Term Project Report Phase - II

Cryptsetup-Javacard

Team Members: Hitesh Lilhare (476356), Urvek Chandravadan Shah (476355)

1. Brief Description of the project:

Cryptsetup: Cryptsetup is utility used to conveniently setup disk encryption based on DMCrypt kernel module.

Cryptsetup-javacard isa “A JavaCard key manager for Cryptsetup”. Developers have created shell scripts to interact with applet as well as cryptsetup functionality which supports

- New encrypted partition
- Deletion of encrypted partition
- Open encrypted partition and mount it

2. Applet supports following Functionality:

1. Extract public-key of card
2. Establish DH session key for each transaction
3. Authenticate user using master password
4. Change master password
5. Partition key generation
6. Store partition-key to KeyStore in applet
7. Load partition-key from KeyStore
8. Delete partition-key from KeyStore
9. Reset session

The figure below shows the corresponding functional APDUs. There are three main APDU supported by applet and other seven commands supported with INS_COMMAND APDU with a in the applet.

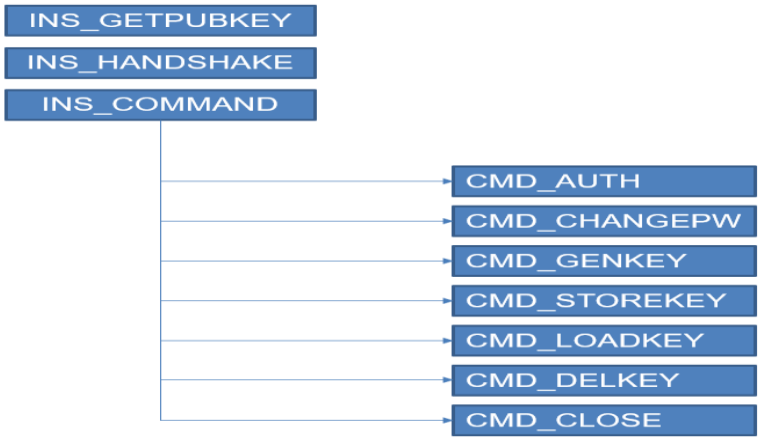

Fig.1

3. APDU and its format:

<table>
<tr><td>INS_GETPUBKEY</td><td>– Request APDU
<table><tr><td>CLA</td><td>INS_GETPUBKEY(0x50)</td><td>0</td><td>0</td><td>Lc</td><td>Le (1024)</td></tr></table>
– Reponses APDU
<table><tr><td>Modulus 1st Byte</td><td>Modulus 2nd Byte</td><td>Exponent 1st Byte</td><td>Exponent 2nd Byte</td><td>SW1</td><td>SW2</td></tr></table></td></tr>
<tr><td>INS-HANDSHAKE</td><td>– Request APDU
<table><tr><td>CLA</td><td>INS_HANDSAHK(0x51)</td><td>0</td><td>0</td><td>Lc</td><td>Pubkey length 1</td><td>Pubkey length 2</td><td>Pub key (Q Value)</td><td>Le (1024)</td></tr></table>
– Response APDU
<table><tr><td>Signlenght 1</td><td>Signlength 2</td><td>Signature</td><td>PubKeyLen 1</td><td>PubKeyLen2</td><td>PubKey(Q Value)</td><td>DHpubkey length 1</td><td>DHpubKey length 2</td><td>DH Pub Key (W Value)</td></tr></table></td></tr>
<tr><td>INS_COMMAND</td><td>– Request APDU
<table><tr><td>CLA</td><td>INS_COMMAND(0x52)</td><td>0</td><td>0</td><td>Lc</td><td>HMAC-32 byte</td><td>Seq num-2byte</td><td>Iv – 16 byte</td><td>AES_cbc Enc(data)</td><td>Le (1024)</td></tr></table>
– Response APDU
<table><tr><td>HMAC-32 byte</td><td>Seq num+1 (2byte)</td><td>Iv – 16 byte</td><td>AES_cbc Enc(data)</td><td>Sw1</td><td>Sw2</td></tr></table></td></tr>
<tr><td>CMD_AUTH</td><td>– Request Data
<table><tr><td>CMD_AUTH - 0x00</td><td>Master password length 1</td><td>Master password length 2</td><td>UTF-8 Encoded Master Password</td></tr></table>
– Response Data
<table><tr><td>0 (2 byte)</td></tr></table></td></tr>
<tr><td>CMD_GENKEY</td><td>– Request Data
<table><tr><td>CMD_GENKEY- 0x02</td><td>data Length 1</td><td>data Length 2</td><td>Data (key size)</td></tr></table>
– Response Data
<table><tr><td>key size -02 byte</td><td>Key (key size)</td></tr></table></td></tr>
<tr><td>CMD_STOREKEY</td><td>– Request Data
<table><tr><td>CMD_STORE KEY 0x03</td><td>Data length 2</td><td>Data Length 2</td><td>UUID – 40 byte</td><td>Key length – 1 byte</td><td>Key</td></tr></table>
– Response Data
<table><tr><td>0 (2 byte)</td></tr></table></td></tr>
<tr><td>CMD_LOADKEY</td><td>– Request Data
<table><tr><td>CMD_LOADKEY 0x04</td><td>Data length 2</td><td>Data Length 2</td><td>UUID – 40 byte</td></tr></table>
– Response Data
<table><tr><td>key length</td><td>Key</td></tr></table></td></tr>
<tr><td>CMD_DELKEY</td><td>– Request Data
<table><tr><td>CMD_DELKEY 0x05</td><td>Data length 2</td><td>Data Length 2</td><td>UUID – 40 byte</td></tr></table>
– Response Data
<table><tr><td>0 (2 byte)</td></tr></table></td></tr>
<tr><td>CMD_CHANGEPW</td><td>– Request Data
<table><tr><td>CMD_CHANGEPW - 0x01</td><td>New password length 1</td><td>new password length 2</td><td>UTF-8 Encoded new Password</td></tr></table></td></tr>
<tr><td></td><td>– Response Data
<table><tr><td>0 (2 byte)</td></tr></table></td></tr>
</table>

Table 1: APDU Commands

4. **Sensitive values that are protected:**

- Master password
- Partition keys
- UUID – related to LINUX Partition

5. **Cryptographic Algorithms and Protocol used**

– Signing:

- Key Pair Generation: ALG_RSA_CRT of bit length 1024
- Signature: RSA_SHA_PKCS1

– DH Key Pair:

- Key Pair Generation: ALG_EC_FP of bit length 192

– Session Key Agreement:

- ALG_EC_SVDP_DHC: Elliptic curve secret value derivation primitive, Diffie-Hellman version with cofactor multiplication and compatibility mode, as per [IEEE P1363].
- Note: This algorithm computes the SHA-1 message digest of the output of the derivation primitive to yield a **20-byte** result.

– Cipher Key:

- ALG_AES_BLOCK_128_CBC_NOPAD

– HMAC:

- Message Digest ALG_SHA_256 with digest block size of 64 bytes.

– Random Data:

- ALG_SECURE_RANDOM

6. **Security review of code:**

The code quality of cryptsetup-javacard found very good. The code is adhering to standard java card development guidelines, which are mentioned in a positive comment section. During the security code review, the bugs discovered are mentioned as negative comments in below section.

Positive Comments	
`short dataLen = apdu.setIncomingAndReceive();` `byte[] apduBuffer = apdu.getBuffer();`	Applets should not read from the APDU buffer before calling **setIncomingAndReceive() or receiveBytes()** to transfer the incoming data into the APDU buffer

`if (apduBuffer[ISO7816.OFFSET_CLA] != CLA_KEYSTORAGEAPPLET) {`
`  ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);`
`}`
	Verify that if the applet can accept this APDU message

`masterPassword = new OwnerPIN(MAX_PW_TRIES, MAX_PW_LEN);`
`if (bLength == 0) {`
`  ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);`
`} else {`
`  /* set master password from install data: */`
`  masterPassword.update(bArray, bOffset, bLength);`
`}`
	PINs and Keys are handled securely. Accessed via secure APIs

`auxBuffer = JCSystem.makeTransientByteArray(AUX_BUFFER_SIZE, JCSystem.CLEAR_ON_DESELECT);`
`cipherKey = (AESKey)KeyBuilder.buildKey(KeyBuilder.TYPE_AES_TRANSIENT_DESELECT,`
`  KeyBuilder.LENGTH_AES_256, false);`
	Sensitive data has been initialized at beginning and clear at the end of session.

`cipherKey = (AESKey)KeyBuilder.buildKey(KeyBuilder.TYPE_AES_TRANSIENT_DESELECT,` `  KeyBuilder.LENGTH_AES_256, false);`	Sensitive data stored in transient memory

`public static final byte[] AID = new byte[] {`
`  (byte)0x4a, (byte)0x43, (byte)0x4b, (byte)0x65, (byte)0x79, (byte)0x53,`
`  (byte)0x74, (byte)0x6f, (byte)0x72, (byte)0x61, (byte)0x67, (byte)0x65`
`};`
	All initialized buffers declared as static

`public static final byte INS_GETPUBKEY = (byte)0x50;`
`public static final byte INS_HANDSHAKE = (byte)0x51;`
`public static final byte INS_COMMAND  = (byte)0x52;`
	Declare your constants as static final

`private short dhHandshake(APDU apdu) {`
`  byte[]  apdubuf = apdu.getBuffer();`
`  short  dataLen = apdu.getIncomingLength();`
	Temporary entry points should be stored in local variables (not in class, instance variable etc.)

Negative Comment	
`private static short readShort(byte[] buf, short offset) { /* read high byte: */ short res = (short)(buf[(short)(offset + (short)1)] & (short)0xFF); res <<= (short)8; /* read low byte: */ res |= (short)(buf[offset] & (short)0xFF); return res; }`	Util.getShort()
`private static void writeShort(byte[] buf, short offset, short value) { buf[offset] = (byte)(value & (short)0xFF); ++offset; value >>= (short)8; buf[offset] = (byte)(value & (short)0xFF); }`	Util.setShort()
`STATE_IDLE = (short)0; STATE_KEY_ESTABILISHED = (short)1; STATE_AUTHENTICATED = (short)2;`	Not safe against fault induction attacks.
• RSA 1024 • SHA 1	Weak crypto is used

Table 2: Security Code Review Results

7. Attacker Model: Fault induction attack:

1. When processor increments the number of password tries counter, attacker can induce a fault which may prevent the processor to increment the counter resulting in more than 5 number of password tries which leads to the brute force attack on password.
2. As Hamming Distance between applet state variable is 1 bit only which makes it vulnerable to fault induction attack.

8. State Diagram of applet:

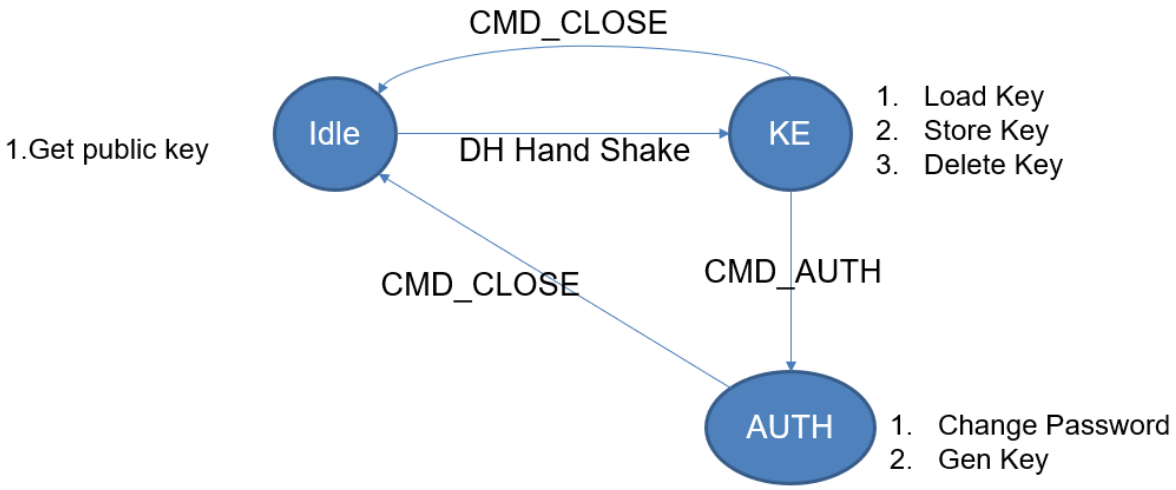

Fig. 2