Java Card 3 Programming
Presentation objectives

- Introducing the concepts and the technology of the smart cards
- Describing the protocols between cards and terminals
- Describing how to program the Java Cards
- Exploring the tools and the environments provided by the manufacturers to develop solutions with smart cards
Presentation content

- Introduction
- ISO7816 Protocol
- Java Card
- The basic rules for Java Card programming
- Cyphering
- SIM Card
- Smart Card Web Server
- Java Card 3.0 Connected Edition
- Conclusion
Introduction

History, technology, standards
In this chapter, we'll see

- A brief history of the smart cards
- The applications supported by the smart cards
- The standards supported
Brief history

- Early seventies, first patents
  - Dr Arimura, R Moreno, M Ugon
- Early eighties, first field testing for a memory card
  - Phone card in France
- Mid eighties, large scale introduction of smart cards in banking system
- Mid nineties, SIM card introduced in mobile telephony
What is a smart card

• A plastic card like a credit card with an embedded micro chip
  – With or without visible contacts
    • Maybe contactless

• Standardized
  – ISO 7816
    • Mechanical properties
    • Electrical behavior
    • Communication protocol

• Contains a software which
  – Protects internal data
  – Give access to these data in a secure way
For what applications …

- Payment
- Loyalty systems
- Access systems
- Telephony
  - Mobile (GSM …)
- e-Government
  - ID card, passport
- File system
  - Health
  - Education
  - …
Standards

- ISO 7816
- GSM 11.11 V6.1.0
- GSM 11.14 V7.1.0
  - SIM Toolkit specs
- GSM 03.19 V1.0.0
  - SIM API for Javacard
- ETSI
  - TS 31 130
  - TS 102 241
  - TS 102 588
- Java Card
  - Java Card Forum
- EMV
  - Europay, Mastercard, Visa
- Global Platform
Conclusion

• In this chapter, we have seen
  • A brief history of the smart cards
  • The applications supported by the smart cards
  • The standards supported
ISO7816 Protocol

*Physical description, communication layer, file system*
• In this chapter, we'll see
  • An introduction to the ISO7816 standard
  • What is an APDU
  • How to exchange data between the CAD and the smart card
Mechanical and Electrical Aspects

- ISO 7816 standard describes
  - The physical organisation of the plastic card
  - Indicates the various zones
- It specifies also the purpose and the organisation of the contacts
  - For a smart contacted card
- Possible power voltage
  - 3V or 5V
  - Lower maybe in the future
USB and NFC port

- Recent additions to the SIM card had standardized
  - A USB port in place of the two optional contacts on the bottom of the circuit
  - A NFC (Single Wire Protocol) port for the last optional contact
Half-duplex serial protocol

- Due to the unique pin dedicated to input/output, the first protocol used by the smart cards were
  - Serial
  - Half-duplex

- Communication characteristics:
  - Data: 8 bits
  - Parity: even
  - Stop: 1 bit

- Speed starting at 9600 Bps
USB IC
ETSI TS 102.600

- Max speed 12 Mb/s

- Three flavours:
  - Integrated Circuit Card Devices
    - Compatible with the previous serial protocol
  - Mass Storage
    - Disk emulation
  - Ethernet Emulation Mode
    - To support TCP/IP protocol
NFC SWP
ETSI TS 1002.613 & 622

• Single Wire Protocol
• Full duplex
  • Current and voltage modulation
• Max speed 1.6 Mb/s
• The smart card can act as
  • A RFID tag
  • A RFID tag reader
Terminology

• The smart card reader powered by
  – a PC
  – A cash register
  – a mobile phone
  is called a terminal
• In the standard ISO 7816 it is called:
  – The Card Acceptance Device
  – Or CAD
Answer to Reset

- When a card is inserted into the reader, a micro-switch signals this event to the terminal.
- The terminal powers up the card
  - Using a particular protocol
- When it is properly powered, the card sends back to the terminal a message called "Answer to Reset"
General protocol

- After sending **Answer to Reset**, the card waits until the terminal starts a communication
- The card never starts a communication
- The card answers to a demand coming from the terminal and waits for the next demand
Application Protocol Data Unit

- The APDUs are the commands sent by the terminal to the smart card
- The APDU can
  - Carry parameters to the card
  - Expect results from the card
- Card and terminal must synchronize to
  - The number of bytes to exchange
  - The direction of the exchange
    - This is done by the software embedded in each device
Application Protocol Data Unit

- **CLA**: Class of the APDU: one byte which is characteristic of the APDU of the application
- **INS**: Instruction: this is the command
- **P1, P2**: two parameters which can be combined to form a short integer
- **LC**: Length of parameters which will be exchanged between the terminal and the card (from the terminal to the card, or from the card to the terminal)
No parameters exchanged

- LC == 0
- The card receives the APDU
- It processes it
- It returns a status word
  - Two bytes
Parameters sent by the terminal

- \( LC \neq 0 \)
- \( LC \) indicates the length of the data in bytes
- The software in the terminal and the software in the card must agree on the direction of the exchange
- The card acknowledges by sending back the \textbf{INS} byte
  - Simple case
Data expected by the terminal

- LE ≠ 0
  - The 5th byte is called LE in this case

- The card acknowledges the APDU by sending back the INS byte
  - Simple case

- Data are returned by the card, followed by the status word
Status word

- Status report of the internal operation done by the card
- **0x9000** means success!
- When different, could indicate
  - Denied access
  - File not found
  - No such CLA or INS expected
  - ...
Conclusion

• In this chapter, we have seen
  • An introduction to the ISO7816 standard
  • What is an APDU
  • How to exchange data between the CAD and the smart card
Java Card

Java Card Forum, history of the versions, programming aspects
Introduction

- In this chapter, we'll see
  - An introduction to the Java Card system
  - What is a Java Card Applet
  - What is the Java Card Runtime Environment
  - The lifecycle of an Applet
  - How to protect access with an OwnerPIN
Operating systems

- **Beginning:** proprietary systems
  - Only the applications were standardized
    - B0' for French banking system

- **Now:** multi-application systems
  - MULTOS
  - Windows for Smart Card
    - Dead but replaced by .NET for smart cards
  - Java Card
Java Card History

• Early 1996
  – First development
    • Schlumberger, Bull CP8, GemPlus, Sun
  – Schlumberger's Cyberflex
  – Java Card Forum
    • Most of the smart cards manufacturers
    • Sun
      – As a Java guru
Why Java in a smart card

- Java is an interpreted language
  - Need a Java Virtual Machine to run
- Applications could be portable from one smart card to another
- Applications run securely in a "sand box"
- Small footprint for the applications
Is Java for Java Card pure Java?

- No until Java Card 3.0!
- Roughly:
  - Basic types restricted to
    - Boolean
    - Small integers
      - Byte
      - Short
      - Int (optional)
  - No Strings
  - Arrays restricted to one-dimensional arrays
  - Limited libraries
    - Including java.lang
  - No garbage collector
- Less restrictions for Java Card 3.0
Which version in this course?

- In this course we will introduce the Java Card 3
  - Classic Edition
    - Java Card 2.2.2
    - Connected Edition
Available libraries

- Basically, javacard and javacardx contain the smart card API
  - framework, security and crypto
- java.lang is reduced mainly to the exception definitions
- java.io and java.rmi was introduced in the last 2.2 version
  - java.io to manage channels
  - java.rmi to manage remote method invocation
SIM Toolkit

• For SIM Toolkit two more packages
  – access
  – toolkit

• Will be detailed later
How Java works in a smart card

- A Java Virtual Machine is embedded
  - 4 K bytes
  - Basic library

- **Java Card Runtime Environment**
  - In charge of
    - Activation of applications
    - Low level communication protocol
    - Application downloading
Roles of the JCRE

- Downloading a package
- Creating an instance of an applet
- Selecting an applet
- Transmitting an APDU to a selected applet
- Managing the communication protocol with the CAD
Downloading a package

- Applets must be encapsulated in a package
- External processes
  - Compile the applets
  - Verify the bytecode
  - Create a jar-like container
    - CAP file
    - Will be seen later
- Package and applets are associated an identifier for future selection
What is a Java Card Applet

- A java object which is
  - Running using the JVM
  - Controlled by the JCRE
- The class of this object must extend the class `javacard.framework.Applet`
- The class must overload several methods

```java
package ePurse;
import javacard.framework.*;
class EPurse extends Applet {
    short balance;
    public EPurse(){...}
    public static void install(...){...}
    public boolean select(){...}
    public void process(APDU apdu)
        {...}
}
```
Class APDU

- This class provides the basic features needed to handle the ISO7816 protocol from the applet point of view
- It gives access to the internal buffer dedicated to the communication
- This buffer can be
  - Retrieved by the applet
  - Filled up by the applet and sent to the CAD
Main methods of the APDU

- These methods help to
  - Get the internal buffer
  - Start receiving data
    - Acknowledgement
  - Start transmitting data
- Utilities help to
  - Transform 2 bytes in a short and vice versa
  - Copy buffers
  - Compare buffers

```java
byte buffer[] = apdu.getBuffer();
apdu.setIncomingAndReceive();
short le = apdu.setOutgoing();
apdu.setOutgoingLength(le);
apdu.sendBytes(ISO7816.OFFSET_CDATA, le);
apdu.setOutgoingAndSend(...);
```
Class ISO7816

- This class encapsulates most of the ISO7816 constants needed to program the applets
- Constants are prefixed by
  - CLA for class related constants
  - INS for instruction related constants
  - OFFSET for offsets in the buffer
  - SW for status word related constants

<table>
<thead>
<tr>
<th>Field Summary</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>static byte</td>
<td></td>
</tr>
<tr>
<td><strong>CLA_ISO7816</strong></td>
<td></td>
</tr>
<tr>
<td>APDU command CLA : ISO 7816 = 0x00</td>
<td></td>
</tr>
<tr>
<td>static byte</td>
<td></td>
</tr>
<tr>
<td><strong>INS_EXTERNAL_AUTHENTICATE</strong></td>
<td></td>
</tr>
<tr>
<td>APDU command INS : EXTERNAL AUTHENTICATE = 0x82</td>
<td></td>
</tr>
<tr>
<td>static byte</td>
<td></td>
</tr>
<tr>
<td><strong>INS_SELECT</strong></td>
<td></td>
</tr>
<tr>
<td>APDU command INS : SELECT = 0xA4</td>
<td></td>
</tr>
<tr>
<td>static byte</td>
<td></td>
</tr>
<tr>
<td><strong>OFFSET_CDATA</strong></td>
<td></td>
</tr>
<tr>
<td>APDU command data offset : CDATA = 5</td>
<td></td>
</tr>
<tr>
<td>static byte</td>
<td></td>
</tr>
<tr>
<td><strong>OFFSET_CLA</strong></td>
<td></td>
</tr>
<tr>
<td>APDU header offset : CLA = 0</td>
<td></td>
</tr>
<tr>
<td>static byte</td>
<td></td>
</tr>
<tr>
<td><strong>OFFSET_INS</strong></td>
<td></td>
</tr>
<tr>
<td>APDU header offset : INS = 1</td>
<td></td>
</tr>
<tr>
<td>static byte</td>
<td></td>
</tr>
<tr>
<td><strong>OFFSET_LC</strong></td>
<td></td>
</tr>
<tr>
<td>APDU header offset : LC = 4</td>
<td></td>
</tr>
<tr>
<td>static byte</td>
<td></td>
</tr>
<tr>
<td><strong>OFFSET_P1</strong></td>
<td></td>
</tr>
<tr>
<td>APDU header offset : P1 = 2</td>
<td></td>
</tr>
<tr>
<td>static byte</td>
<td></td>
</tr>
<tr>
<td><strong>OFFSET_P2</strong></td>
<td></td>
</tr>
<tr>
<td>APDU header offset : P2 = 3</td>
<td></td>
</tr>
</tbody>
</table>
Lifecycle of an applet

- The JCRE downloads the package containing the Applet
- It calls the static method `install` on the Applet
- This method creates an instance
  - Or more
- And `register` this instance using an AID
Lifecycle of an Applet

- When the instance is created and registered it can be called
- The JCRE can
  - select
  - deselect the instance
- Can call the instance to **process** an APDU
Example of an Applet

package ePurse;

import javacard.framework.*;

public class ePurse extends Applet {
    public final static byte E_PURSE_CLA = (byte)0x80;
    public final static byte E_PURSE_ADD = (byte)0xB0;
    public final static byte E_PURSE_SUB = (byte)0xB2;
    public final static byte E_PURSE_BAL = (byte)0xB4;

    private short balance;

    public static void install(byte[] bArray, short bOffset, byte bLength) {
        new ePurse();
    }

    protected ePurse() {
        balance = (short)0;
        register();
    }
public void process(APDU apdu) {
    byte[] buffer = apdu.getBuffer();

    if(selectingApplet())
        return;

    if(buffer[ISO7816.OFFSET_CLA] != EPURE_CLA)
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

    short amount = (short)0;
    switch(buffer[ISO7816.OFFSET_INS]){
        case EPURE_ADD:
            apdu.setIncomingAndReceive();
            amount = Util.getShort(buffer,ISO7816.OFFSET_CDATA);
            balance += amount;
            break;
        case EPURE_SUB:
            apdu.setIncomingAndReceive();
            amount = Util.getShort(buffer,ISO7816.OFFSET_CDATA);
            if(amount<=balance)
                balance -= amount;
            else
                ISOException.throwIt(ISO7816.SW_WRONG_DATA);
            break;
case EPURSE_BAL:
    Util.setShort(buffer, ISO7816.OFFSET_CDATA, balance);
    apdu.setOutgoingAndSend(ISO7816.OFFSET_CDATA, (short)2);
    break;
    
    default:
    ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
Simulation script

```java
1  // Test script for Applet 'EPurse'
2  powerup:
3     // Select EPurse //aid/B684893388/03
4     0x00 0xA4 0x04 0x00 0X06 0XB6 0X04 0X09 0X33 0X50 0X04 0x7F;
5
6  // balance
7     0x80 0xB4 0x00 0x00 0x00 0x02;
8
9  // add 0x100
10    0x80 0xB0 0x00 0x00 0x02 0x01 0x00 0x7F;
11
12  // balance
13     0x80 0xB4 0x00 0x00 0x00 0x02;
14
15  // sub 0x50
16    0x80 0xB2 0x00 0x00 0x02 0x00 0x50 0x7F;
17
18  // balance
19     0x80 0xB4 0x00 0x00 0x00 0x02;
20
21  powerdown;
22```
Result

```
Output

Default Device * build.xml (build, run) *

Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x00
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 06, b6, 84, 89, 33, 88, 04, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b4, P1: 00, P2: 00, Lc: 00, Le: 02, 00, 00, SW1: 90, SW2: 00
CLA: 80, INS: b0, P1: 00, P2: 00, Lc: 02, 01, 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b4, P1: 00, P2: 00, Lc: 00, Le: 02, 01, 00, SW1: 90, SW2: 00
CLA: 80, INS: b2, P1: 00, P2: 00, Lc: 02, 00, 50, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b4, P1: 00, P2: 00, Lc: 00, Le: 02, 00, b0, SW1: 90, SW2: 00

run:
BUILD SUCCESSFUL (total time: 4 seconds)
```
Netbeans 6.9
Other Java Card features

- Many features available
  - PIN code management
  - Transaction handling using JCSystem
    - Possibility to group together a certain number of actions into a transaction
    - Possibility to **abort** or **commit** the transaction
  - Shareable applets
  - Possibility to have several applets selected at the same time
• This class helps the developer to protect the access to some features of the smart card using a PIN code

```java
private OwnerPIN pinCode;

/** Creates a new instance of EPurse */
public EPurse() {
    balance = (short) 0;
    pinCode = new OwnerPIN(EPURSE_PIN_TRY_LIMIT,
                            EPURSE_PIN_MAX_SIZE);
}
```
• The CAD must validate the PIN code prior to access the other features

```java
case EPURSE_ADD:
apdu.setIncomingAndReceive();
if (!pinCode.isValidated())
    ISOException.throwIt(
        ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
break;
case EPURSE_PIN:
apdu.setIncomingAndReceive();
if (!pinCode.check(buffer,
    ISO7816.OFFSET_CDATA, EPURSE_PIN_MAX_SIZE))
    ISOException.throwIt(
        ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
break;
```
• The OwnerPIN proposes a method to unblock a blocked PIN code (after a TRY_LIMIT unsuccessful attempts)

```java
case EPURSE_UNBLOCK:
    pinCode.resetAndUnblock();
```
The OwnerPIN proposes a method to reset the validated flag

```java
public boolean select()
{
    pinCode.reset();
}
```
Conclusion

• In this chapter, we have seen
  • An introduction to the Java Card system
  • What is a Java Card Applet
  • What is the Java Card Runtime Environment
  • The lifecycle of an Applet
  • How to protect access with an OwnerPIN
TrUST Me

The key rules for Javacard Programming
Java Card Programming

Issues

- Programming a Java Card seems simple
  - Reduced language
  - Reduced library
  - Most exciting features of Java available in Java Card
  - Most difficulties coming from the ISO7816 protocol hidden by the JCRE and the API
Java Card Programming
Issues

- Powerful tools help developing applets
  - Basic toolkit available for free from Sun (Oracle)
    - Helps testing and debugging applets
  - Enhanced toolkits provided by most of the manufacturers to
    - Upload applets in target Java Cards
    - Test, on board, the uploaded applets
Java Card Programming

Issues

- Most of the trainee’s applets suffer from the following drawbacks:
  - No consistency in data when the card is teared suddenly from the reader
  - Poor usability and security
  - Time out and memory issues not taken in account
• A Java Card applet must be
  • Transaction aware
  • Usable
  • Secure
  • Time-out aware
  • Memory aware
Transaction aware

- **Context**
  - Memorize the ten last operations for an e-purse
  - Operation is qualified by
    - The type
    - The amount
    - The date
case EPURSE_ADD:
apdu.setIncomingAndReceive();
if (!pinCode.isValidated())
    ISOException.throwIt(
        ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
amount = Util.getShort(buffer, ISO7816.OFFSET_CDATA);
balance = (short)(balance + amount);
list.add(buffer, ISO7816.OFFSET_INS,
    ISO7816.OFFSET_CDATA, (short)2,
    (short)(ISO7816.OFFSET_CDATA + (short)2),
    (short)8);
break;
... 

```java
try{
    JCSystem.beginTransaction();
    amount = Util.getShort(buffer, ISO7816.OFFSET_CDATA);
    balance = (short)(balance + amount);
    list.add(buffer, ISO7816.OFFSET_INS,
             ISO7816.OFFSET_CDATA, (short)2,
             (short)(ISO7816.OFFSET_CDATA + (short)2),
             (short)8);
    JCSystem.commitTransaction();

} catch(TransactionException ex){ ... }

break;
```
Usability

- Context
  - On an e-purse, each operation must be accepted only if the user’s PIN code had been validated and if the operation is possible.
case EPURSE_ADD:
    apdu.setIncomingAndReceive();
    if (! pincode.check(buffer, ISO7816.OFFSET_CDATA, (byte)2))
        ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
    amount =
        Util.getShort(buffer, (short)(ISO7816.OFFSET_CDATA + (short)2));
    balance = (short)(balance + amount);
    list.add(buffer, ISO7816.OFFSET_INS,
        (short)(ISO7816.OFFSET_CDATA + (short)2), (short)2,
        (short)(ISO7816.OFFSET_CDATA + (short)4),
        (short)8);
    break;
Usability

- PIN code must not be checked at each operation
  - But at each session starting
- PIN code must be deselected after the applet had been also deselected
Security

• **Context**
  • Iris scan security system with the card holder’s iris characteristics in a smart card

• **Problem:**
  - Which part of the system must decide if the iris scanned corresponds to the data stored in the smart card:
    • The card acceptance device?
    • The Java Card?
Security
(proposed answers)

- **Answer 1:**
  - *The scanned data are passed to the smart card which returns yes or no!*

- **Answer 2:**
  - *The Card Acceptance Device get the stored data from the card to compare it with the scanned data*
Security

- Mutual authentication is needed prior any data exchange
- Card Acceptance Device
  - which is more difficult to replace by a forged one
  - must make the comparison between data stored in the card and the data scanned
**Time out**

- **Context**
  - A message is sent to the Java Card to be encrypted using a first command
  - A second command must be issued to get back the encrypted message
• What if
  • The Java card is teared from the card reader after the first command arrives and before the second command is issued

• Or if
  • The second command arrives before the first one is issued
• Time out aware applet
  
  • Must blank the message to be encrypted if deselect and/or select is called before the second command is issued
  
  • Must refuse the second command if the first was not sent before
Memory aware

- **Context**
  - Memorize the ten last operations for an e-purse
  - Operation is qualified by
    - The type
    - The amount
    - The date
case EPURSE_ADD:
    apdu.setIncomingAndReceive();
    amount = Util.getShort(buffer, ISO7816.OFFSET_CDATA);
    Operation op = new Operation(buffer,
        ISO7816.OFFSET_INS, ISO7816.OFFSET_CDATA, (short)2,
        (short)(ISO7816.OFFSET_CDATA + (short)2), (short)8);
    list.add(op);
    break;
• More Memory aware code
  • Avoid creating object on the fly
  • Create all the objects needed during construction phase
  • Recycle already created objects
Conclusion

At the beginning, smart card programming was done:
- In assembly language
- At a low level
- By engineers aware of the
  - Transactions
  - Usability
  - Security
  - Time-out
  - Memory usage
Conclusion

• Today, thanks to Java Card, applet programming can be done:
  • In Java
  • At a high level
  • By simple Java programmers
Conclusion

• The Java Card programmers must be aware of:
  • Transactions
  • Usability
  • Security
  • Time out
  • Memory usage
Security

*Hardware and software aspects*
Objectives

- In this chapter, we'll see
  - An introduction about the security aspects of the smart cards
    - From a hardware point of view
    - From a software point of view
Hardware security

- A smart card contains important data
  - It could contain money
    - Electronic purses
- It must be tamper resistant
- "If you know the attack you can build the shield"
The attacks

- X raying the micro-chip
- Measuring the power consumption variation during critical APDU
  - When the PIN code is transmitted for example
- Measuring the answer delay
  - To try to predict what branches in the program are completed
The shields

• The micro-chip uses an internal shield to protect itself against an X-Ray scanning
• It guarantees the same delay for both branches of an alternative statement
• It guarantees the same power consumption in all cases
Software attacks and shields

• Data are protected using cryptography
  – Various techniques
    • DES, DES3, AES
    • RSA
    • SHA

• Cryptography is based on
  – A public algorithm
  – A key
    • Private (DES, DES3, AES)
    • Public (RSA)
Symmetric Enciphering

Bob

SAMe KEY

Alice
Asymmetric enciphering

Bob

Bob’s private key

Bob’s Public Key

Alice

Michel Koenig

Smart cards tutorial
Signing using asymmetric keys

Bob’s private key

Bob’s Public Key

Bob

Alice

Signing process:

1. Bob signs a document using his private key.
2. The signed document is sent to Alice.
3. Alice verifies the signature using Bob’s public key.
Certify public key

**X509 Certificate**

- Subject (name, company, e-mail …)
- Issuer’s subject
- Public Key
Certification Authority

- Thawte,
- Verisign,
- ...

Certificate
- Subject (name, company, e-mail ...)
- Start Date
- End Date
- Issuer's subject
- Public Key

Certificate
- Subject (name, company, e-mail ...)
- Start Date
- End Date
- Issuer's subject
- Public Key

Certificate
- Subject (name, company, e-mail ...)
- Start Date
- End Date
- Issuer's subject
- Public Key
Authentication

Authorization

Privacy

Integrity

Non-repudiation
Protect private key
With Smart cards

• The Private key is born, lives and dies inside the card
  – Key pair generation
  – Secure access
  – Cryptographic algorithm process inside the card

• Physically secure
  – No hard drive storage of the private key

• Portable
  – No multi-key
  – Multiple Device

• Enciphering is done inside the card
  – Computer Independent
Hashing (a.k.a. FingerPrint)

• Modifying one bit completely changes the Hash

• Hash result is completely unpredictable

• Usual algorithms are MD5 (used for Linux Password storage) or SHA-1
Digital Signature (Email)

Sender's private key & X509 certificate

Sender

Kps
Sender's PK

Kss
Sender's SK

Receiver

Letter

8365923334
Hash

Signing

Digital Signature 8A!G@3&04

Certificate Authority

CA Public key (certificate checked)

Identification/Authentication of the content of the letter

8365923334
Hash

=?
S/MIME Encryption

Sender

Message

- generate "symmetric document key" (PC)

- encrypt message with symmetric key (PC)

Encrypted message

- get certificate of receiver, verify certificate and extract public key

- encrypt "sym document key" with receiver's public key

Receiver

- unwrap document key with the receiver's private key

- decrypt message with "sym document key"

Message

Trust Centre

Kpr

X.509

- get certificate of receiver, verify certificate and extract public key

- encrypt "sym document key" with receiver's public key

- decrypt message with "sym document key"
Example

```java
package crypto;

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

/**
 * @author Michel Koenig
 */

public class KeyManager {

    private byte [] bufferA = null, bufferB = null;
    private short size = (short)0;
    private RSAPublicKey publicKey=null;
    private RSAPrivateKey privateKey=null;
    private Cipher cip=null;
```
Example

```java
public KeyManager(short size) {
    this.size = size;
    bufferA = new byte[size];
    bufferB = new byte[size];

    KeyPair keyPair = null;
    keyPair = new KeyPair(KeyPair.ALG_RSA, (short)(size * (short)8));
    keyPair.genKeyPair();
    publicKey = (RSAPublicKey)keyPair.getPublic();
    privateKey = (RSAPrivateKey)keyPair.getPrivate();

    cipher = Cipher.getInstance(Cipher.ALG_RSA_PKCS1, false);
}
```
public short getPublicKey(byte buffer[], short offset) throws javacard.framework.ISOException {
    short length = publicKey.getExponent(buffer,offset);
    offset += length;
    length += publicKey.getModulus(buffer,offset);
    return length;
}

public short getModulusPrivateKey(byte buffer[], short offset) throws javacard.framework.ISOException {
    short length = privateKey.getModulus(buffer,offset);
    return length;
}

public short getExponentPrivateKey(byte buffer[], short offset) throws javacard.framework.ISOException {
    short length = privateKey.getExponent(buffer,offset);
    return length;
}
public short encryptPublic(byte [] buffer, short offset, short length) {
    cip.init(publicKey, Cipher.MODE_ENCRYPT);
    Util.arrayCopy(buffer, offset, bufferA, (short)0, length);
    length = cip.doFinal(bufferA, (short)0, length, bufferB, (short)0);
    Util.arrayCopy(bufferB, (short)0, buffer, offset, length);
    return length;
}

public short decryptPublic(byte [] buffer, short offset, short length) {
    cip.init(publicKey, Cipher.MODE_DECRYPT);
    Util.arrayCopy(buffer, offset, bufferA, (short)0, length);
    length = cip.doFinal(bufferA, (short)0, length, bufferB, (short)0);
    Util.arrayCopy(bufferB, (short)0, buffer, offset, length);
    return length;
}
Example

```java
public short encryptPrivate(byte[] buffer, short offset, short length)
{
    cip.init(privateKey, Cipher.MODE_ENCRYPT);
    Util.arrayCopy(buffer, offset, bufferA, (short)0, length);
    length = cip.doFinal(bufferA, (short)0, length, bufferB, (short)0);
    Util.arrayCopy(bufferB, (short)0, buffer, offset, length);
    return length;
}

public short decryptPrivate(byte[] buffer, short offset, short length)
{
    cip.init(privateKey, Cipher.MODE_DECRYPT);
    Util.arrayCopy(buffer, offset, bufferA, (short)0, length);
    length = cip.doFinal(bufferA, (short)0, length, bufferB, (short)0);
    Util.arrayCopy(bufferB, (short)0, buffer, offset, length);
    return length;
}
```
Example

```java
package crypto;

import javacard.framework.*;

/**<p><strong>\* @author Michel Koenig</strong></p>**/

public class Crypto extends Applet {
    private static final byte KMT_CLA = (byte)0x80;

    private static final byte KMT_GET_PUBLIC = (byte)0xB0;
    private static final byte KMT_ENC_PUBLIC = (byte)0xB2;
    private static final byte KMT_DEC_PUBLIC = (byte)0xB4;
    private static final byte KMT_ENC_PRIVATE = (byte)0xB6;
    private static final byte KMT_DEC_PRIVATE = (byte)0xB8;

    private KeyManager keyManager = null;

    public static void install(byte[] bArray, short bOffset, byte bLength) {
        new Crypto();
    }
}
```
Example

```java
protected Crypto() {
    keyManager = new KeyManager((short)64);
    register();
}

public void process(APDU apdu) {
    byte[] buffer = apdu.getBuffer();
    short length = (short)0;

    if (selectingApplet())
        return;

    if (buffer[ISO7816.OFFSET_CLA] != KMT_CLA)
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
```
try{
    switch(buffer[ISO7816_Offset_INS]){
        case KMT_GET_PUBLIC:
            length = keyManager.getPublicKey(buffer, (short) 0);
            apdu.setOutgoing();
            apdu.setOutgoingLength(length);
            apdu.sendBytesLong(buffer, (short) 0, length);
            break;

        case KMT_ENC_PUBLIC:
            length = apdu.setIncomingAndReceive();
            length = keyManager.encryptPublic(buffer, ISO7816_OffsetCDATA, length);
            apdu.setOutgoing();
            apdu.setOutgoingLength(length);
            apdu.sendBytesLong(buffer, ISO7816_OffsetCDATA, length);
            break;
    }
}
Example

```java
case KMT_DEC_PRIVATE:
    length = apdu.setIncomingAndReceive();
    length = keyManager.decryptPrivate(buffer, ISO7816.OFFSETCDATA, length);
    apdu.setOutgoing();
    apdu.setOutgoingLength(length);
    apdu.sendBytesLong(buffer, ISO7816.OFFSETCDATA, length);
    break;

default:
    ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);

} catch (ISOException e) {
    ISOException.throwIt(ISO7816.SW_DATA_INVALID);
}
```
Encrypting w/public

```plaintext
//Test script for Applet 'Crypto'
powerup;
// Select Crypto //aid/B684893388/8D
0x00 0xA4 0x04 0x00 0x00 0xB6 0x84 0x39 0x33 0x88 0x8E 0x7F;
//Send the APDU here
//0x80 0xCA 0x00 0x00 <length> <data> 0x7F;
0x80 0xB2 0x00 0x00 0x10 0x00 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x40;
powerdown;
```

Michel Koenig
Smart cards tutorial
Result

Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x00

CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 06, b6, 84, 89, 33, 88, 8e, Le: 00, SW1: 90, SW2: 00

CLA: 80, INS: b2, P1: 00, P2: 00, Lc: 10, 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0a, 0b, 0c, 0d, 0e, 0f, Le: 40, 66, ff, e8, 04, 8a, 41, 9e, c2, dd, e7, 44, 08, a5, 41, c2, e5, 79, 3d, 65, 31, a5, c6, c8, 54, bd, 49, 52, eb, d3, 65, 0e, b6, da, 99, f0, e4, 89, b6, 08, a4, f6, 64, f9, 3d, ba, bb, 93, 61, f8, a4, 95, 3a, 13, 2d, 17, 73, 7b, 4c, 49, 27, 9f, 1e, 8c, 9c, SW1: 90, SW2: 00
Conclusion

• In this chapter, we have seen
  – An introduction about the security aspects of the smart cards
    • From a hardware point of view
    • From a software point of view
SIM Cards

Proactive SIM cards
Introduction

- In this chapter, we'll see
  - The standards driving the smart cards for mobile telephony
  - What is the SIM Toolkit
  - How Java Card handles the SIM toolkit
  - A full example of a Java Card applet built using the SIM Toolkit library
SIM cards

• Standardized by ETSI for GSM
• GSM 11.11 V6.1.0
  – SIM specs
    • Subscriber Identification Module
• GSM 11.14 V7.1.0
  – SIM Toolkit specs
• GSM 03.19 V1.0.0
  – Javacard SIM API
Proactives SIM

- Using the SIM Toolkit, possibility to
  - Program the SIM
  - Make the SIM card application driving the phone
    - Access to keyboard, display, …
Internal organization

- The SIM contains a certain number of "files" grouped into "directories"
- Terminology:
  - Element File: file
  - Dedicated File: directory
File hierarchy
File hierarchy

File:
- Description: Master file
- Length: 11552 bytes

Access Conditions:
- Delete and Reduce File are controlled by Key 0 (ADM)
- Create and Extend File are controlled by Key 0 (ADM)
- Rehabilitate is controlled by Key 0 (ADM)
- Invalidate is controlled by Key 0 (ADM)

DF Attributes:
- DF is not invalidate
- DF is in utilization phase
- Delete is forbidden
- Reduce is controlled by relevant AC
Proactive SIM

- The ISO7816 standard does not permit that the card starts talking first
  - A card is waiting for an APDU and responds when it receives the APDU
- Proactive SIM cards use a specific status word to indicate to the Mobile Equipment that they want to talk to it
Proactive protocol
Allowed commands for the SIM

- The SIM card can
  - Display text on the phone display
  - Input data from the keyboard
  - Play tone
  - Send a SMS
  - Process an incoming SMS
  - ...

Proactive SIM: DISPLAY TEXT
Proactive SIM: GET INKEY
Proactive SIM: GET INPUT
Proactive SIM: MORE TIME
Proactive SIM: PLAY TONE
Proactive SIM: POLL INTERVAL
Proactive SIM: POLLING OFF
Proactive SIM: REFRESH

Proactive SIM: SELECT ITEM
Proactive SIM: SEND SHORT MESSAGE
Proactive SIM: SEND SS
Proactive SIM: SEND USSD
Proactive SIM: SET UP CALL
Proactive SIM: SET UP MENU
Proactive SIM: PROVIDE LOCAL INFORMATION (MCC, MNC, LAC, Cell ID&IMEI)
Proactive SIM: PROVIDE LOCAL INFORMATION (NMR)
SIM Toolkit applet

• A SIM Toolkit applet must
  – Import `sim.access` and `sim.toolkit` packages
  – Extend the `javacard.framework.Applet`
  – Implement the interfaces
    • `ToolkitInterface`
    • `ToolkitConstants`
• Example:

```java
import sim.toolkit.*;
import sim.access.*;
import javacard.framework.*;

public class MyApplet1 extends javacard.framework.Applet implements ToolkitInterface, ToolkitConstants {
    // Mandatory variables
    private SIMView gsmFile;
    private ToolkitRegistry reg;
```
SIMView

- The **SIMView** interface is the interface between the applet and the GSM filesystem
- It proposes
  - Constants to identify in a simple way the regular GSM files
  - Methods to access these files
• Example:

```java
/** DF under MF */
/** File identifier : DF TELECOM = 0x7F10 */
public static final short FID_DF_TELECOM = (short)0x7F10;
/** File identifier : DF GSM = 0x7F20 */
public static final short FID_DF_GSM = (short)0x7F20;
/** File identifier : DF DCS-1800 = 0x7F21 */
public static final short FID_DF_DCS_1800 = (short)0x7F21;
/** File identifier : DF IS-41 = 0x7F22 */
public static final short FID_DF_IS_41 = (short)0x7F22;
/** File identifier : DF FP-CTS = 0x7F23 */
```
• Example:

```java
public short select(short fid,
                     byte fci[],
                     short fciOffset,
                     short fciLength) throws
                        NullPointerException,
                        ArrayIndexOutOfBoundsException,
                        SIMViewException;
```
SIMSystem

- The **SIMSystem** class provides one method which is
  - `SIMView getTheSIMView()`
The SIM Applet communicates with the mobile equipment through the ToolkitRegistry. The SIM applet gets an entry from the ToolkitRegistry in order to receive and process the events sent by the mobile equipment, and to send commands to the mobile equipment.
SIM Toolkit applet

• Example

```java
// Main Menu
private byte idMenu1;
private byte[] Menu1;

public MyApplet1() {
    // Get the GSM application reference
gsmFile = SIMSystem.getTheSIMView();
    // Get the reference of the applet ToolkitRegistry object
    reg = ToolkitRegistry.getEntry();
    /**@todo: Customize your menu titles here*/
    Menu1 = new byte[] { (byte) '1', (byte) ' ', (byte) 'M', (byte) 'e',
        (byte) 'n', (byte) 'u', (byte) '1' };
    // Define the applet Menu Entry
    idMenu1 = reg.initMenuEntry(Menu1, (short) 0, (short) Menu1.length,
        PRO_CMD_SELECT_ITEM, false, (byte) 0, (short) 0);
}
public byte initMenuEntry(
    byte[] menuEntry,  /* the menu entry string */
    short offset,      /* its offset */
    short length,      /* its length */
    byte nextAction,   /* action associated */
    boolean helpSupported, /* true if help available */
    byte iconQualifier,
    short iconIdentifier /* 0 if no icon */
)
throws java.lang.NullPointerException,
    java.lang.ArrayIndexOutOfBoundsException,
    ToolkitException,
    TransactionException
/**
 * Method called by the JCRE at the installation of the applet
 * @param bArray the byte array containing the AID bytes
 * @param bOffset the start of AID bytes in bArray
 * @param bLength the length of the AID bytes in bArray
 */

public static void install(byte[] bArray, short bOffset, byte bLength) {
    // Create the Java SIM toolkit applet
    MyApplet1 StkCommandsExampleApplet = new MyApplet1();
    // Register this applet
    StkCommandsExampleApplet.register(bArray,
        (short) (bOffset + 1), (byte) bArray[bOffset]);
}
/**
 * Method called by the SIM Toolkit Framework
 * @param event the byte representation of the event triggered
 */

public void processToolkit(byte event) {
    // Manage the request following the MENU SELECTION event type
    if (event == EVENT_MENU_SELECTION) {
        // Get the selected item
        EnvelopeHandler envHdlr = EnvelopeHandler.getTheHandler();
        byte selectedItemId = envHdlr.getItemIdIdentifier();
        // Perform the required service following the Menu1 selected
        // item
        if (selectedItemId == idMenu1) {
            menu1Action();
        }
    }
private byte [] helloWorld;
private void menu1Action() {

    // Get the received envelope
    ProactiveHandler proHdlr = ProactiveHandler.getTheHandler();

    helloWorld = new byte[]{(byte)'H', (byte)'e', (byte)'l', (byte)'l',
                           (byte)'o', (byte)' ', (byte)'w', (byte)'o', (byte)'r', (byte)'l', (byte)'d'};

    // Initialize the display text command
    proHdlr.initDisplayText((byte) 0x00, DCS_8_BIT_DATA, helloWorld, (short) 0, (short) (helloWorld.length));

    proHdlr.send();
    return;
}


Running
Documentation

• More documentation in
  – 3gpp 43019-560
Conclusion

In this chapter, we have seen:

- The standards driving the smart cards for mobile telephony
- What is the SIM Toolkit
- How Java Card handles the SIM toolkit
- A full example of a Java Card applet built using the SIM Toolkit library
Smart Card Web Server

An other way for the SIM card to control the handset
Introduction

• In this chapter, we'll see:
  • A new approach to interface the applications in the SIM card, using the handset
  • The architecture of the SCWS
  • A full application for a SIM card supporting SCWS
Introduction

• SIM Toolkit was introduced at the time when handset had few capabilities for interfacing
  • Text oriented display
  • No graphics
  • Hierachical menus
• Modern handsets support
  • Full color graphic interface
  • Point and pin menus
Introduction

- Axalto developers proposed at Cartes 2000 a simplified web server inside the SIM card
  - SESAME 2000
- With
  - the introduction of the USB port
  - the powerfulness of modern SIM card
  - the size of SIM applications
  
this solution was rapidly adopted and standardized
The standard adopted is called: **Smart Card Web Server**

This standard supposes:

- A TCP/IP link
  - On USB
- A TCP/IP stack on board
SCWS

Low Level Resources

Java Virtual Machine

Javacard APIs

SCWS API

STK Api

Applets
Packages and classes

/
* Imported packages
*/
import javacard.framework.*;
import uicc.scws.HttpRequest;
import uicc.scws.HttpResponse;
import uicc.scws.ScwsConstants;
import uicc.scws.ScwsException;
import uicc.scws.ScwsExtension;
import uicc.scws.ScwsExtensionRegistry;
ScwsConstants

• MIME types
  – CONTENT_TYPE_IMAGE_GIF
  – CONTENT_TYPE_TEXT_HTML

• Status code
  – SC_OK (200)
  – SC_NOT_FOUND (404)

• Parsing tags
  – URI_QUERY_TAG
ScwsExtension

• The applet (servlet!) working in SCWS mode must implement ScwsExtension
• That means overriding the methods
  – doGet()
  – doPost()
  – doHead()
  – ...

HttpRequest

• Not really the J2EE HttpRequest but enough to extract data from a HTTP request
• Provides methods like
  – `findAndCopyKeywordValue`
  – `getContentLength`
  – `getContentType`
• As for `HttpRequest`, helps the user to provide an HTTP response to the request
• Provides methods like
  – `setContentType()`
  – `appendContent()`
  – `writeStatusCode()`
  – `flush()`
Example

• In the next servlet, the strings are encoded as arrays of bytes
  – Strings are not supported by Java Card 2
• In the next two pages, the pseudo code written in comment show how the servlet would be written if String was supported by this release of Java Card
Example

/*

public class HelloWorld extends javacard.framework.Applet implements AppletEvent, ScwsExtension {

public final static String url = "/HelloWorld";
public final static String appId = "HelloWorld;

public byte[] temporaryBuffer;
public final static short TEMPORARY_BUFFER_LENGTH = (short) 100;
public final static String HTML_BEGIN = "<html>"+"<head>"+
"<title>"+"Hello"+"</title>"+"</head>"+
"<body BGCOLOR=""#FFFFFF"">"+"<center>";
public final static String HELLO = "Hello ";
public final static String HTML_END = "</center>"+"</body>"+
"</html>"
Example

- Unfortunately String are not yet supported by Java Card
  - Strings are supported by Java Card 3
- The arrays of bytes are not so easy to read, but the result is the same
public class HelloWorld extends javacard.framework.Applet implements AppletEvent, ScwsExtension {

/** the servlet url */

public final static byte[] url = {
        (byte)'/', (byte)'H', (byte)'e',
        (byte)'l', (byte)'l', (byte)'o',
        (byte)'W', (byte)'o', (byte)'r',
        (byte)'l', (byte)'d' 
};

public final static byte[] appId = {
        (byte)'H', (byte)'e', (byte)'l',
        (byte)'l', (byte)'o',
        (byte)'W', (byte)'o', (byte)'r',
        (byte)'l', (byte)'d' 
};
// Temporary operation buffer

class
{
    public byte[] temporaryBuffer;

    public final static short TEMPORARY_BUFFER_LENGTH = (short) 100;

    public final static byte[] HTML_BEGIN = {
        (byte)'<', (byte)'h', (byte)'t', (byte)'m', (byte)'l', (byte)'>',
        (byte)'<', (byte)'h', (byte)'e', (byte)'a', (byte)'d', (byte)'>',
        (byte)'<', (byte)'t', (byte)'i', (byte)'t', (byte)'l', (byte)'e',
        (byte)'H', (byte)'e', (byte)'l', (byte)'l', (byte)'o',
        (byte)'</', (byte)'t', (byte)'i', (byte)'t', (byte)'l', (byte)'e',
        (byte)'H', (byte)'e', (byte)'a', (byte)'d', (byte)'>',
        (byte)'</', (byte)'h', (byte)'e', (byte)'a', (byte)'d', (byte)'>',
    };

    public final static byte[] HTML_END = {
        (byte)'/', (byte)'h', (byte)'t', (byte)'m', (byte)'l', (byte)'e',
        (byte)'/', (byte)'h', (byte)'e', (byte)'a', (byte)'d', (byte)'e',
        (byte)'/', (byte)'h', (byte)'e', (byte)'a', (byte)'d', (byte)'e',
        (byte)'/', (byte)'h', (byte)'e', (byte)'a', (byte)'d', (byte)'e',
    };

    // Constructor
    public MyClass()
    {
        // Initialization code
    }

    // Methods
    public void methodName()
    {
        // Method body
    }

    // Properties
    public int property;
}

// Class declaration

Example

```
(byte) '\<', (byte) '"', (byte) 'b', (byte) 'o', (byte) 'd', (byte) 'y', (byte) ' ',
(byte) 'B', (byte) 'G', (byte) 'C', (byte) 'O', (byte) 'L', (byte) 'O', (byte) 'R',
(byte) '='', (byte) '"', (byte) '#', (byte) 'F', (byte) 'F', (byte) 'F', (byte) 'F',
(byte) 'F', (byte) 'F', (byte) 'F', (byte) 'F', (byte) 'F', (byte) 'F', (byte) 'F',
(byte) '<', (byte) 'c', (byte) 'e', (byte) 'n', (byte) 't', (byte) '>',
(byte) '<', (byte) '/c', (byte) 'e', (byte) 'n', (byte) 't', (byte) '>',
(byte) '<', (byte) '/b', (byte) 'o', (byte) 'd', (byte) 'y',
(byte) '<', (byte) '/h', (byte) 't', (byte) 'm', (byte) 'l', (byte) '>',
```

```
public final static byte[] HELLO = {
(byte) 'H', (byte) 'e', (byte) 'l',
(byte) 'l', (byte) 'o', (byte) ' ',
};
```

```
public final static byte[] HTML_END = {
(byte) '<', (byte) '/c', (byte) 'e', (byte) 'n', (byte) 't', (byte) '>',
(byte) '<', (byte) '/b', (byte) 'o', (byte) 'd', (byte) 'y',
(byte) '<', (byte) '/h', (byte) 't', (byte) 'm', (byte) 'l', (byte) '>',
};
```
public HelloWorld(byte[] buffer, short offset, byte length) {
    // First LV is instance AID
    short aid = offset;
    offset += buffer[offset] + (byte) 1;
    // Second LV is Privilege
    offset += buffer[offset] + (byte) 1;
    // Third LV is specific install parameter (extract from TAG C9)
    offset++; // skip C9 Length
    // Register the new applet instance to the JCRE
    register(buffer, (short) (aid + (short) 1), buffer[aid]);
    // Register application id, there is corresponding appId in the
    // Run/Debug configuration for URL Mapping
    ScwsExtensionRegistry.register(this, appId, (short) 0,
                                 (short) appId.length);
try {
    // Create a temporary buffer for read/write
    temporaryBuffer = JCSystem.makeTransientByteArray(TEMPORARY_BUFFER_LENGTH, JCSystem.CLEAR_ON_RESET);
} catch (SystemException se) {
    // create buffer in persistent memory as not enough transient
    // is available
    temporaryBuffer = new byte[TEMPORARY_BUFFER_LENGTH];
}
Example

```java
public void doGet(HttpRequest req, HttpResponse resp) throws ScwsException {
    try {
        resp.writeStatusCode(ScwsConstants.SC_OK);
        resp.setContentType(ScwsConstants.CONTENT_TYPE_TEXT_HTML);
        resp.enableChunkMode();
        short queryLength = req.findAndCopyKeywordValue(ScwsConstants.URI_QUERY_TAG, temporaryBuffer,(short)0,
                                                          (short)temporaryBuffer.length);
        resp.appendContent(HTML_BEGIN,(short)0,(short)HTML_BEGIN.length);
        resp.appendContent(HELLO, (short)0, (short)HELLO.length);
        resp.appendContent(temporaryBuffer, (short)0, queryLength);
        resp.appendContent(HTML_END,(short)0,(short)HTML_END.length);
    } catch(Exception e) {resp.writeStatusCode(ScwsConstants.SC_BAD_REQUEST);}
    resp.flush();
```
Static HTML

• Static HTML file: helloworld.html

```html
<html>
<body>
<p>Simagine HelloWorld</p>
<br>
<form action="/HelloWorld" method="get">
    <input name="name" type="text">
    <br>
    <input value="Type in your name" type="submit">
</form>
</body>
</html>
```
Running
Conclusion

• In this chapter, we have seen:
  • A new approach to interface the applications in the SIM card, using the handset
  • The architecture of the SCWS
  • A full application for a SIM card supporting SCWS
Java Card 3.0 Connected Edition

*A new and rich flavour of Java Card*
Introduction

• In this chapter, we'll see
  • The main enhancements introduced by Java Card 3
  • The restrictions of Java Card 3 compared to Java SE
  • A full example of a servlet
Features

• Java Card 3.0 has two editions:
  • The Classic Edition
    – Compatible with Java Card 2
    – Applications are built with applets
  • The Connected Edition
    – With a WEB server embedded
    – HTTP, TCP/IP over USB
Features

- Java Card 3.0 classic edition remains applet oriented
- Java Card 3.0 connected edition is servlet oriented
  - Specifications of the supported servlets are extracted from the Servlet API Specifications 2.4
    - Everything which deals with floating point numbers, J2EE, etc. are not taken in account.
Features

• But, like traditional servlets, the Java Card 3 servlets support the methods:
  • doGet
  • doPost
  • doHead
  • doPut
  • doDelete
  • doOptions
  • doTrace
Features

• Better support of the Java language
  • All data types except float and double
  • Multiple threads
  • Extensive API support (java.lang, java.util, GCF, and so on)
  • Direct handling of class files, with all loading and linking on card
  • All new Java language syntax constructs, like enums, generics, enhanced for loops, auto boxing/unboxing, and so on
  • Automatic garbage collection
Architecture
Example

- The following example is created with NetBeans 6.9 with the Java Card wizard
- It is the web instance of the very well known « Hello world » program
  - Most code is automatically generated by the Java Card wizard
Example
Example

The image shows a software interface for choosing a project. The left panel is labeled "Steps" and includes the following steps:

1. Choose Project
2. ...

The right panel is labeled "Choose Project" and contains categories and projects. The categories include:

- Java
- Java Web
- Java EE
- Java Card
- Java ME
- Maven
- Groovy
- NetBeans Modules
- Samples

The projects listed in the "Projects" tab include:

- Classic Applet Project
- Extended Applet Project
- Classic Library Project
- Extension Library Project
- Web Project

The description field indicates:

Web Application for Java Card™ 3 Platform [Connected Edition]

The interface also includes buttons for back, next, finish, cancel, and help.
Example
Example
Example

```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sayhello;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```
Example

```java
public class SayHello extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html><head><title>SayHello</title></head>");
            out.println("<body><h1>SayHello</h1>");
            out.println("Hello from sayhello.SayHello to");
            out.println(request.getParameter("name"));
            out.println("</body></html>");
        } finally {
            out.close();
        }
    }
}
```
Example
Example

```html
<html>
<head>
  <title>SayHello</title>
</head>
<body>
  <div style="text-align:center">
    <form method="get" action="/sayhello/sayhello">
      <table border=0>
        <tr>
          <td>Your Name:</td>
          <td><input type="text" name="name"/></td>
        </tr>
        <tr>
          <td>&nbsp;</td>
          <td><input type="submit" value="Say Hello"/></td>
        </tr>
      </table>
    </form>
  </div>
</body>
</html>
```
Example
Example
Example
Example

```xml
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/javacard/jcweb-app_3_0.xsd">
  <display-name>Say Hello</display-name>
  <servlet>
    <servlet-name>SayHello</servlet-name>
    <servlet-class>sayhello.SayHello</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SayHello</servlet-name>
    <url-pattern>/sayhello</url-pattern>
  </servlet-mapping>
</web-app>
```
Example
Example

SayHello

Hello from sayhello. SayHello to Michel
Conclusion

• In this chapter, we have seen
  • The main enhancements introduced by Java Card 3
  • The restrictions of Java Card 3 compared to Java SE
  • A full example of a servlet
Conclusion
In 1996, the Java Card system changed dramatically the way to program secure applications for smart cards. Despite many concurents on the field, this system remains today the first language for smart cards in the world. Combined with Java for Mobile Equipment it represents the solution to develop secure applications for the future powerful smartphones.