Security and Trusted Hardware Applications
Week #5 Tutorial

Pedro Antunes - up201507254

May 30, 2022

Instructor: Bernardo Portela

1

If the processing of a JC applet requires guarantees of atomicity and involves multiple values or data structures, we must protect this processing by creating a transaction. However, when using transactions we have to look at the memory capacity of the respective card and take precautions that the data stored during a transaction does not exceed the capacity of the card. If this happens, it leads to abrupt failures in the processing of the applet's functions.

To develop these precautions, the JCs provide a data structure called JCSystem that provides two important methods for us to control the memory capacity of the card. These methods are:

- **getMaxCommitCapacity()** - Returns the total number of bytes in the commit buffer, used by an transaction.
- **getUnusedCommitCapacity()** - Returns the number of unused bytes in the commit buffer.

2

To extend the class Applet, a JC applet must develop four methods which are:

- **install()** - It creates an applet instance, creates and initialize object that the applet needs. After this, register the AID (uniquely identified of an applet) by calling register() method;
- **select()** - It checks if selection conditions are met by checking AID of the applet. If this check fails, JCRE(Java Card Runtime Environment) returns status work 0x6999;
- **process()** - This method is the main method for applet functionalities. It work with APDU data structure and It must be overridden to describe what to do with APDUs commands.
- **deselect()** - It terminates the execution of an JC applet by deselect this applet on JCRE with AID. It performs necessary cleanup operations like resetting security control flow conditions before another selection.

3

```
public void process(APDU apdu){

                    (...)

   // Sets the applet to send mode
   // Expected response length: le
   short le = apdu.setOutgoing();

   // Inform the host that we will send 10 bytes
   apdu.setOutgoingLength((short) 10);

                    (...)

   // Send data
   apdu.sendBytes((short) 0, (short) 10);
}
```

Figure 1. Process function on "send-only" mode showing half-duplex behavior.

In Figure 1, we can see a process implementation of "send-only" mode. The APDU data structure has special methods to implementing this mode like setOutgoing() that is responsible to retrieve the response length that a command apdu want to receive. Further, we can inform the host with response length that we will send by calling setOutgoingLength() method. The sendBytes() method will send bytes data back to the host.

4

Both instructions are retrieving the value referer to INS (Instruction code) on APDU buffer. The variable ISO7816.OFFSET_INS is a constant defined by the standard ISO 7816 and his value is 1.

According to good programming practices, we should follow standard definitions and so we should use ISO7816.OFFSET_INS when we want to retieve the value referer to INS on APDU buffer. This is the case we should follow because the APDU buffer indexes on an APDU command can change through past the years. Sometimes, the tecnhologies that we use need some approach changes driven by tool fields that are no longer used or by inserting new tool fields.

By using standard constants, index values can be changed and applet-related code remains up to date.

5

An applet implementing algorithm SHA-2 to compute a message digest it does not work in properly in all Java Cards because there are JCRE on JC that doesn't support such message digest algorithm. When we develop an applet that implements crypto algorithm we must know if the JC supports the algorithms choosen ones.

To circumvent this problem the programmer must know which algorithms, key types and key length are supported by JC. The JCA (JCRE Cryptography Architecture) makes extensive use of factory methods by naming convention to specify crypto algorithms like was shown in this exercise. JCRE, if it have support to do it, it can use this and may choose to implement some of the selected factory algorithms.

But we should keep in mind that when we are downgrading the security level of the cryptographic algorithms used, we can be at risk of having the information inside a JC vulnerable to weak encryption attacks. Therefore, considering getting another type of card/reader to improve the cryptographic algorithms used in a JC applet is a good solution.