Secure Element Development

Josef Langer, Andreas Oyrer

4th Sept. 2009, Oulu Developers Summit
Agenda Part I: 11:45 – 12:30

- What is a Secure Element?
- Secure Elements & SmartCards
- Components of a Secure Elements
- Tools for Developing Secure Elements
- Standards
Agenda Part II 1:30 – 3:00

- Secure Element in Details
- Presentation Development Tools
- Communicating with the Secure Element:
  - Mobilephone: J2ME
  - PC: Reader-Writer mode via PC/SC
- Difficulties in Programming Secure Elements
- Benefits and Drawbacks with Secure Element Programming
Abbreviations and Definitions

- J2ME: Java Platform Micro Edition
- JSR: Java Specification Requests
- Midlet: Java-applications running on a mobile phone
- Applet: JavaCard-Application running on a SmartCard
- APDU: Application Protocol Data Unit
- SCWS: Smart Card Web Server
Abbreviations and Definitions

- SWP: Single Wire Protocol
- USB: Universal Serial Bus
- PC/SC: Smartcard Standard for PC
- API: Application Programming Interface
- SAT: SIM Application Toolkit
What is a Secure Element?

- Secure storage in your NFC device
- Current Secure Element Implementations
  - Embedded in Mobile Phone
  - SIM Based
  - Removeable Secure Element (SD Card)
Motivation for Secure Elements

- Necessary for several Applications:
  - Payment
  - Ticketing
  - Government
  - Secure Authentication
  - ....

- Trusted
- Secure
Secure Element communication in NFC Devices
NFC Device Standards

- OMA
- Java Community Process
- ETSI
- NFC forum
- Single Wire Protocol

Mobile Device
- Browser
- JSR 257
- MMI
- Midlet
- JSR 177
- Application Processor
- Modern

NFC Controller
- Peer to Peer Interface (P2P)
- R/W Interface
- Card Emulation Interface

NFC Device Standards

Single Wire Protocol

OMA

Java Community Process

ETSI

NFC forum

ISO

ECMA

GSMA

3GPP

GSM Association

A Global Initiative

EMVCo

www.nfc-research.at
© Josef Langer, Andreas Oyrer 2009
What is a SmartCard

- It is a computer, which does not look like a computer
- It is an Embedded System
- No Monitor, no keyboard
- Only a simple communication interface
  - Single Wire Protocol
  - USB
  - T0/T1 protocol
Form Factor SmartCard

- Contact-based
  - Plastic Card ID1
  - Plastic Card (UICC)
  - As USB token

- Contactless
  - Plastic card (Mastercard)
  - Embedded in Jewelry (e.g. ring)
  - Embedded in Watches
Application Domains of Smartcards

- Secure Storage
- Payment
- Authentication
- Signing
- E-Pass
- E-Health
- Ticketing, …
## Components of a SmartCard

<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>8-32 bit CISC / RISC</td>
</tr>
<tr>
<td>Crypto Proc.</td>
<td>AES, DES, ECC</td>
</tr>
<tr>
<td>True Random Generator</td>
<td></td>
</tr>
<tr>
<td>Communication Interface</td>
<td></td>
</tr>
<tr>
<td>ROM</td>
<td>&lt; 512 kB</td>
</tr>
<tr>
<td>RAM</td>
<td>&lt; 100 kB</td>
</tr>
<tr>
<td>EEPROM/Flash</td>
<td>&lt;256kB / &lt; 2MB</td>
</tr>
<tr>
<td>MMU Memory Management Unit</td>
<td></td>
</tr>
</tbody>
</table>
Security of SmartCards

- **Crypto Co Processors:**
  - special math. co-processors that are optimized for the calculation of Crypto algorithms

- **True Random Generator:**
  - Special hardware block, which is responsible for the generation of random numbers
  - Random numbers are often required for the generation of keys in smart cards

- **Memory Management Unit (MMU):**
  - A hardware memory management unit is used to control memory accesses
  - Configuration of the MMU via the smart card operating system
  - The MMU secures access to ROM, RAM and EEPROM
Filename of a Smartcard

MF: Masterfile
DF: Dedicated File
EF: Elementary File
- Internal EF
- Working EF

Source: Rankl, Chipkartenhandbuch
Filesystem of a Smartcard

Example Ticketing:

- Tickets stored in Working EF
- Keys for Authentication stored in Internal EF

Source: Rankl, Chipkarten
Next Generation SmartCards

- SmartCard as Internet Node
- Secure Execution Environment for application protecting values
- Several Applications
Standards for Several Applications

- Payment
  - EMVCo

- Transportation
  - VDV
  - ITSO
  - Calypso
No More SmartCards?

- NFC Handset = Contactless Smartcards
  - Payment
  - Ticketing
  - Loyalty

- Benefits of NFC Handset
  - Display to view Card Content
  - OTA Transactions

- JSR 177: Secure Applications and Trust Services API
Multi Applications on a Card

- Separated by Firewalls
Example: G&D NFC SIM

- Secure storage and execution environment for NFC service applications
- Graphical User Interface (GUI)
  - SmartCard Web Server
  - Midlets on the phone
- NFC - UICC interface
  - Fast reaction times required for certain contactless applications, e.g. ticketing
  - battery-off mode of the NFC device is possible.
Example: ProxSIM from G&D
Use-Case Smart Poster with Secure Element

- User Interaction for activating the Reader/Writer Mode
- SIM Card starts applet and reads Tag
- Then different actions are possible:
  - SMS sending
  - Midlet start
  - SCWS – GUI
  - Launch Browser with HTTP Adress (Cinema Webpage)
Example: ProxSIM from G&D

<table>
<thead>
<tr>
<th>Chip platform</th>
<th>ProxSIM® Gemini</th>
<th>ProxSIM® Libra</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEPROM</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Flash</td>
<td>512 KB</td>
<td>768 KB</td>
</tr>
<tr>
<td>ROM</td>
<td>16 KB</td>
<td>32 KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32 KB</td>
<td>50 KB</td>
</tr>
<tr>
<td>Voltage range</td>
<td>1.8 / 3 / 5 V</td>
<td>1.8 / 3 / 5 V</td>
</tr>
<tr>
<td>Crypto coprocessor</td>
<td>optional</td>
<td>optional</td>
</tr>
<tr>
<td>High-speed capabilities</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>3GPP release status</td>
<td>Rel. 5</td>
<td>Rel. 6</td>
</tr>
</tbody>
</table>

Source: G&D
Example: ProxSIM from G&D

<table>
<thead>
<tr>
<th></th>
<th>ProxSIM® Gemini</th>
<th>ProxSIM® Libra</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIM with</td>
<td>SIM with</td>
<td></td>
</tr>
<tr>
<td>NFC interface</td>
<td>standardized</td>
<td></td>
</tr>
<tr>
<td>(SWP)</td>
<td>NFC interface</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(SWP/HCP)</td>
<td></td>
</tr>
</tbody>
</table>

### Java Card™ functionality

<table>
<thead>
<tr>
<th>Feature</th>
<th>2.2</th>
<th>2.2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Card™</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Cryptographic API</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>- Cryptographic functions</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>- Support of Integer for Java™</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Smart defragmentation</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>GlobalPlatform</td>
<td>OP 2.0,1* / 2.1</td>
<td>GP 2.1,1</td>
</tr>
</tbody>
</table>

### Contactless features

<table>
<thead>
<tr>
<th>Feature</th>
<th>SWP¹</th>
<th>SWP¹ / HCP³</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFC interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contactless protocols:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- ISO 14443 Type A / 14443 Type B</td>
<td>● / ●</td>
<td>● / ●</td>
</tr>
<tr>
<td>- Calypso</td>
<td>–</td>
<td>●</td>
</tr>
<tr>
<td>- Automatic detection</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

* = available  ● = not available

Source: G&D
Secure Applications and Trust Services API

- Keys required to upload Applets
- J2ME Midlets must be signed
- Secure Element can be locked
- Development Tools
  - G&D SmartCafé
  - JCOP Tools from NXP (tools.jcop@nxp.com)
  - Gemalto Dev-Tools
Manufacturers & Card Samples

- G&D ProxSIM
- Oberthur Cosmo Dual 72K
- NXP JCOP 4.1 V2.2.1 72K
- NXP JCOP 3.1 V2.2 36K
- Axalto Cyberflex Palmera V5
- Gemplus GCX4 72k PK
- Gemplus GXPPro-R3.2 E64
- Axalto Cyberflex 32 e-gate
End of Part I

Thank you for your attention!
Agenda Part II

- Secure Element in Details
- Communication with Secure Elements
- Presentation Development Tools
- Communicating with the Secure Element:
  - Mobilephone: J2ME
  - PC: Reader-Writer mode via PC/SC
- Difficulties in Programming Secure Elements
- Benefits and Drawbacks with Secure Element Programming
How to use the secure element?

- „Outside-in“ Approach
- Use existing contactless applets
  - PayPass, Calypso, Oyster
- J2ME app “reads” SIM
  - Display for Smartcard Chip
  - OTA for Smartcard Chip
  - Keyboards for Smartcard Chip
- 2 Components
  - SmartCard Applet
  - Software on Handset
Secure Applications and Trust Services API

- Handset Acts as a reader of Secure Element
- Two Applications needed:
  - J2ME as „GUI“ and „Reader“
  - JavaCard as application in Secure Element
- JSR177 acts as interface between J2ME and internal JavaCard
Dual Interface Card

- Smart Card
  - Contact Card
  - Contactless Card
  - Dual Interface Card
Layers for SmartCard Communication

Reader / Writer

- Application Layer
- Transport Layer
- Datalink Layer
- Physical Layer

SmartCard

- Application Layer
- Transport Layer
- Datalink Layer
- Physical Layer

Application level

APDUs

Error correction

Physical level
Standards Contactless / Contact

![Diagram showing standards comparison between ISO/IEC 7816 and ISO/IEC 14443](image)
Secure Element Communications Overview
JSR177 Security and Trust Services APIs

- defines an API to support communication with smart card applications using the APDU protocol
- defines a Java Card RMI client API
  - allows a J2ME application to invoke a method of a remote Java Card object.
- supports application level digital signature signing
  - but not verification
- allows basic user credential management
- defines a subset of the J2SE cryptography API.
  - support message digest, signature verification, encryption, and decryption
JSR177 compared to JSR257

- JSR177 defines an APDUConnection.
  - Java applications can use this interface to communicate with applications on resident smart cards using the APDU protocol (as defined in ISO-7816-4).
- JSR177 and JSR257 allow the applications to access smart cards.
- JSR177 provides only access to resident smart cards.
- JSR257 allows with ISO14443Connection the access to the whole smart card and the RF interface
- JSR257 provides a lower level interface to the Secure Element
Tools for J2ME Development

- Certificate Required for Secure Element access
  - Java Code Signing Certificate
  - Certificate is needed for access to restricted APIs

- Netbeans IDE (Integrated Development Environment)
  - Good Mobile Integration – High Demand for Resources

- Eclipse Plugin:
  - Formaly known as EclipseME
  - Eclipse Environment
Tools for SE Development

- JCOP Tools from NXP (from IBM)
  - Eclipse Plugin
- G&D Sm@rtCafé
- GPShell
- Gemalto DS
- Gemplus GemXpresso RADIII
- Schlumberger Cyberflex Access
JCOP: Java Card Open Platform

- Open Platform (OP) => Global Platform
- Global Platform
  - Definition of the Life Cycle Management
  - Command Set for Multi Application Smartcard
  - Independed from OS of SmartCard
  - Security/Platform Management
- JavaCard + GP allows multiple applications on one card
Multiapplications

System mode

User task 1

User task 2

User task 3

ROM

EEPROM

RAM

DATA (RAM)

DATA (EEPROM)

CODE

HW-FIREWALL
JavaCard Basics

- Security and communication
- JavaCard Virtual Machine never exits
- Two heaps
  - Volatile and Non-Volatile memory
- Firewall between applications
- Inter-application communication
  - Shared Interface Objects (SIO)
- Atomic Operations
JavaCard Basics

- JavaCard Application Architecture
JavaCard Basics

- JavaCard Communication

Source: http://java.sun.com/javacard/reference/techart/javacard1/
JavaCard Basics

- APDU – Application Protocol Data Unit
  - Logical data packet
  - Exchanged between the CAD and the Java Card Framework
  - Several protocols: T=0, T=1, T=CL
  - Applet processes the command APDU and returns response APDU

![Command APDU Diagram]

![Response APDU Diagram]
JavaCard Basics

- Command APDU
  - CLA: application specific class of instruction (e.g. file access, security)
  - INS: specific instruction within the instruction class
  - P1, P2: instruction parameter
  - LC: number of bytes in the DATA Field (optional)
  - DATA: command data (optional)
  - LE: maximal number of bytes in response (optional)
JavaCard Basics

- **Response APDU**
  - **DATA**: data returned from the applet (optional)
  - **SW1, SW2**: status words; response code
    - Status OK: 90 00
**JavaCard Virtual machine - JCVM**

- **JCVM**
  - Interprets Java "byte code"
  - Is a subset of the Java desktop Virtual machine

- **Supported Java features JavaCard 2.2**
  - small primitive data types: boolean, byte, short
  - one-dimensional arrays
  - packages, classes, interfaces, and exceptions
  - object-oriented features: inheritance, virtual methods, overloading and dynamic object creation
  - access scope, and binding rules
  - garbage collection
  - optional: int
JavaCard Virtual machine - JCVM

- Unsupported Java features JavaCard 2.2
  - characters and strings
  - large primitive data types: long, double, float
  - finalization (and garbage collection prior to JC 2.2)
  - multi-dimensional arrays
  - dynamic class loading
  - security manager
  - object serialization and cloning
  - threads
JavaCard Runtime Environment - JCRE

- **Life time**
  - initialized at card initialization time (only once)
  - after each reset, JCRE enters “receive-process-reply” loop
  - applets and persistent data are preserved over resets

- **Responsible for**
  - card resource management
  - network communication
  - applet execution
  - system and applet security

- ** Defines the JavaCard API**
JavaCard Runtime Engine - JCRE

- responsible for
  - card resource management
  - network communication
  - applet execution
  - system and applet security

- defines the JavaCard API

- Additional JavaCard features
  - persistent and transient objects
  - atomic operations and transactions
  - applet firewall and sharing mechanisms
Java Card Cryptographic Support

- `javacardx.crypto.Cipher`
- `javacard.crypto.Signature`
- `javacard.security.MessageDigest`
- `javacard.security.RandomData`
- `javacard.security.KeyPair`
  - DES, RSA, DSA, Elliptic Curves,
- `javacard.security.KeyAgreement`
  - Diffie Hellman
- `javacard.security.Checksum`;
Secure Element Development Process

1. Write Applet

2. Compile it with Java Compiler
   - checks for language violations
   - generates standard Java byte code
   - performs basic optimizations

3. Convert with JavaCard Converter
   - checks for language subset violations
   - performs additional optimizations at the byte-code level
   - allocates storage and creates VM data structures to represent classes
Secure Element Development Process

4. CAP – Converted Applet
   - executable binary class representations
   - optimized for small memory footprint
   - only one package

5. Export
   - for verification and linking
   - (similar to C header files)

6. Loader/Linker
   - assembles a CAP file into an executable applet or a linkable shared library

7. JCVM
   - Stack machine
   - Executes Java byte-code
JavaCard Applet

- **Definition**
  - an Applet extends the `javacard.framework.Applet` class
  - an Applet is uniquely identified by an AID
  - any number of applets may be installed
  - only one applet is running at a time

- **Applet life cycle**
  - applet’s life starts when it is registered with the JCRE
  - must be explicitly selected by the host
  - purely reactive behaviour
JavaCard Applet Class

```java
public abstract class Applet {
    public static void install(byte[] bArray, short bOffset, byte bLength);
    protected final void register();
    protected final void register(byte bArray, short bOffset, byte bLength);
    public boolean select();
    public void deselect();
    protected final boolean selectingApplet();
    public abstract void process(APDU apdu);
    ...

    public class myApplet extends Applet {
        public static void install(byte[] bArray, short bOffset, byte bLength) {
            (new myApplet()).register(bArray,(short)(bOffset+1),bArray[bOffset]);
        }
        protected myApplet() { // constructor
            ...
        }
    }
}
```

- Creates an instance of the applet subclass
- Should perform any necessary initializations and must call register method
- Installation is successful if the register method does not throw an exception
- After successful installation the applet is selectable
JavaCard Applet Class

```java
public abstract class Applet {
    public static void install(byte[] bArray, short bOffset, byte bLength);
    protected final void register();
    protected final void register(byte bArray, short bOffset, byte bLength);
    public boolean select();
    public void deselect();
    protected final boolean selectingApplet();
    public abstract void process(APDU apdu);
    ...
};
```

- Registers the new applet instance with the JCRE
- Uses the AID specified in the CAP file (only one applet instance possible), or
- The AID passed in bArray (multiple instances possible)
JavaCard Applet Class

```java
public abstract class Applet {
    public static void install(byte[] bArray, short bOffset, byte bLength);
    protected final void register();
    protected final void register(byte[] bArray, short bOffset, byte bLength);
    public boolean select();
    public void deselect();
    protected final boolean selectingApplet();
    public abstract void process(APDU apdu);
    ...
}
```

- Called by the JCRE to inform the applet that it has been selected
- Default applet is selected automatically on card reset
- Returns “false” if the applet cannot be selected (e.g. remaining PIN count is 0) otherwise “true”
JavaCard Applet Class

```java
public abstract class Applet {
    public static void install(byte[] bArray, short bOffset, byte bLength);
    protected final void register();
    protected final void register(byte bArray, short bOffset, byte bLength);
    public boolean select();
    public void deselect();
    protected final boolean selectingApplet();
    public abstract void process(APDU apdu);
}
```

- Called by the JCRE to inform the applet that another (or the same) applet will be selected
- Needed to cleanup before the JCRE gives control to the newly selected applet (e.g. PIN is no longer validated)
JavaCard Applet Class

public abstract class Applet {
    public static void install(byte[] bArray, short bOffset, byte bLength);
    protected final void register();
    protected final void register(byte bArray, short bOffset, byte bLength);
    public boolean select();
    public void deselect();
    protected final boolean selectingApplet();
    public abstract void process(APDU apdu);
}

- called by the JCRE to process an incoming APDU command
- upon normal return the JCRE sends the ISO 7816 defined success code 0x9000 in the APDU response
- selectingApplet: used by the process method to distinguish between applet selects from other SELECT APDU commands

public class myApplet extends Applet {
    public void process(APDU apdu) {
        if (selectingApplet()) {
            ... // the applet was selected
            return;
        }
        ... // process APDU
    }
};
public void process(APDU apdu) throws ISOException
{
    byte[] buffer = apdu.getBuffer();
    // .. process the incoming data and reply
    if (buffer[ISO7816.OFFSET_CLA] == (byte)0)
    {
        switch (buffer[ISO7816.OFFSET_INS])
        {
            case ISO.INS_SELECT:
                ... // send response data to select command
                short Le = apdu.setOutgoing();
                // assume data containing response bytes in replyData[] array.
                if (Le < ..)
                    ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

                apdu.setOutgoingLength((short)replyData.length);
                apdu.sendBytes(replyData, (short)0, (short)replyData.length);
                break;

            case ...
            
        }
    }
}
JavaCard Framework

- javacard.framework.APDU
  - encapsulates an Application Protocol Data Unit according to ISO 7816
  - singleton object owned by the JCRE
  - zeroed out by the JCRE before each new message received

```java
public final class APDU {
    public byte[] getBuffer();
    public static byte getProtocol();

    public short setOutgoing();
    public short setOutgoingNoChaining();
    public short setOutgoingLength(short len);

    public short receiveBytes(short offset);
    public short setIncomingAndReceive();

    public void sendBytes(short offset, short length);
    public void sendBytesLong(byte[] data, short offset, short length);
    public void setOutgoingAndSend(short offset, short length);

    public static APDU getCurrentAPDU();
    public static byte[] getCurrentAPDUBuffer();

    ...

};
```
JavaCard Framework

- javacard.framework.Util
  - common static utility functions
  - provides utility methods for manipulation of arrays and shorts

```java
public class Util {
    public static short arrayCopy(byte[] src, short srcOfs, byte[] dest, short dstOfs, short length);
    public static short arrayFill(byte[] bArray, short offset, short length, byte value);
    public static short arrayCompare(byte[] src, short srcOfs, byte[] dest, short dstOfs, short length);
    public static short makeShort(byte b1, byte b2);
    public static short getShort(byte[] arr, short ofs);
    public static short setShort(byte[] array, short offset, short value);

    ...
}
```
JavaCard 3 - What’s New

- JavaCard 3 supports APDU, but also HTTP(S) for high-speed interfaces, such as USB.
JavaCard 3 – What‘s New

- Classic Applets (Java Card 2 limitations apply for these applications)
  - Communication using APDU protocol
  - Backward compatibility

- Extended Applets
  - Communication using APDU protocol
  - Similar to Classic Applets, and can use all the new APIs, like Threads, Strings

- Servlet Applications
  - Based on Servlet 2.4 API
  - Communication using standard HTTP/HTTPS protocol

JavaCard 3 – What’s New

- high-level architecture for Java Card 3
JavaCard 3 – What’s New

- Java Card 3 also offers full Java language support, including support for
  - All data types except float and double
  - Multiple threads
  - Extensive API support (java.lang, java.util, GCF, and so on)
  - All new Java language syntax constructs, like enums, generics, enhanced for loops, auto boxing/unboxing, and so on
  - Automatic garbage collection
JavaCard Benefits

- Security and stability
  - strictly controlled access to methods
  - object-oriented programming, greater modularity and reusability
  - Applet firewall

- Simple and rapid prototyping

- Storage and management of multiple applications
  - Various applets from different service providers
  - Enlargement of card functionality without the requirement to issue new cards
  - compatibility with existing standards (ISO7816, Global Platform)
JavaCard Technology Drawbacks

- Lower performance compared to assembly language implementations.
  - Max applet size a few 10Kbyte
  - JC 2.2 supported data types: only 8bit to 16bit.

- Higher power consumption due to higher overhead and higher clock rate for comparable performance.
  - But card is most of the time inactive

- It requires more hardware resources (more expensive).
Near Field Communication
Research Lab
Hagenberg

Happy to answer any questions

Josef Langer
Josef.Langer@fh-hagenberg.at

Andreas Oyrer
andreas.oyrer@fh-hagenberg.at