MTAT.07.017
Applied Cryptography

Smart Cards (JavaCard)

University of Tartu

Spring 2018
Smart Card Security Model

Parties involved in a smart card–based system:

- Cardholder
- Data owner
- Terminal owner
- Card issuer
- Card (software)manufacturer

Smart card threat models:

- attacks by the terminal against the cardholder
- attacks by the cardholder against the terminal
- attacks by the cardholder against the data owner
- attacks by the cardholder against the issuer
- attacks by the terminal owner against the issuer
- attacks by the issuer against the cardholder
- attacks by the (software)manufacturer against the data owner

Estonian ID card

- Used for:
  - Protected RSA private key storage
  - Perform on-card signing/decryption
  - Authorize cryptographic operations (using PIN)
- Cardholder / Data owner / Terminal / Card issuer / Card manufacturer / Software manufacturer
- Attacks:
  - by the terminal against the cardholder
  - by the cardholder against the terminal owner
  - by the cardholder against the data owner
  - by the issuer against the cardholder
  - by the (software)manufacturer against the data owner
Mobile phones (SIM card)

- **Used for:**
  - Store phone book contacts and SMS messages
  - Store settings (operator information)
  - Store 128-bit symmetric subscriber authentication key
  - Perform RUN GSM ALGORITHM
  - Authorize operations (using PIN)
  - Mobile-ID

- **Attacks:**
  - by the cardholder against the data owner
  - by the terminal owner against the issuer
  - by the issuer against the cardholder
EMV stands for Europay, MasterCard and Visa

- Used for:
  - Store symmetric MAC key
  - Authentication of credit card transactions (using PIN)

- Attacks:
  - by the terminal against the cardholder
  - by the terminal owner against the issuer
  - by the issuer against the data owner
Pay TV

• Used for:
  • TV signal decryption
  • Store channel filters

• Attacks:
  • by the cardholder against the data owner/issuer
  • by the terminal owner against the issuer
Tachograph

- Used for:
  - Record driving activities

- Attacks:
  - by the cardholder against the data owner/issuer
  - by the terminal owner against the issuer
Attacks Against Smart Cards

- Side channel attacks:
  - Timing analysis
  - Power analysis
  - EM signal analysis

- Introducing glitches, faults (voltage, clock rate)
  - Induce bit errors

- Physical attacks:
  - Chemical etching
  - Chip re-wiring
  - Addition of a track
  - Cutting of a track

- Countermeasures
  - Metal layers
  - Onboard sensors (temp, light, frequency)
  - ...
Java Card

• Card capable of running code written in Java
• Stripped down version of Java
  • Data types: boolean, byte, short
  • Not supported: char, String, float, int
  • One dimensional arrays
  • No threads
• Rich cryptography API available
  • Employs cryptographic coprocessor
  • Algorithm support depends on card ([https://www.fi.muni.cz/~xsvenda/jcalgtest/table.html](https://www.fi.muni.cz/~xsvenda/jcalgtest/table.html))
  • Side-channel protection guaranteed only for crypto API calls
• Java .class file has to be converted to .cap file

• Estonian ID cards issued since 2011 are Java Cards
GlobalPlatform

- Standard for applet management on Java Cards
- Multiple applets can be installed
  - Applet is selected using Application Identifier (SELECT AID)
  - Applet can be set as default applet (selected by default)
  - Applets are isolated (with exceptions – Shareable Interface)
- Applet can be deleted (usually), but never downloaded
- Security Domain (SD)
  - Every applet belongs to a SD
  - Card Issuer Security Domain (ISD)
  - Supplementary Security Domains (SSDs)
  - Secure Channel Protocol for communication with SD
Java Card Applet

$ cat TestApplet.java
package appcrypto;

import javacard.framework.*;
import javacard.security.*;
import javacardx.crypto.*;

public class TestApplet extends Applet {
    RandomData rnd;

    public static void install(byte[] ba, short ofs, byte len) {
        (new TestApplet()).register();
    }

    public void process(APDU apdu) {
        byte[] buf = apdu.getBuffer(); // contains first 5 APDU bytes

        switch (buf[ISO7816.OFFSET_INS]) {
            case (byte)0x00:
                if (buf[ISO7816.OFFSET_LC] != (byte)1) {
                    ISOException.throwIt(ISO7816.SW_DATA_INVALID);
                }
                apdu.setIncomingAndReceive(); // read APDU data bytes
                short len = (short)(buf[ISO7816.OFFSET_CDATA] & (short)0xff); // get rid of sign
                rnd = RandomData.getInstance(RandomData.ALG_SECURE_RANDOM);
                rnd.generateData(buf, (short)0, len);
                apdu.setOutgoingAndSend((short)0, len); // return response data
                return;
            }
            ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
        }
    }
}
Converting to CAP

$ sudo apt-get install opensc openjdk-8-jdk ant
$ wget https://github.com/martinpaljak/ant-javacard/releases/download/v1.6/ant-javacard.jar
$ git clone https://github.com/martinpaljak/oracle_javacard_sdks

$ cat build.xml
<?xml version="1.0" encoding="UTF-8"?>
<project default="applet" basedir=".">

  <target name="jcpro">
    <taskdef name="javacard" classname="pro.javacard.ant.JavaCard" classpath="ant-javacard.jar"/>
  </target>

  <target name="applet" depends="jcpro">
    <javacard>
      <cap jckit="oracle_javacard_sdks/jc222_kit/" aid="0102030405" output="applet.cap" sources=".">
        <applet class="appcrypto.TestApplet" aid="0102030405060708"/>
      </cap>
    </javacard>
  </target>
</project>

$ ant
applet:
  [cap] INFO: using JavaCard v2.2.2 SDK in oracle_javacard_sdks/jc222_kit/
  [cap] Setting package name to appcrypto
  [cap] Building CAP with 1 applet from package appcrypto
  [cap] appcrypto.TestApplet 0102030405060708
  [compile] Compiling 1 source file to /tmp/jccpro2076737826668178311
  [cap] CAP saved to applet.cap
BUILD SUCCESSFUL
Total time: 2 seconds
Installing CAP file

$ wget https://github.com/martinpaljak/GlobalPlatformPro/releases/download/v0.3.5/gp.jar

$ java -jar gp.jar --install applet.cap --default
CAP loaded

$ java -jar gp.jar --list
[..]
AID: 0102030405060708 (........)
    App SELECTABLE: Default selected
AID: 0102030405 (.....)
    ExM LOADED: (none)
    0102030405060708 (........)

$ opensc-tool -s 00:00:00:00:01:05:00
Received (SW1=0x90, SW2=0x00):
A2 3C BA 73 A2 .<.s.

$ opensc-tool -s 00:00:00:00:01:a0:00
Received (SW1=0x90, SW2=0x00):
3F 35 13 B2 7D F0 FB 3E D7 CC 6F 3E 75 38 1C 00 ?5..}..<..o>u8..
3F 35 13 B2 7D F0 FB 3E D7 CC 6F 3E 75 38 1C 00 ?5..}..<..o>u8..
A8 35 13 B2 7D F0 FB 3E D7 CC 6F 3E 75 38 1C 00 .5..}..<..o>u8..
A8 71 13 B2 7D F0 FB 3E D7 CC 6F 3E 75 38 1C 00 .q..}..<..o>u8..
A8 71 71 F1 B2 7D F0 FB 3E D7 CC 6F 3E 75 38 1C 00 .q..}..<..o>u8..
A8 71 F1 83 65 F0 FB 3E D7 CC 6F 3E 75 38 1C 00 .q..e..}..<..o>u8..
A8 71 F1 83 65 B4 FB 3E D7 CC 6F 3E 75 38 1C 00 .q..e..}..<..o>u8..
A8 71 F1 83 65 B4 70 3E D7 CC 6F 3E 75 38 1C 00 .q..e.p>..<o>u8..
A8 71 F1 83 65 B4 70 76 81 CC 6F 3E 75 38 1C 00 .q..e.pv.o>u8..
A8 71 F1 83 65 B4 70 76 81 66 6F 3E 75 38 1C 00 .q..e.pv.fo>u8..

$ java -jar gp.jar --deletedeps --delete 0102030405
Blank Java Card (Feitian)

- You are given a blank Java Card (16 cards available)
  - Chip: ST31
  - EEPROM: 50K
  - RAM: 5K
  - Java Card 2.2.2
  - GlobalPlatform 2.1.1
  - DES/3DES/AES128
  - MD5/SHA1/SHA224/SHA256
  - RSA-2048 (on-card generation)
  - ECC-256 (on-card generation)
  - Contactless Interface
  - Garbage collector
  - No security certifications

**Warning:** On-card RNG flawed!
Blank Java Card (Infineon)

- You are given a blank Java Card (9 cards available)
  - Chip: Infineon SLE78
  - EEPROM: 150K
  - RAM: ?
  - Java Card 3.0.4
  - GlobalPlatform 2.2.1
  - DES/3DES/AES256
  - MD5/SHA1/SHA256/SHA512
  - RSA-2048 (on-card generation)
  - ECC-521 (on-card generation)
  - CC EAL5+ high certification

*Warning: Infineon RSALib flaw!*
Task

Write Java Card applet that performs on-card RSA 1024/2048-bit key generation and decryption.

$ python test.py --keysize 2048
[+] Selected reader: Gemalto PC Twin Reader 00 00
[+] Feitian FT-Java/D11CR
[+] Generating 2048-bit RSA key...
[+] Key generated in 3.90829 seconds!
[+] Retrieving public key...
[+] n=182755735622914188302216599628741478372766442369552289574366115181
[+] e=65537
[?] Enter message to encrypt: Hello world!
[+] Encrypted message: 48d39a52a0650b2c506c52343beaeeb53976b6a3f44522f8
[+] Sending ciphertext to card...
[+] Message decrypted in 0.859465 seconds!
[+] Decrypted message: Hello world!

Commit TestApplet.java to your repository.

The students who will not get the card, will be able to get it next week and will have one week extension for this homework.
Hints

• Find out the communication protocol from test.py
• API calls to use:

```java
keypair = new KeyPair(KeyPair.ALG_RSA, KeyBuilder.LENGTH_RSA_*);
keypair.genKeyPair();

pub = (RSAPublicKey) keypair.getPublic();
pub.getExponent(byte[] buffer, short offset);
pub.getModulus(byte[] buffer, short offset);

rsa = Cipher.getInstance(Cipher.ALG_RSA_PKCS1, false);
rsa.init(keypair.getPrivate(), Cipher.MODE_DECRYPT);
rsa.doFinal(byte[] inBuff, short inOffset, short inLength,
            byte[] outBuff, short outOffset);
```

• Size limit for data APDU body is 255 bytes
  • The first two bytes of ciphertext are embedded in P1 and P2
• Make the ciphertext continuous using:

```java
Util.arrayCopyNonAtomic(byte[] src, short srcOff, byte[] dest,
                         short destOff, short length);
```

• Make sure the keypair is generated only once
• Java has signed types – cast byte to short using 0xff mask
• Debugging possible only via the data or SW returned!
Java Card Development under Eclipse

- Create new Java Project. “File — New — Project... — Java Project — Project name: appcrypto”.
- Right-click on the project “New — Class — Name: TestApplet, Package: appcrypto”.
Right-click on your project “Build Path — Configure Build Path... — Libraries — Add External JARs” and add oracle_javacard_sdks/jc222_kit/lib/api.jar. This will enable Java Card code validation and completion.