        // Append optional "Alpha identifier" to the Send short message command
        //proHdlr.appendTLV(TAG_ALPHA_IDENTIFIER, strings,
                            //(short) ( (MSG_SENDING_SMS * STRING_RECORD_LENGTH) + (short) 1),
                            //(short) (strings[ (short) (MSG_SENDING_SMS * STRING_RECORD_LENGTH)]));

        // Define and append optional "Service center address" for the message (TON/NPI + number)
        Util.arrayFillNonAtomic(tempBuffer, (short) 0, (short) tempBuffer.length, (byte) 0x00
);
        tempBuffer[0] = (byte) 0x91;
        tempBuffer[1] = (byte) 0x21;
        tempBuffer[2] = (byte) 0x43;
        tempBuffer[3] = (byte) 0x65;
        tempBuffer[4] = (byte) 0x87;
        proHdlr.appendTLV(TAG_ADDRESS, tempBuffer, (short) 0, (short) 5);
        // Define and append SMS TPDU
        Util.arrayFillNonAtomic(tempBuffer, (short) 0, (short) tempBuffer.length, (byte) 0x00
);
        // TP-MTI
        tempBuffer[0] = (byte) 0x01;
        // TP-MR
        tempBuffer[1] = (byte) 0x00;
        // TP-DA length
        tempBuffer[2] = (byte) 0x04;
        // TP-DA
        tempBuffer[3] = (byte) 0x91;
```

```
        tempBuffer[4] = (byte) 0x34;
        tempBuffer[5] = (byte) 0x12;
        // TP-PID
        tempBuffer[6] = (byte) 0x41;
        // TP-DCS
        tempBuffer[7] = (byte) 0xF2;
        // TP-UDL
        tempBuffer[8] = (byte) 0x05;
        // TP-UD 'Hello' in 7-bit packed format
        /*tempBuffer[9] = (byte) 0xC8;
      tempBuffer[10] = (byte) 0x32;
      tempBuffer[11] = (byte) 0x9B;
      tempBuffer[12] = (byte) 0xFD;
      tempBuffer[13] = (byte) 0x06;*/
        Util.arrayCopyNonAtomic ( smsUserData,  (short)0x00, tempBuffer,  (short)0x09, (short
)smsUserData.length) ;
        //proHdlr.appendTLV(TAG_SMS_TPDU, tempBuffer, (short) 0, (short) 14);
        proHdlr.appendTLV( (byte) 11, tempBuffer, (short) 0, (short)(9+smsUserData.length));

        // Send the command to the mobile
        proHdlr.send();
        return;

    }

}
```