

jpass-smartcard

Valentyn Kuznietsov, Petr Skyva, Zuzana Matějková

OVERVIEW

The project is the fork of jpass (<https://github.com/gaborbata/jpass>) .

Repository of the project: <https://github.com/ValentinKuznetsov/jpass-smartcard>.

Features

1. Java card with applet instead of using master password
2. Secure channel between card and app
3. Bulk encryption for encrypting and decrypting database of password

SPECIFICATIONS

1. User can export the database by providing his smart card, instead of typing a password.
2. Upon the launch, the application asks for a smart card with a key, which will be used to decrypt the database.
3. Smart card is locked by a factory-set PIN (use 1234)
4. The secure channel is established between the card and the desktop side, such that all data is encrypted.
 - a. AES with zero IV is used
 - b. Unfortunately, the key which is used for the secure channel is hardcoded both on the desktop side and the applet side; we almost managed to have Diffie-Hellman working, but, unfortunately, due to an issue of RSA-decrypting on the card side we, having spent many hours, failed to do so.

BUILDING

No extra-actions are required. Just pull the code and build.

For testing it on the real card, the source code should be tweaked: just replace true to false in the following code:

```
22. this.card = new JavaCard(true);
```

CardsManager.java

PROJECT STRUCTURE (important packages)

1. Applet package
 - a. JPassApplet - the applet which is on the card and used for bulk encryption.
 - b. SecureChannel - the additional layer between the applet process() method and APDU's processing. Basically, it wraps (encrypts) all outgoing packets and unwraps (decrypts) all incoming packets.
2. DiffieHellman
 - a. Has Diffie-Hellman implementation. Unfortunately, is not finished because of some issue with modulo operation. (Description of this issue is omitted in this report)
3. jpass.data
 - a. DocumentHelper.java
 - i. basically, the new additional layer of enc/dec the database with a card was added.
4. Changes to the UI (not that important)
5. jpass.smartcard
 - a. APDUFactory - used for creating APDU requests.
 - b. CardsManager - singleton, has an instance of JavaCard. Provides API to work with the card. Basically, this API is a wrapper around JavaCard.
 - c. JavaCard - the class which performs APDU requests to the card and processing the response.

Secure Channel

Secure channel is way of transferring data that is resistant to overhearing and tampering. We implemented secure channel based on symmetric cryptography.

For key negotiation, we tried to implement Diffie-Hellman, but there were issues with deriving shared secret between JavaCard and desktop application. Thus, unfortunately, now the secure channel is based on hardcoded keys on both sides.

The implementation is in SecureChannel.java. There are two pairs of functions, wrap and unwrap (two - for applet side, two - for desktop side). These functions are used to wrap(encrypt) APDUs before sending to the card, unwrap(decrypt) on the card to process, wrap again the response, and unwrap the response on the desktop side.

SecureChannel.java

JavaCard part

```
public short wrapResponseAPDU(short offset, short length)
    - Wrap (encrypt) response APDU
    - Offset where encryption starts
    - Length how many bytes to encrypt
    - Response codes are not encrypted (such as 90 00), only
      data
public void unwrapCommandAPDU()
    - Unwrap (decrypt) incoming APDU
```

Desktop part

```
public byte[] encryptSend(byte[] buffer)
    - Wrap (encrypt) dispatched APDU
    - Buffer data part of dispatched APDU
    - Only data part is encrypted, but not the instructions.
public byte[] decryptResponse(ResponseAPDU response)
    - Unwrap (decrypt) received response APDU from card
```

For crypto operations the JavaCard library `javacardx.crypto.Cipher` is used.
Hard coded private key on JavaCard is set in `JPassApplet.java`.

On desktop side private key is initiated in `JavaCard.java` in same way as in JavaCard in constructor.

Diffie-Hellman

The implementation of the Diffie-Hellman protocol failed, because the generation of the shared secret failed. Most likely, there was some issue with saving output of RSA (which was used for modulo operations), such as signed values.

Desktop side

The original JPass applet is easy to use - It is quite intuitive. User creates entries in the database by providing title, name, password, url and notes. He can export this database in the encrypted form using the card; he can import the database as well (after validating PIN).

Firstly it is checked whether the card is connected, and if yes, user is challenged to enter PIN. If the PIN is correct, he can choose name and location of the file to which he would like to store encrypted database. The same flow applies for importing.

The instance of the card is encapsulated by the CardsManager.java class which is a singleton and provides all necessary API to work with the card.

Applet side

The following instructions are supported:

```
INS_VERIFYPIN                = (byte) 0x11;
Accepts 4-byte pin and verifies it.

INS_SETPIN                    = (byte) 0x12;
Accepts a pin and sets it.

INS_ISPINUNLOCKED            = (byte) 0x13;
Tells whether card is already unlocked with a PIN or not.

INS_CBC_BULK_INIT            = (byte) 0x50;
Inits AES-CBC-PKCS5 bulk encryption. Accepts 16-byte IV.

INS_CBC_BULK_PROCESS         = (byte) 0x51;
Processes a 16-byte chunk. Accepts 16(or less)-byte chunk, and the
mode of operation (01 for encryption, 02 for decryption)

INS_CBC_BULK_FINISH         = (byte) 0x52;
Finalizes the BULK decryption. Returns padding block, if needed
(encryption), or unpadded last block (if decryption)

INS_DH_INIT                  = (byte) 0x60;
Inits the DH protocol with some private key, P and G.

INS_DH_GET_Y_PART            = (byte) 0x61;
Obtains first part of Y for DH protocol.

INS_DH_GET_Y_PART2          = (byte) 0x65;
Obtains second part of Y for DH protocol.

INS_DH_SET                   = (byte) 0x62;
Sets Y for the opposite party.

INS_DH_FINAL                 = (byte) 0x63;
Finalizes the DH protocol (sides should have shared secret at that
point)
```

Please also see APDUFactory.java in jpass.smartcard package for better understanding of the APDUs.

Feel free to contact any of us if you need help:

@valentykuznietsov (telegram)

Valentyn.Kuznietsov - Skype, and I will share the contact with another team members, if you'd like.

References and reused source code:

PKCS5Padding.java

Initial implementation was taken here and ported to the JavaCard Platform:

<http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/com/sun/crypto/provider/PKCS5Padding.java#PKCS5Padding.padWithLen%28byte%5B%5D%2Cint%2Cint%29>

(In particular, conversions to (short) were added, and some changes in the API)

SecureChannel.java

Initial implementation was taken here:

<https://github.com/jderuiter/javacard-openpgpcard/blob/master/src/openpgpcard/OpenPGPSecureMessaging.java>

(But, more than 90% of the source code was removed completely or reworked)

Diffie-Hellman.java

Initial implementation was taken here:

<https://github.com/ASKGLab/DHApplet/blob/master/src/dhapplet/DH.java>

TerminalManager.java

Was initially CardManager.java by Petr Svenda, provided during the semester.