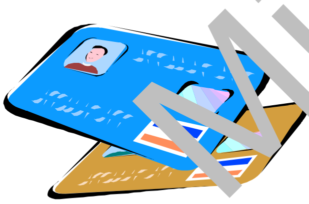
Smart Cards

A comprehensive tutorial

Michel Kœnig

University of Nice-Sophia Antipolis - ESSI

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
- Security
- Java Card
- Client side programming
- Tools and environments
- Cyphering
- Conclusion
- Ap 1: Remote Method Invocation
- Ap 2: SIM Card

Introduction

History, technology, standards

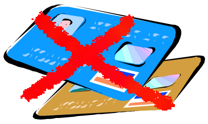
Objectives

- In this chapter, we'll see
  - A brief history of the smart cards
  - What is a smart card
  - What are the applications available from the smart cards
  - What kind of prerequisites are needed to attend this tutorial

Brief history

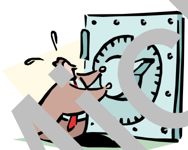
- Early seventies, first patents
  - Dr Arimura, R Moreno, M Ugon
- Early eighties, first field testing for a memory card
  - Phone card in France
- Mid eighties, large scale introduction of smart cards in banking system
- Mid nineties, SIM card introduced in mobile telephony

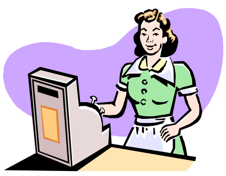
What is a smart card



- A plastic card like a credit card with an embedded micro chip
  - With or without visible contacts
    - Maybe contactless
- Standardized
  - ISO 7816
    - Mecanical 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 …)
- File system
  - Health
  - Education
  - …

Standards

- **ISO 7816**
  - Mecanical level
  - Electrical and communication protocol
- **GSM 11.11 V6.1.0**
  - SIM specs
- **GSM 11.14 V7.1.0**
  - SIM Toolkit specs
- **GSM 03.19 V1.0.0**
  - SIM API for Javacard
- **Java Card**
  - Java Card Forum
- **EMV**
  - Europay, Mastercard, Visa
- **Open Platform**

Content of this tutorial

- Exploring how a terminal can communicate with a smart card
- Discussing about security
- Understanding the organization of a Java Card
- Learning how to program a Java Card
- Discovering the tools available to program, test and deploy Java Cards

Conclusion

- In this chapter, we have seen
  - A brief history of the smart cards
  - What is a smart card
  - What are the applications available from the smart cards
  - What kind of prerequisites are needed to attend this tutorial

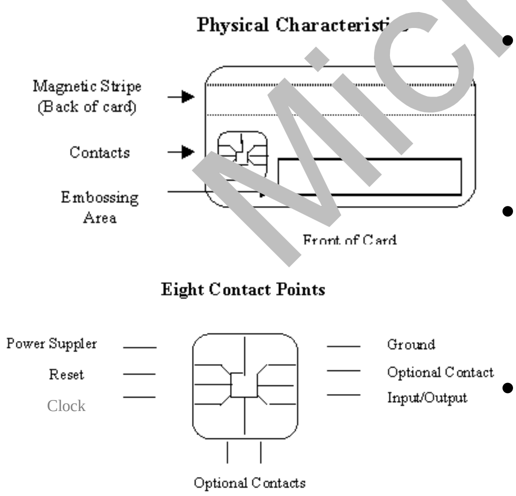
ISO7816 Protocol

Physical description, communication layer, file system

Objectives

- In this chapter, we'll see
  - An introduction to the ISO 7816 Protocol
  - Some mecanical and physical aspects of the cards standardized by ISO 7816
  - An extract of the protocol about the data communication

Mecanical 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 contactful card
- Possible power voltage
  - 3V or 5V
  - Lower maybe in the future

Half-duplex serial protocol

- Due to the unique pin dedicated to input/output, the protocol is
  - Serial
  - Half-duplex
- Com caracteristics:
  - Data: 8 bits
  - Parity: even
  - Stop: 1 bit
- Speed starting at 9600 Bps

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 **C**ard **A**cceptance **D**evice
  - 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`**"

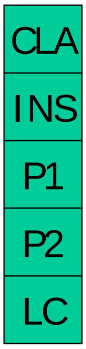
Michel Koenig Smart cards tutorial 17

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

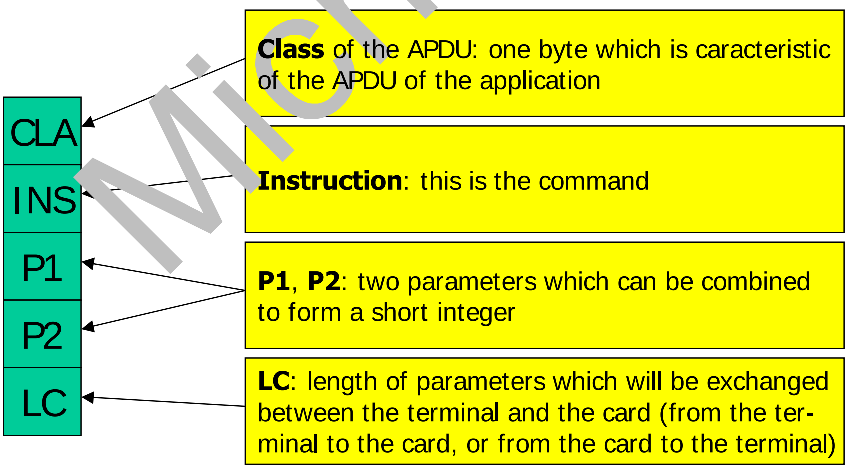
Michel Koenig Smart cards tutorial 18

Application **P**rotocol **D**ata **U**nit

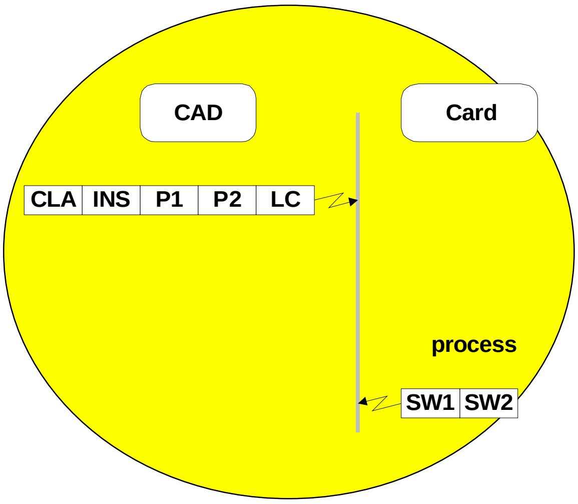


- The APDUs are the commands sent by the terminal to the smart card
- The APDU can
  - carry parameters to the card
  - Expect results from the card
- Card and terminal must synchronize to
  - the number of bytes to exchange
  - The direction of the exchange
    - This is done by the software embedded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

Application **P**rotocol **D**ata **U**nit

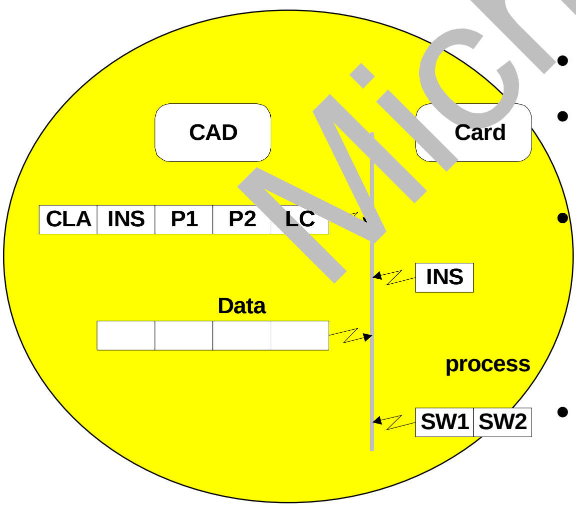


No parameters exchanged



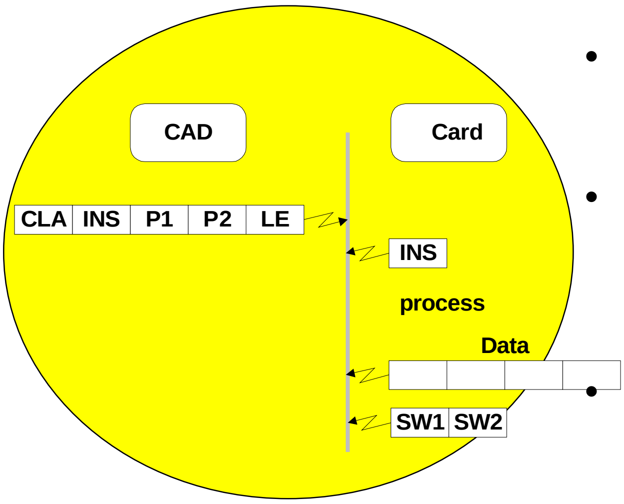
- **LC** ==0
- The card receives the APDU
- It processes it
- It returns a status word
  - Two bytes

Parameters sent by the terminal



- **LC** ¹ 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 **INS** byte
  - Simpliest case

Data expected by the terminal



- **LE** ¹ 0
  - The 5th byte is called LE in this case
- The card acknowledges the APDU by sending back the **INS** byte
  - Simpliest 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 ISO 7816 Protocol
  - Some mecanical and physical aspects of the cards standardized by ISO 7816
  - An extract of the protocol about the data communication

Java Card

Java Card Forum, history of the versions, programming aspects

Objectives

- In this chapter, we'll see
  - The various operating systems available for the smart cards
  - An introduction to the Java Card system
  - How the Applet are working
  - Some classes and methods provided by the Java Card API

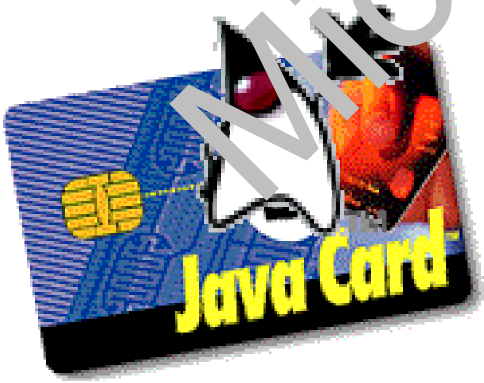
Operating systems

- Beginning: proprietary systems
  - Only the applications were standardized
    - B0' for french banking system
- Now: multi-application systems
  - MULTOS
  - Windows for Smart Card
    - Dead
  - 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"
- Byte code is small

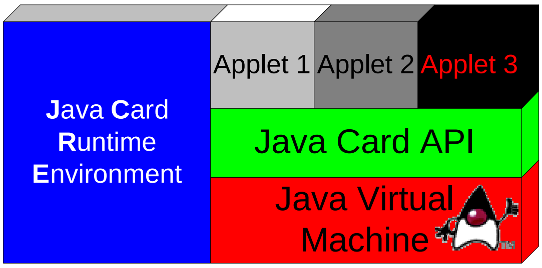
Is Java for Java Card pure Java?

- No!
- Roughly:
  - Basic types restricted to
    - Boolean
    - Small integers
      - Byte
      - Short
      - Int (optionnal)
  - Arrays restricted to one-dimensionnal arrays
  - Limited libraries
    - Including java.lang

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 version
  - **`java.io`** to manage channels
  - **`java.rmi`** to manage remote method invocation

How Java works in a smart card

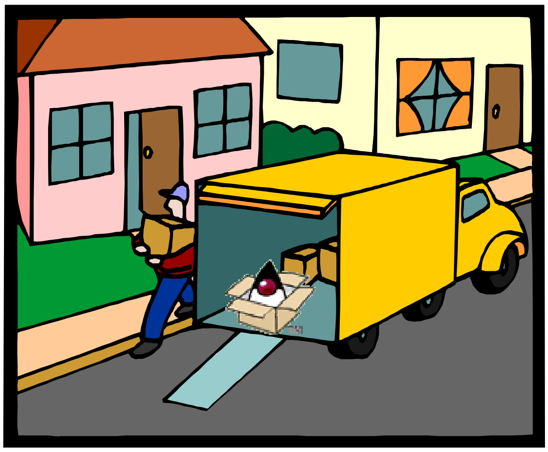


- A Java Virtual Machine is embedded
  - 4 K bytes
  - Basic library
- **J**ava **C**ard **R**untime **E**nvironment
  - In charge of
    - Activation of applications
    - Low level communication protocol
    - Application downloading

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



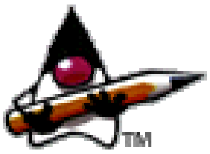
- Applets must be encapsulated in a package
- External processes
  - Compile the applets
  - Verify the bytecode
  - Create a jar-like container
    - CAP file
  - Will be seen later
- Package and applets are associated an identifier for future selection

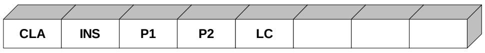
What is a Java Card Applet

```
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)
        {…}
}
```

- 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

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

```
byte buffer[] = apdu.getBuffer();

apdu.setIncomingAndReceive();

short le = apdu.setOutgoing();
apdu.setOutgoingLength(le);
apdu.sendBytes(ISO7816.OFFSET_CDATA,
                                  le);

apdu.setOutgoingAndSend();
```

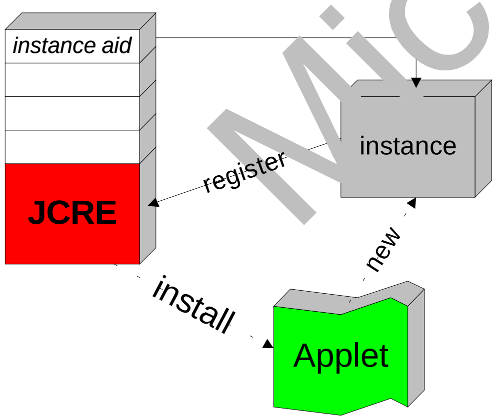
- These methods help to
  - Get the internal buffer
  - Start receiving data
    - Acknowledgement
  - Start transmitting data
- Utilities help to
  - Transform 2 bytes in a short and vice versa
  - Copy buffers
  - Compare buffers

Class ISO7816

Member Summary	
Fields	
static byte	CLA_ISO7816 APDU command CLA : ISO 7816 = 0x00
static byte	INS_EXTERNAL_AUTHENTICATE APDU command INS : EXTERNAL AUTHENTI
static byte	INS_SELECT APDU command INS : SELECT = 0xA4
static byte	OFFSET_CDATA APDU command data offset : CDATA = 5
static byte	OFFSET_CLA APDU header offset : CLA = 0
static byte	OFFSET_INS APDU header offset : INS = 1
static byte	OFFSET_LC APDU header offset : LC = 4
static byte	OFFSET_P1 APDU header offset : P1 = 2
static byte	OFFSET_P2 APDU header offset : P2 = 3
static short	SW_APPLET_SELECT_FAILED Response status : Applet selection failed = 0x6999;

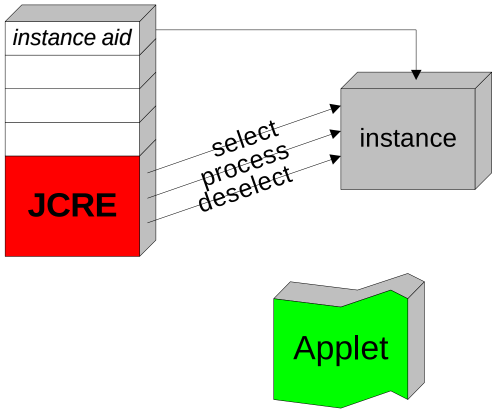
- This class encapsulates most of the ISO7816 constants needed to program the applets
- Constants are prefixed by
  - CLA for class related constants
  - INS for instruction related constants
  - OFFSET for offsets in the buffer
  - SW for status word related constants

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`**

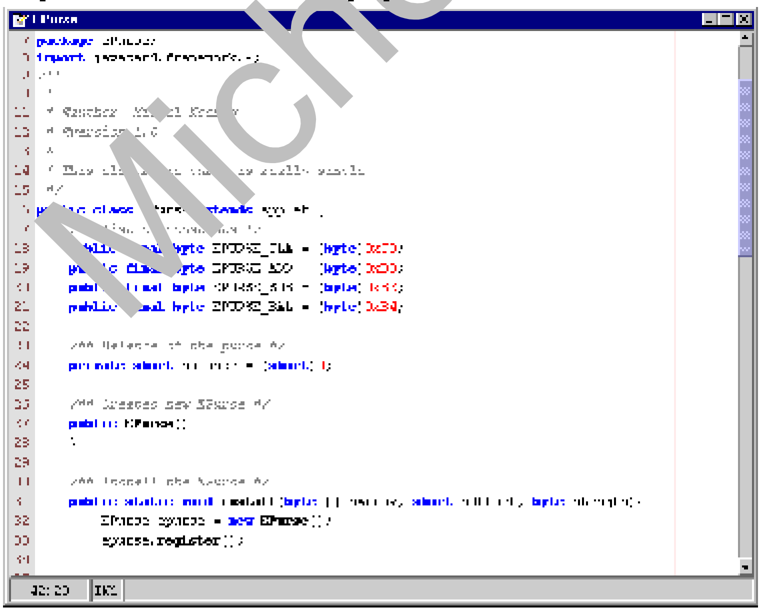
Lifecycle of an Applet

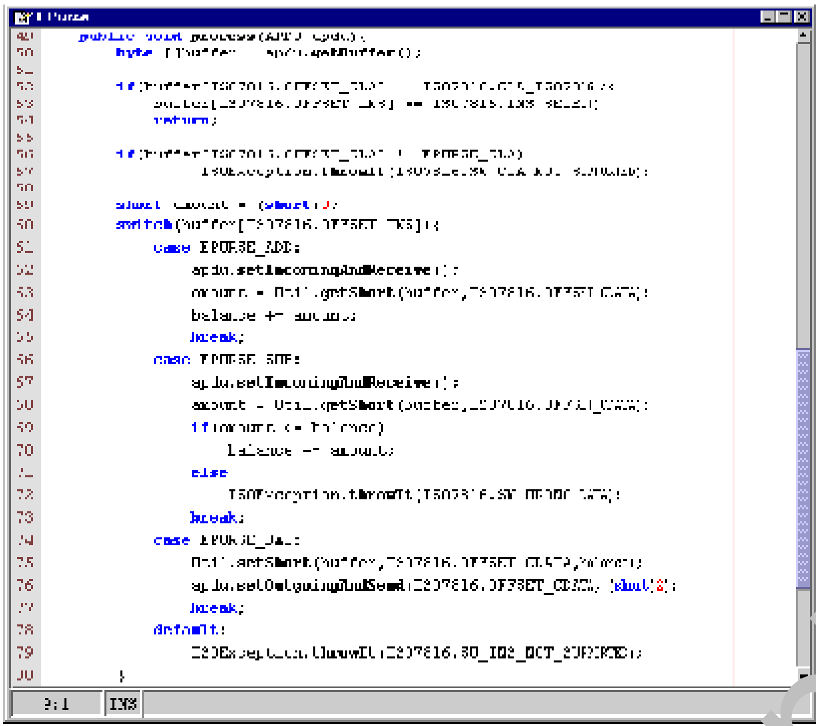


- When the instance is created and registered it can be called
- The JCRE can
  - **`select`**
  - **`deselect`**

  the instance
- Can call the instance to **`process`** an APDU

Example of an Applet





Converting the **`.class file`**

- The file **`EPurse.class`** must be converted before downloading
- The downloading format is called a **`CAP File`**
- The tool **`converter`**
  - Makes the conversion
  - Assigns an AID to the package
  - Assigns an AID to the applet

The converter

- The converter can be called using a configuration file : **EPurse.opt**

Directory where the files will be exported

Applet AID

```
-out EXP JCA CAP
-exportpath C:\jc211\api21
-applet 0xa0:0x0:0x0:0x0:0x62:0x3:0x1:0xc:0x3:0x1 ePurse.EPurse
ePurse
0xa0:0x00:0x00:0x00:0x62:0x03:0x01:0x0c:0x03 1.0
```

Package name

Package AID

Applet name

Conversion result

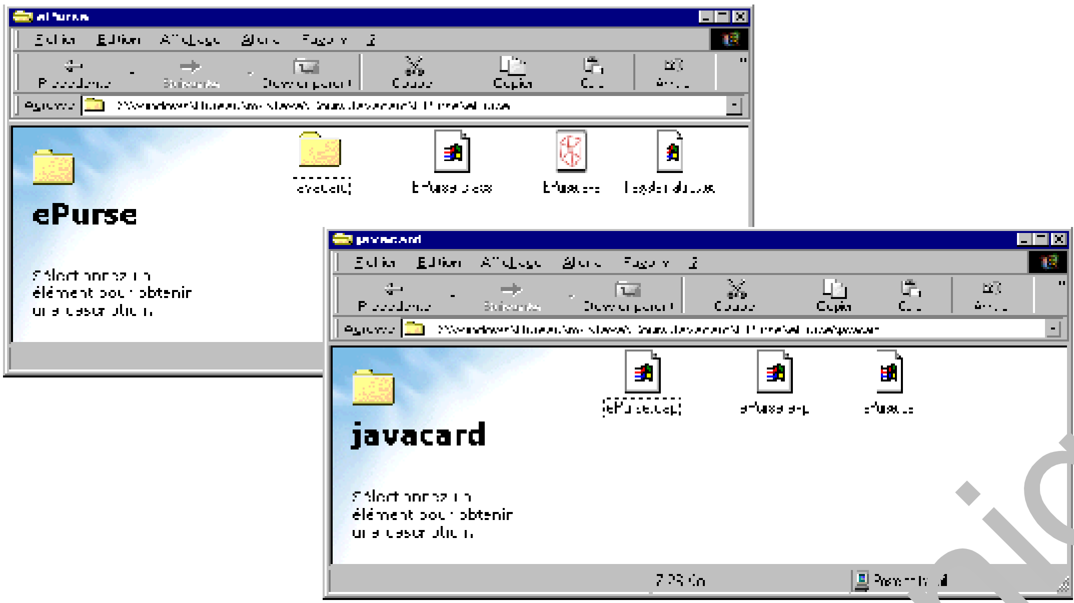
```
>converter -config epurse.opt

Java Card 2.2 Class File Converter (version 1.1)
Copyright (c) 2000 Sun Microsystems, Inc. All rights reserved.


conversion completed with 0 errors and 0 warnings.

>
```

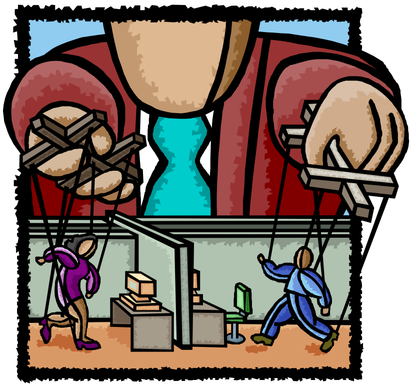
Conversion result



Java Card simulation

- The Toolkit provides tools for simulation
  - **`jcwde`** : **J**ava**C**ard **W**orkstation **D**evelopment **E**nvironment
    - Which simulates the Java Card
  - **`apdutool`** :
    - Which simulates the Java Card reader

The jcwde tool



- The simulator
  - Uses an applet called "Installer" to download and install another applet
  - Listens on a socket for incoming APDU
    - Port 9025 par défaut
  - Manages the protocol
    - Throws an exception in case of trouble

Running the jcwde tool

- This is done using a configuration file which indicates
  - The applet Installer AID
  - The AID of the applet to be downloaded

```
>jcwde -p 9025 jcwde.app
Java Card 2.2 Workstation Development Environment (version 1.1).
Copyright (c) 2000 Sun Microsystems, Inc. All rights reserved.
jcwde is listening for T=0 Apdu's on TCP/IP port 9á025.
```

File `jcwde.app content`

```
// applet                                          AID
com.sun.javacard.installer.InstallerApplet
               0xa0:0x00:0x00:0x00:0x62:0x03:0x01:0x08:0x01
ePurse.EPurse
               0xa0:0x00:0x00:0x00:0x62:0x03:0x01:0x0c:0x03:0x01
```


`apdutool`

- This tool reads a script which contains APDU and sends the APDU to the card
  - In fact to the **`jcwde simulator`**
- It displays the result in hexadecimal
  - On the standard output
  - Or in a specified file when using the option **`-o`**

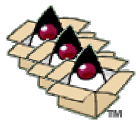
Script example

```
powerup;
// Select the installer applet
0x00 0xA4 0x04 0x00 0x09 0xa0 0x00 0x00 0x00 0x62 0x03 0x01 0x08 0x01 0x7F;
// begin installer command
0x80 0xB0 0x00 0x00 0x00 0x7F;
// create EPurse
0x80 0xB8 0x00 0x00 0x0c 0x0a 0xa0 0x00 0x00 0x00 0x62 0x03 0x01 0x0c 0x03 0x1
0x00 0x7F;
// end installer command
0x80 0xBA 0x00 0x00 0x00 0x7F;
// Select EPurse
0x00 0xa4 0x04 0x00 0x0a 0xa0 0x00 0x00 0x00 0x62 0x03 0x01 0x0c 0x03 0x1 0x7F;
powerdown;
```


Output example

```
>apdutool Essai01.scr
Java Card 2.2 ApduTool (version 1.1)
Copyright (c) 2000 Sun Microsystems, Inc. All rights reserved.
Opening connection to localhost on port 9á025.
Connected.
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 09, a0, 00, 00, 00, 62, 03, 01, 08, 01, Le
: 00, SW1: 90, SW2: 00
CLA: 80, INS: b0, P1: 00, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b8, P1: 00, P2: 00, Lc: 0c, 0a, a0, 00, 00, 00, 62, 03, 01, 0c, 03
, 01, 00, Le: 0a, a0, 00, 00, 00, 62, 03, 01, 0c, 03, 01, SW1: 90, SW2: 00
CLA: 80, INS: ba, P1: 00, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 0a, a0, 00, 00, 00, 62, 03, 01, 0c, 03, 01
, Le: 00, SW1: 90, SW2: 00
>
```

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

OwnerPIN

- This class helps the developer to protect the access to some features of the smart card using a PIN code

```
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);
}
```

OwnerPIN

- The CAD must validate the PIN code prior to access the other features

```
case EPURSE_ADD:
        apdu.setIncomingAndReceive();
        if(!pinCode.isValidated())
                ISOException. throwIt(
                        ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
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;
```


OwnerPIN

- The OwnerPIN proposes a method to unblock a blocked PIN code (after a TRY_LIMIT unsuccessful attempts)

```
case EPURSE_UNBLOCK:
        pinCode.resetAndUnblock();
```

OwnerPIN

- The OwnerPIN proposes a method to reset the `validated` flag

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


Cryptography

- An entire package is dedicated to cryptography
  - Including creation and management of keys
    - Public and private
  - Using several algorithms
    - AES, DES, DES3, RSA
- Two main packages
  - **`javacard.framework.security`**
  - **`javacardx.crypto`**

Cryptography

Class Summary	
Interfaces	
AESKey	AESKey contains a 16/24/32 byte key fo algorithm.
DESKey	DESKey contains an 8/16/24 byte key fo operations.
DSAKey	The DSAKey interface is the base interfa key implementations.
DSAPrivateKey	The DSAPrivateKey interface is used
DSAPublicKey	The DSAPublicKey interface is used t DSA algorithm.
ECKey	The ECKey interface is the base interface implementations.
ECPrivateKey	The ECPrivateKey interface is used t ECDSA (Elliptic Curve Digital Signature using the ECDH (Elliptic Curve Diffie-H
ECPublicKey	The ECPublicKey interface is used to ECDSA algorithm and to generate shared
Key	The Key interface is the base interface fo
PrivateKey	The PrivateKey interface is the base i algorithms.
PublicKey	The PublicKey interface is the base in algorithms.

Class Summary	
RSAPrivateCrtKey	The RSAPrivateCrtKey interface is used to sign data using the RSA algorithm in its Chinese Remainder Theorem form.
RSAPrivateKey	The RSAPrivateKey class is used to sign data using the RSA algorithm in its modulus/exponent form.
RSAPublicKey	The RSAPublicKey is used to verify signatures on signed data using the RSA algorithm.
SecretKey	The SecretKey class is the base interface for keys used in symmetric algorithms (e. g. DES).
Classes	
Checksum	The Checksum class is the base class for CRC (cyclic redundancy check) checksum algorithms.
KeyAgreement	The KeyAgreement class is the base class for key agreement algorithms such as Diffie-Hellman and EC Diffie-Hellman [IEEE P1363].
KeyBuilder	The KeyBuilder class is a key object factory.
KeyPair	This class is a container for a key pair (a public key and a private key).
MessageDigest	The MessageDigest class is the base class for hashing algorithms.
RandomData	The RandomData abstract class is the base class for random number generatio
Signature	The Signature class is the base class for Signature algorithms.
Exceptions	
CryptoException	CryptoException represents a cryptography-related exce

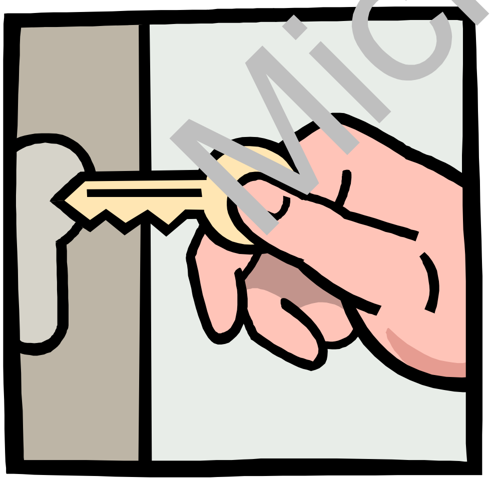
Conclusion

- In this chapter, we have seen
  – The various operating systems available for the smart cards
  – An introduction to the Java Card system
  – How the Applet are working
  – Some classes and methods provided by the Java Card API

Security

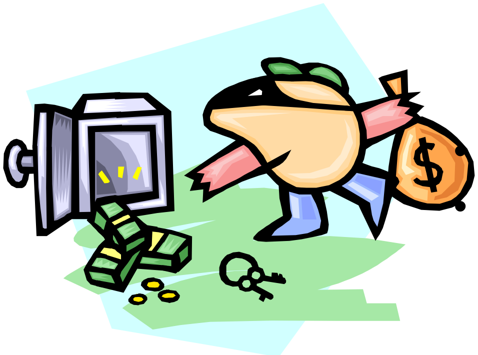
Hardware and software aspects

Objectives



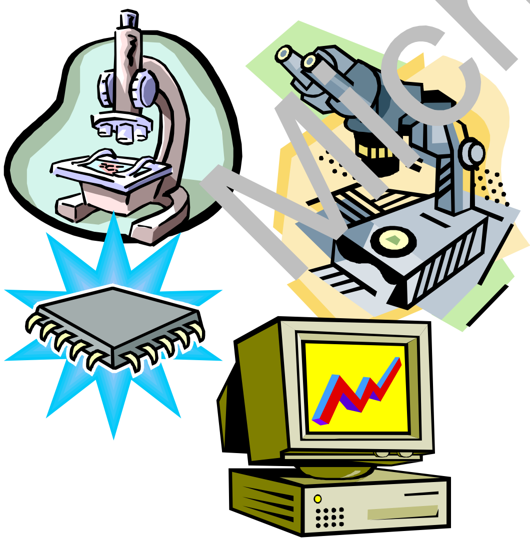
- In this chapter, we'll see
  - An introduction about the security aspects of the smart cards
    - From a hardware point of view
    - From a software point of view

Hardware security



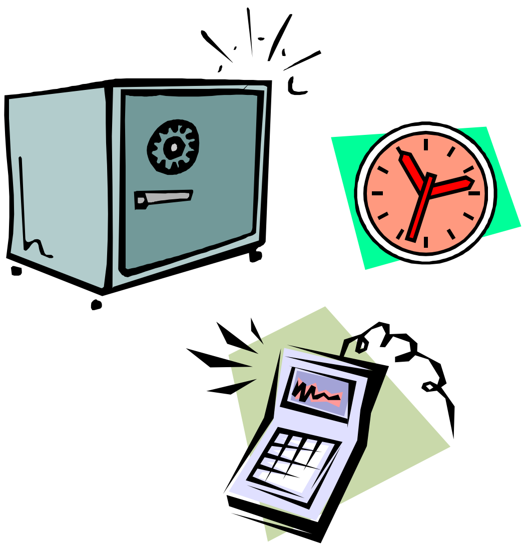
- A smart card contains important data
  - It could contain money
    - Electronic purses
- It must be tamper resistant
- "*If you know the attack you can build the shield*"

The attacks



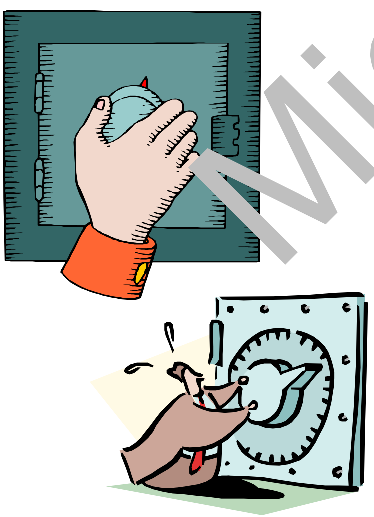
- X raying the micro-chip
- Measuring the power consumption variation during critical APDU
  - When the PIN code is transmitted for example
- Measuring the answer delay
  - To try to predict what branches in the program are completed

The shields



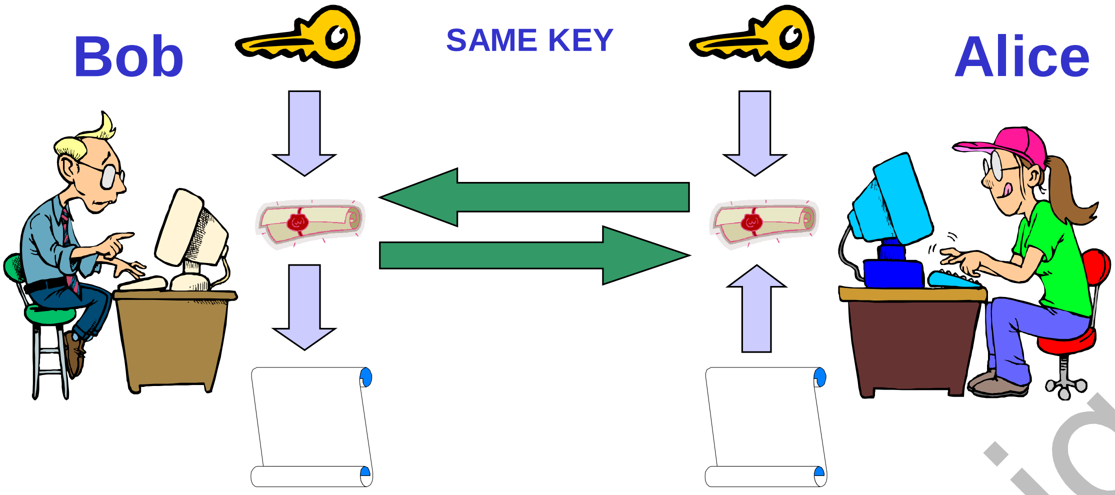
- The micro-chip uses an internal shield to protect itself against an X-Ray scanning
- It guarantees the same delay for both branches of an alternative statement
- It guarantees the same power consumption in all cases

Software attacks and shields



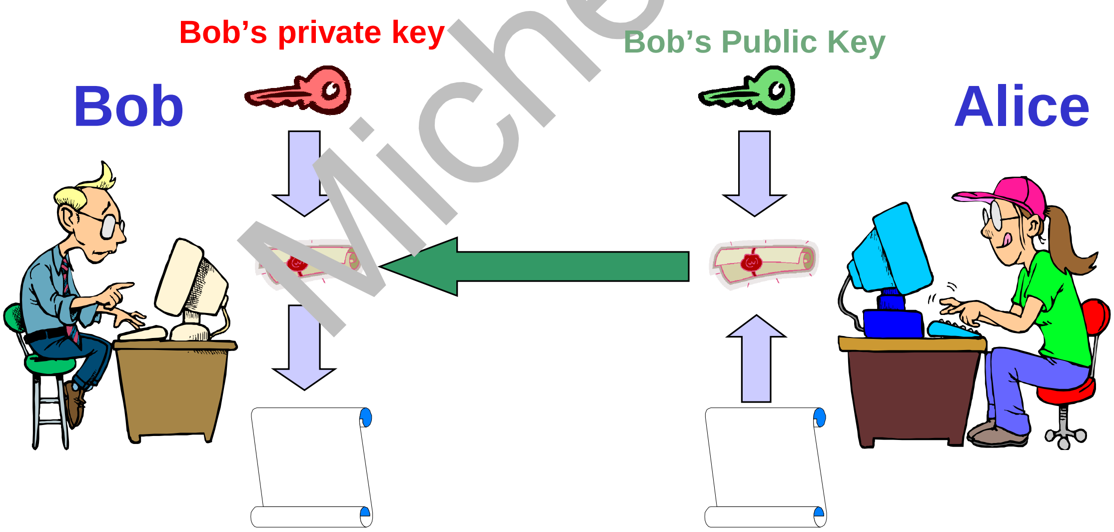
- Data are protected using cryptography
  - Various techniques
    - DES, DES3
    - RSA
    - SHA
- Cryptography is based on
  - A public algorithm
  - A key
    - Private (DES, DES3)
    - Public (RSA)

Symmetric Enciphering



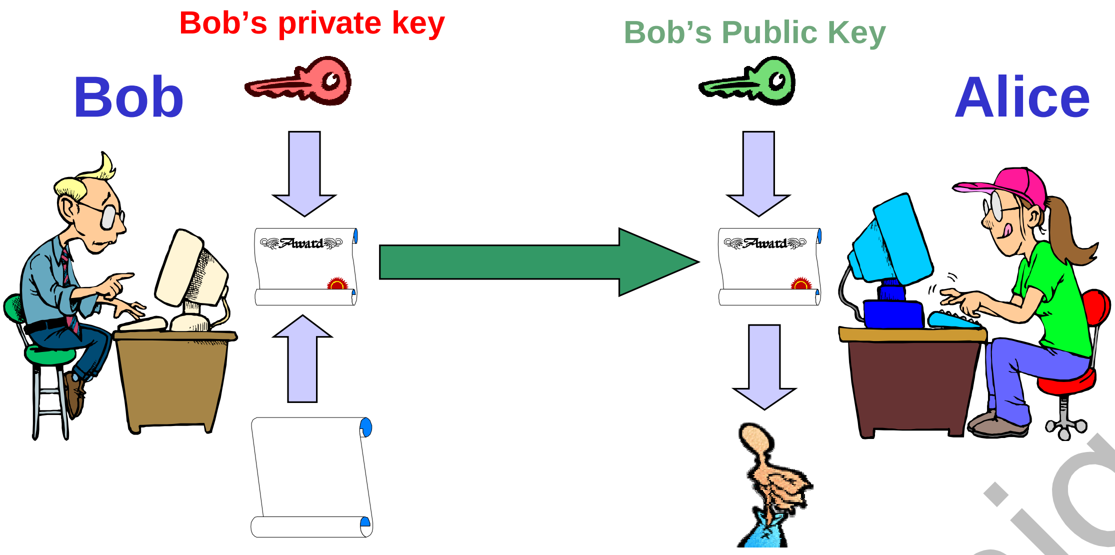
Michel Koenig Smart cards tutorial 69

Asymmetric enciphering



Michel Koenig Smart cards tutorial 70

Signing using asymmetric keys

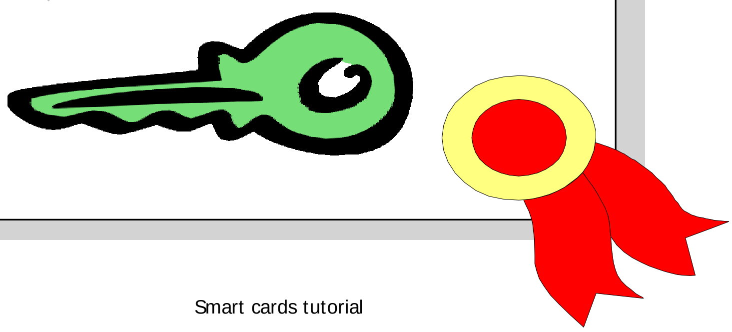


Michel Koenig Smart cards tutorial 71

Certify public key

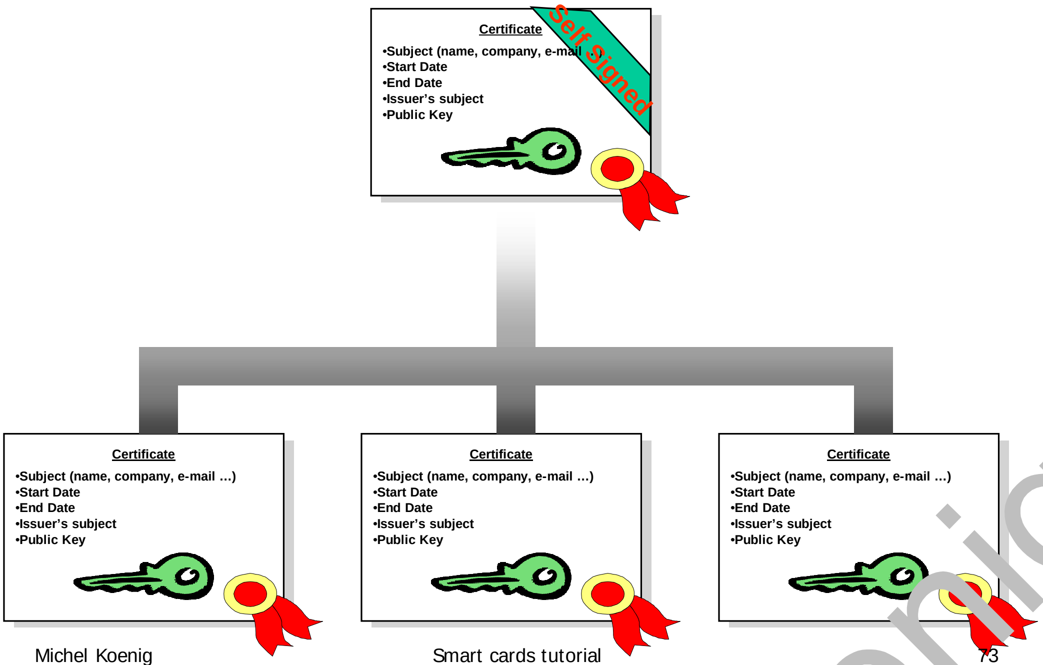
X509 Certificate

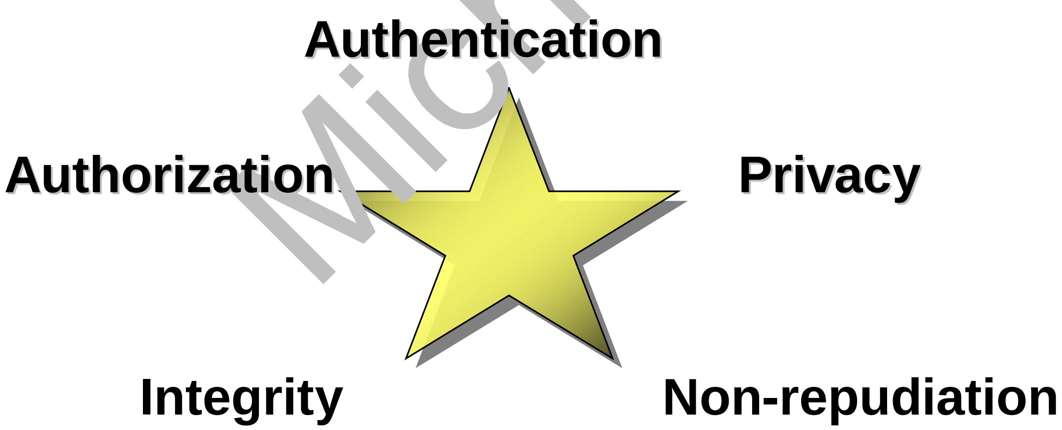
- **Subject (name, company, e-mail …)**
- **Start Date**
- **End Date**
- **Issuer's subject**
- **Public Key**



Michel Koenig Smart cards tutorial 72

Certification Authority





Protect private key With Smart cards

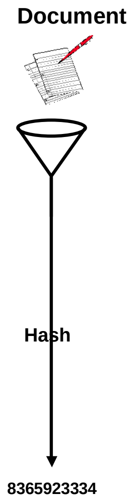
- The Private key born, live and die inside the card
  - Key pair generation
  - Secure access
  - Cryptographic algorithm process inside the card

- 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)

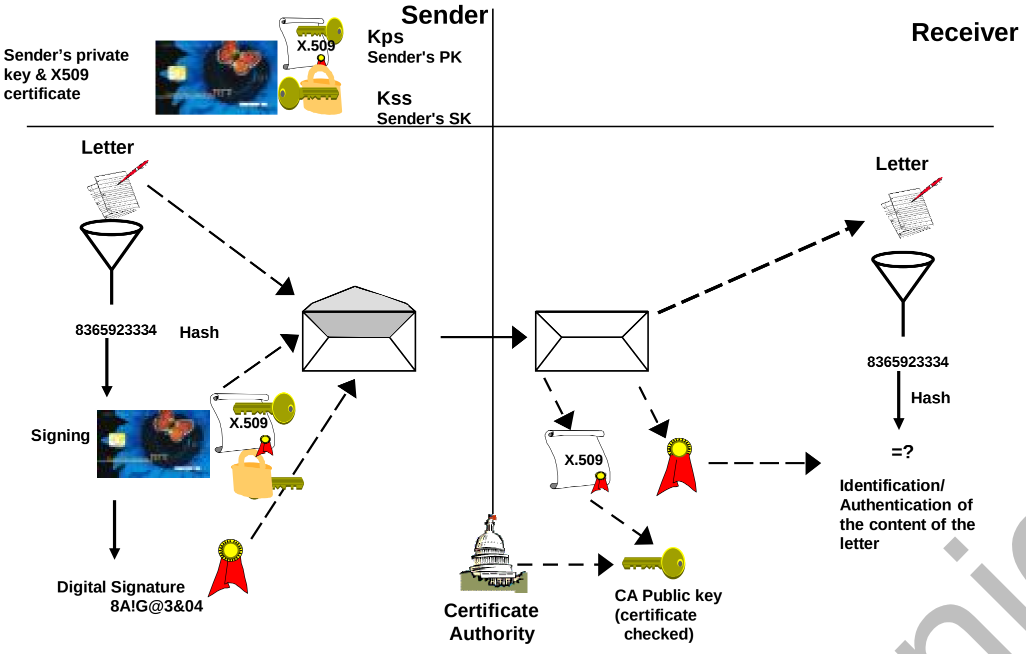


• Modifying one bit completely changes the Hash

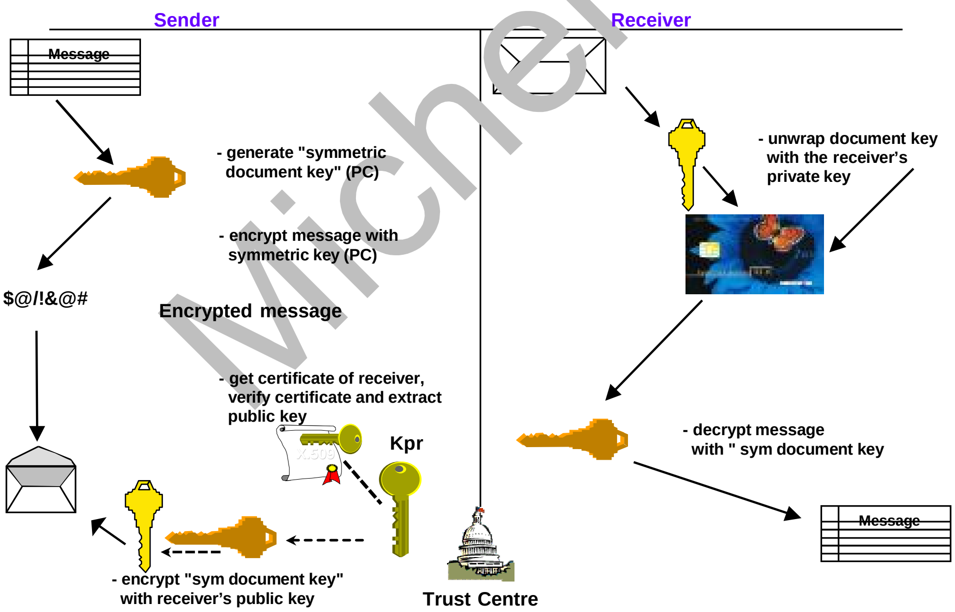
•Hash result is completely unpredicatble

• Usual algorithms are MD5 (used for linux Password storage) or SHA-1

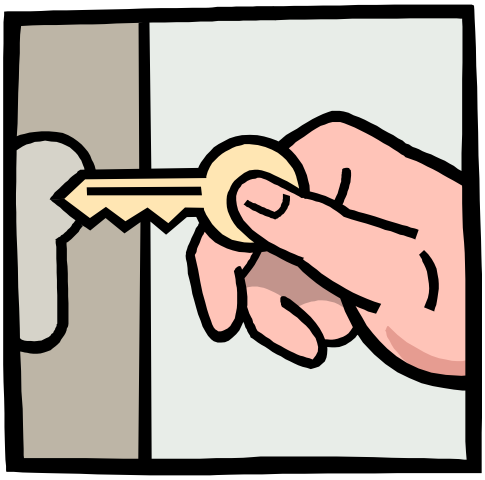
Digital Signature (E mail)



S/MIME Encryption



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

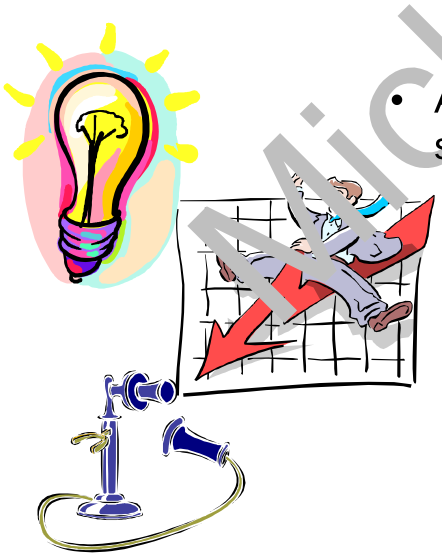
Client side programming

ISO7816-3 protocol

Objectives

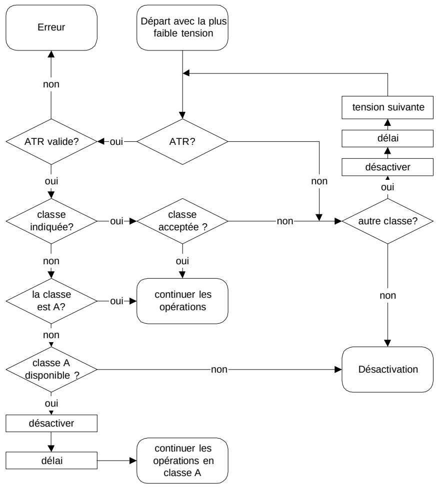
- In this chapter, we'll see
  - The ISO7816 protocol between the card reader and the smart card
  - The protocols between the smart card reader and the PC
    - PC/SC
    - TLP224
    - GBP
    - Open Card

Reader-smart card protocol



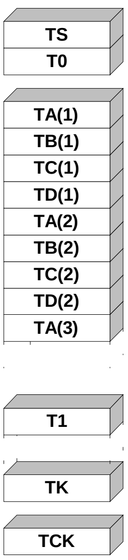
- According to the manufacturer, smart card can support
  - Two different power voltages
    - 3V and 5V
  - Two different ways to transmit characters
    - Direct or inverted
  - Two different protocols
    - T0, T1

Detecting the voltage



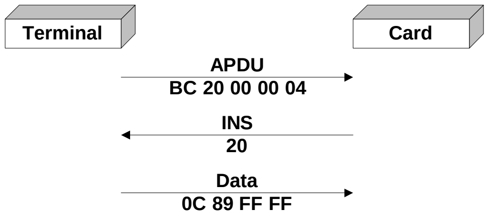
- A complex algorithm is needed to detect the voltage of the smart card
  - Try 3V
    - If ATR detected
      - Is it good?
      - …

Interpreting the ATR



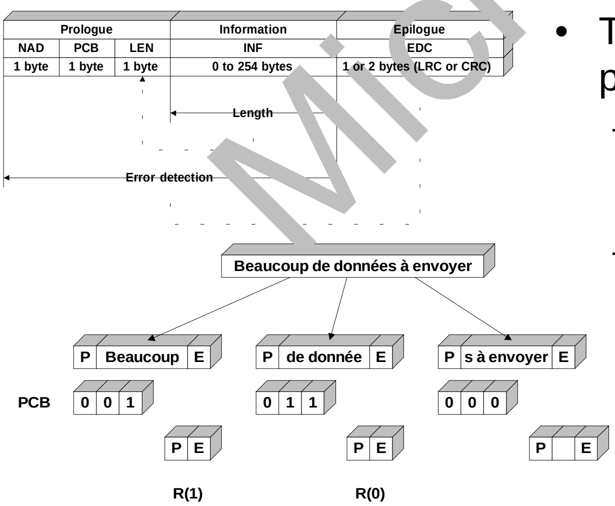
- The smart card returns to the reader an **A**nswer **to** **R**eset message
- This ATR indicates
  - The data coding convention
    - Direct or inverted
  - The protocol
    - T0 or T1
  - Historical bytes
    - Various data
      - Manufacturer for instance

T0 protocol



- T0 is a character oriented protocol
  - One character is transmitted after the other
  - Acknowledgement, if needed, is done after the transmission of the 5 bytes of the APDU
- T0 limits the length of the data transmitted
  - 32 bytes
  - Possibility to chain APDUs

T1 protocol



- T1 is a block oriented protocol
  - The entire APDU, including the extra data, is transmitted all at once
  - Possibility to have sequences of messages for long data

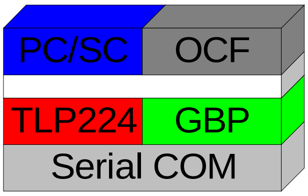
Classes of APDU and Commands

- It is virtually possible to use any value for **CLA** and **INS** in an APDU
- Nevertheless, some values are reserved by ISO7816
  - The constant **`CLA_ISO7816`** is the value of **CLA** reserved by ISO
- The next page displays some values reserved for **INS**

Standard ISO commands

Value	Command	Value	Command
0x0E	Erase binary	0xC0	Get response
0x20	Verify	0xC2	Envelope
0x70	Manage channel	0xCA	Get data
0x82	External authenticate	0xD0	Write binary
0x84	Get challenge	0xD2	Write record
0x88	Internal authenticate	0xD6	Update binary
0xA4	Select file	0xDA	Put data
0xB0	Read binary	0xDC	Update record
0xB2	Read record	0xE2	Append record

PC – Reader protocols



- Most of the smart card readers are connected to a PC through a serial link
  - A new generation use a USB link
- The most common protocols used are
  - TLP224
    - Characters oriented
  - GBP
    - Blocks oriented
- Microsoft had introduced recently a new protocol: PC/SC
- Open Card had introduced a protocol based on Java: OCF

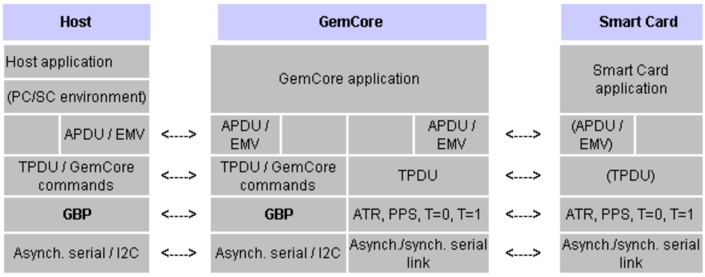
TLP224

ACK	LEN	Message	LRC
0x60	0x01	0x4D	0x2C

0x36	0x30	0x30	0x31	0x34	0x44	0x32	0x43	0x03

- Introduced by Bull CP8
- Encode commands to the smart card reader
  - Power on
  - Power off
  - Send APDU
  - Resend message
    - Error
- The message is encapsulated between
  - ACK (0x60)
  - LN length of message
  - LRC
- Bytes are splitted in quartets and encoded in ASCII

GBP



- Introduced by GemPlus
- Similar to the protocol T=1
  - Simplified
- It is a transport layer which allows the PC to send commands to the reader

PC/SC

To...	Call...
Retrieve the identifier (GUID) of the primary service provider for a given card.	SCardGetProviderId
Retrieve a list of cards previously introduced to the system by a specific user.	SCardListCards
Retrieve the identifiers (GUIDs) of interfaces supported by a given card.	SCardListInterfaces
Retrieve a list of reader groups that have previously been introduced to the system.	SCardListReaderGroups
Retrieve the list of readers within a set of named reader groups.	SCardListReaders

To...	Call...
Connect to a card.	SCardConnect
Reestablish a connection.	SCardReconnect
Terminate a connection.	SCardDisconnect
Start a *transaction*, blocking other applications from accessing a card.	SCardBeginTransaction
End a transaction, allowing other applications to access a card.	SCardEndTransaction
Provide the current status of the reader.	SCardStatus
Requests service and receives data back from a card using *T=0*, *T=1*, and raw protocols.	SCardTransmit

- **PC** to **S**mart **C**ards
- Introduced by Microsoft
  - Helped by smart cards manufacturers

OpenCard

- Offer a portable plateform to develop client-side application
- This application could work
  - With Java Cards
  - Or with other cards
- The technique used is based on the use of a proxy
  - Based on the Remote Method Invocation technique
  - We'll see more about OCF in the next chapter

Conclusion

- In this chapter, we have seen
  - The ISO7816 protocol between the card reader and the smart card
  - The protocols between the smart card reader and the PC
    - PC/SC
    - TLP224
    - GBP
    - Open Card

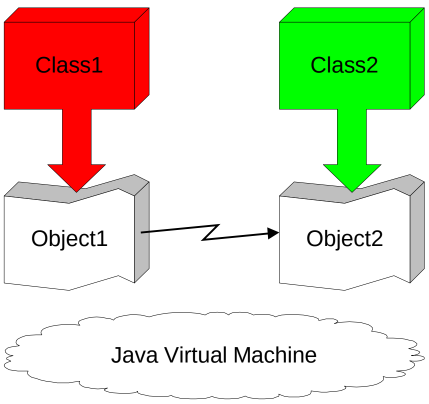
Remote Method Invocation

Principles and programming techniques

Objectives

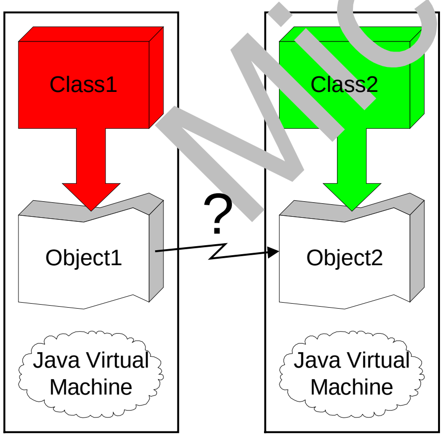
- In this chapter, we'll see
  - What is the Remote Method Invocation technique
  - How this technique was introduced for smart card programming
  - How to develop services using the Open Card Framework

Principles of RMI - intro

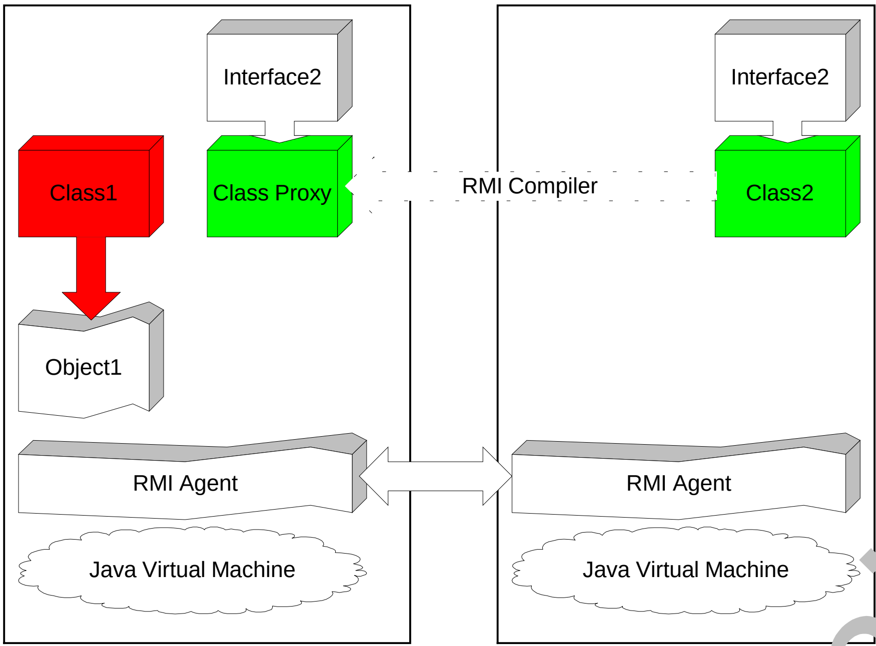


- In a Java program located in a single machine
  - Classes and objects lay in the same memory storage
  - All of them are powered by the same Java Virtual Machine
- During the call of a method the control is passed from one object to the other
  - The JVM does the job

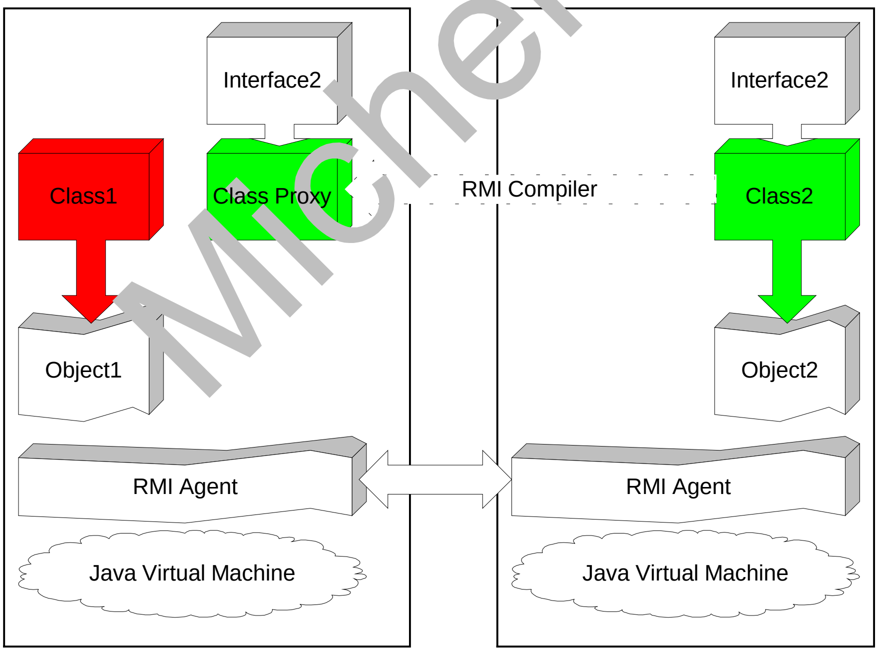
Principles of RMI - intro



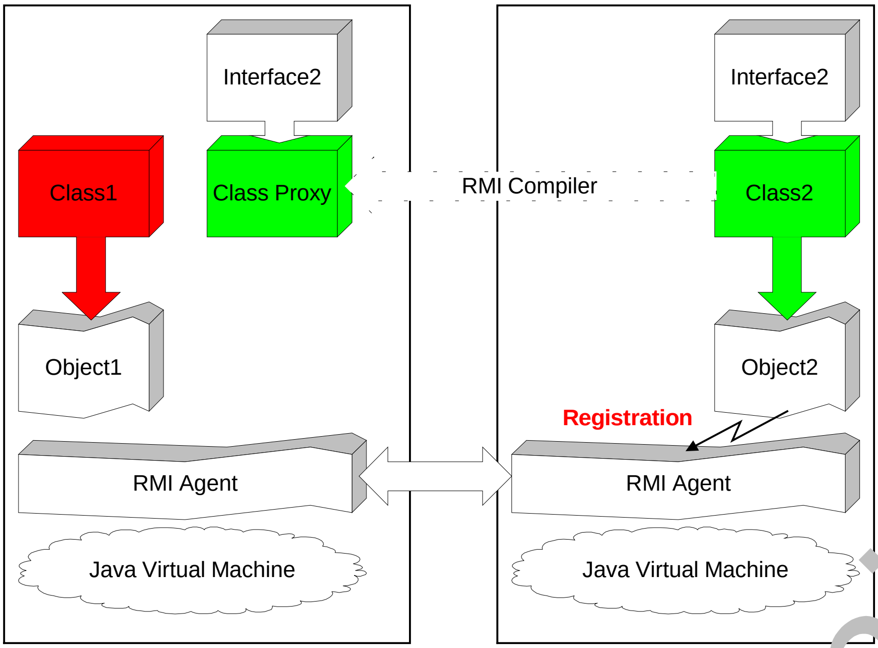
- In the case of a distributed program
  - Classes and objects do not lay in the same memory storage
  - They are powered by two different Java Virtual Machines
- The direct call of one method of **`Object2`** by **`Object1`** is no longer possible

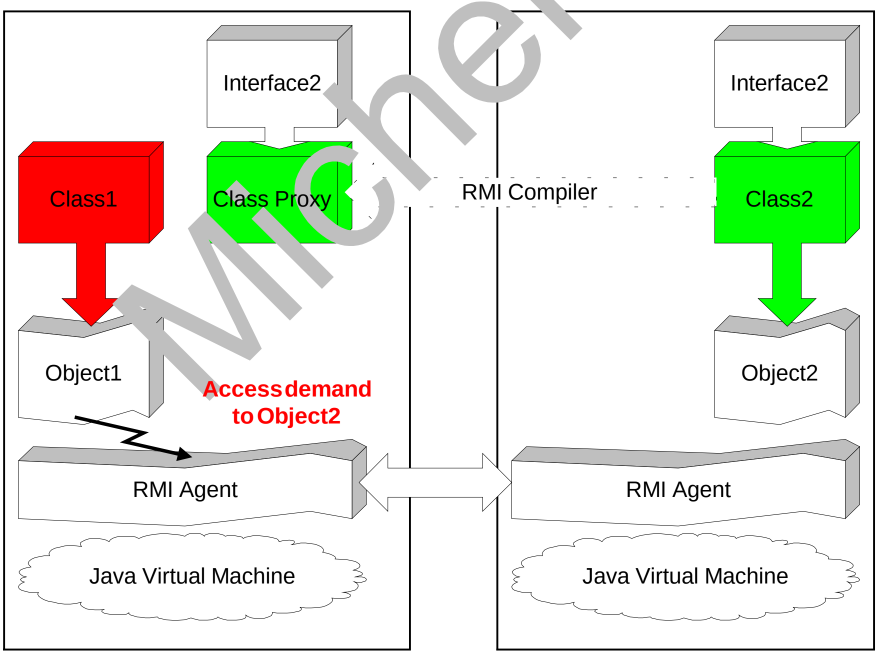


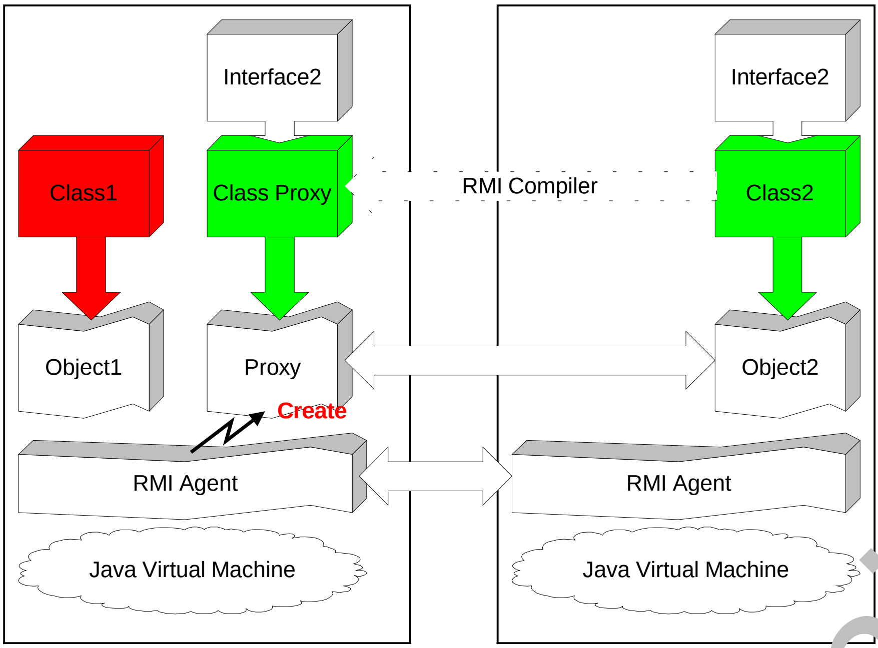
Michel Koenig Smart cards tutorial 99



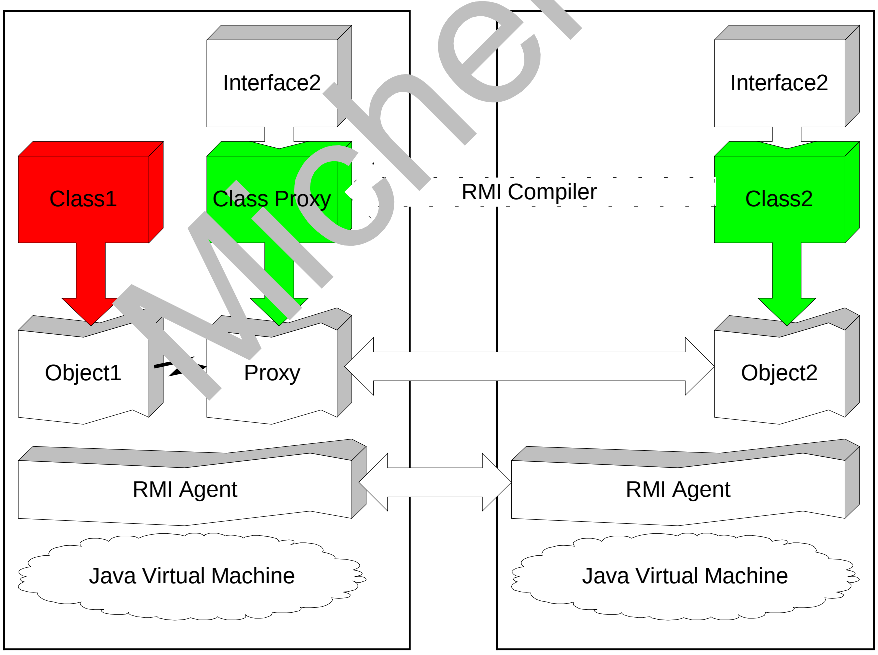
Michel Koenig Smart cards tutorial 100


Interface2
Class1
Class Proxy
RMI Compiler
Interface2
Class2
Object1
Object2
Registration
RMI Agent
RMI Agent
Java Virtual Machine
Java Virtual Machine


Interface2
Class1
Class Proxy
RMI Compiler
Interface2
Class2
Object1
Object2
Access demand
to Object2
RMI Agent
RMI Agent
Java Virtual Machine
Java Virtual Machine

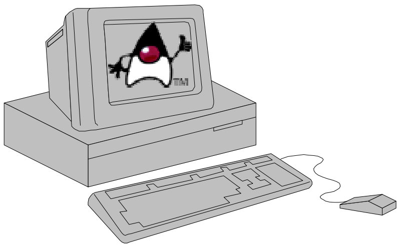


Michel Koenig Smart cards tutorial 103



Michel Koenig Smart cards tutorial 104

RMI for the Java Card



- The application classes reuse the same production environment as in Java Standard case
  - Use base class CardRemoteObject for implementation
  - Use base interface Remote for remote interface

Extra software needed

- Java Card RMI needs extra software to work
  - One part on the card itself
  - One part on the terminal
    - Open Card Framework was choosen by Sun as a reference implementation

Example: Applet-side interface Purse

No difference with standard Java RMI

```
package com.sun.javacard.samples.RMIDemo;

import java.rmi.*;
import javacard.framework.*;

public interface Purse extends Remote{

    public short getBalance() throws RemoteException;
    public void debit(short m) throws RemoteException, UserException;
    public void credit(short m) throws RemoteException, UserException;

}
```


Example: Applet-side (1) implementation EPurseImpl

This is the main difference with standard Java RMI

```
package com.sun.javacard.samples.RMIDemo;

import javacard.framework.UserException;
import javacard.framework.Util;
import javacard.framework.service.CardRemoteObject;
import java.rmi.RemoteException;

public class PurseImpl extends CardRemoteObject implements Purse {

    private short balance = 0;

    public PurseImpl() {
        super();                    // export it
    }
```

Example: Applet-side (2)
implementation EPurseImpl

```
 public void credit(short m) throws RemoteException, UserException {
        if(m<=0) UserException.throwIt(BAD_ARGUMENT);
        balance +=m;
    }
    public void debit(short m) throws RemoteException, UserException {
        if(m<=0) UserException.throwIt(BAD_ARGUMENT);
        balance -=m;
    }
    public short getBalance() throws RemoteException {
        return balance;
    }
}
```


Example: Applet-side (1)
installing applet

```
package com.sun.javacard.samples.RMIDemo;

import java.rmi.*;
import javacard.framework.APDU;
import javacard.framework.ISOException;
import javacard.framework.UserException;
import javacard.framework.Util;
import javacard.framework.service.*;

public class PurseApplet extends javacard.framework.Applet {
    private Dispatcher disp;
    private RemoteService serv;
    private Remote purse;
```

A dispatcher glues together all the services and dispatches APDU to services

A service knows how to process all incoming APDU

Example: Applet-side (2) installing applet

```
public PurseApplet() {
        purse = new PurseImpl();
        disp = new Dispatcher( (short) 1);
        serv = new RMIService(purse);
        disp.addService(serv, Dispatcher.PROCESS_COMMAND);
        register();
    }
    public static void install(byte[] aid, short s, byte b) {
        new PurseApplet();
    }
    public void process(APDU apdu) throws ISOException {
        disp.process(apdu);
    }
}
```

Only one service will be created and attached to the dispatcher

Add the service which was just created as a command processor

Delegate the process of the apdu to the dispatcher (then to the service)

Example: client-side (1) code: PurseClient

```
import opencard.core.service.*;
import examples.purse.*;
import com.sun.javacard.javax.smartcard.rmiclient.*;
import com.sun.javacard.ocfrmiclientimpl.*;
import javacard.framework.UserException;

public class PurseClient extends java.lang.Object {
    /** Creates new PurseClient */
    public PurseClient() {
    }

    public static void main(java.lang.String[] argv) {
        // arg[0] constains the debit amount
        short debitAmount = (short) Integer.parseInt(argv[0]);
```

Suppose that the value to be debited is in the first arg of the main

Example: client-side (2) code: PurseClient

Start the Open Card Framework

This class could be considered as the driver of the card

```
try {
    // initialize OCF
    SmartCard.start();

    // wait for a smartcard
    CardRequest cr = new CardRequest (CardRequest.NEWCARD,
                                      null,OCFCardAccessor.class);
    SmartCard myCard = SmartCard.waitForCard ( cr );

    // obtain a Java Card RMI Card Accessor CardService
    CardAccessor myCS = (CardAccessor)
                   myCard.getCardService(OCFCardAccessor.class, true);

    // create a Java Card RMI connector instance
    JavaCardRMIConnect jcRMI = new JavaCardRMIConnect( myCS );
```


Example: client-side (3) code: PurseClient

Get the remote reference of the Purse (a refernce to the proxy)

```
// select the Java Card applet
byte[] appAID = new byte[] {0x01,0x02,0x03,0x04,0x05,0x06,0x07, 0x08};
jcRMI.selectApplet( appAID );

// obtain the initial reference to the Purse interface
Purse myPurse = (Purse) jcRMI.getInitialReference();

// debit the requested amount
try {
    short balance = myPurse.debit ( debitAmount );
}catch ( UserException jce ) {
    short reasonCode = jce.getReason();
    // process UserException reason information
}
```

Example: client-side (4)
code: PurseClient

```
            // display the balance to user
        }catch (Exception e){
            e.printStackTrace();
        } finally {
            try{
                SmartCard.shutdown();
            }catch (Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

Michel Koenig Smart cards tutorial 115

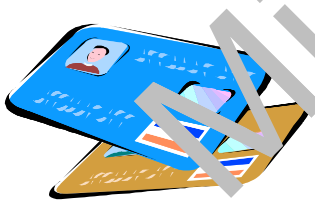
Conclusion

- In this chapter, we have seen
  - What is the Remote Method Invocation technique
  - How this technique was introduced for smart card programming
  - How to develop services using Java Card RMI and the Open Card Framework

Michel Koenig Smart cards tutorial 116

Conclusion

Conclusion



- In this presentation, we have
  - Introduced the concepts and the technology of the smart cards
  - Described how to program the Java Cards
  - Explored the tools and the environments provided by the manufacturers to develop solutions with smart cards

Appendix 1

SIM cards

SIM Cards

Proactive SIM cards

Objectives

- In this chapter, we'll see
  – The standards around the SIM card
  – What is a proactive SIM card
  – How works a proactive SIM card

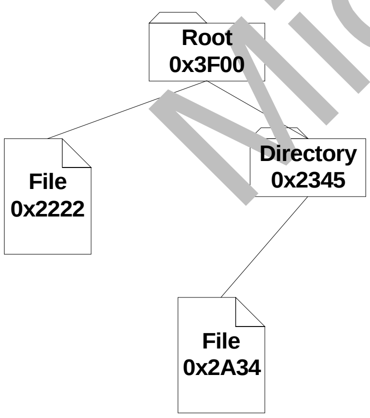
SIM cards

- Standardized by ETSI for GSM
- GSM 11.11 V6.1.0
  – SIM specs
    - **S**ubscriber **I**dentification **M**odule
- GSM 11.14 V7.1.0
  – SIM Toolkit specs
- GSM 03.19 V1.0.0
  – Javacard SIM API

Proactives SIM

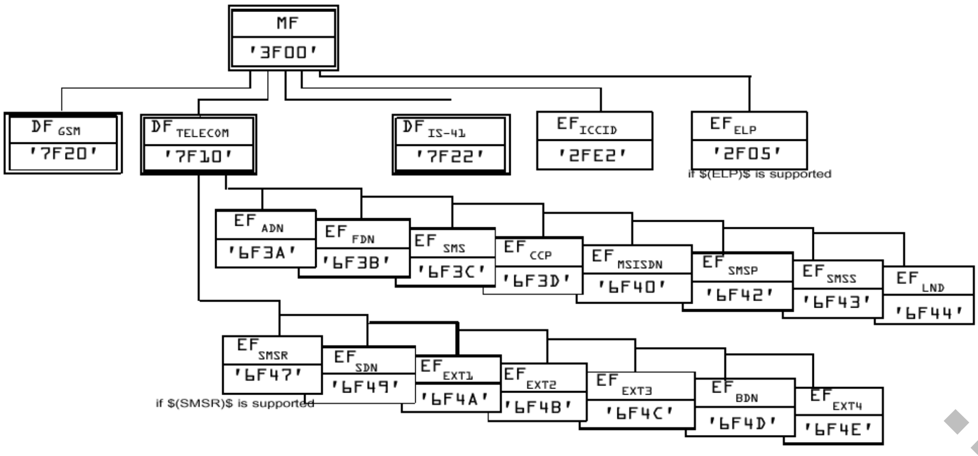
- Using the SIM Toolkit, possibility to
  - Program the SIM
  - Make the SIM card application driving the phone
    - Access to keyboard, display, …

Internal organization of the SIM

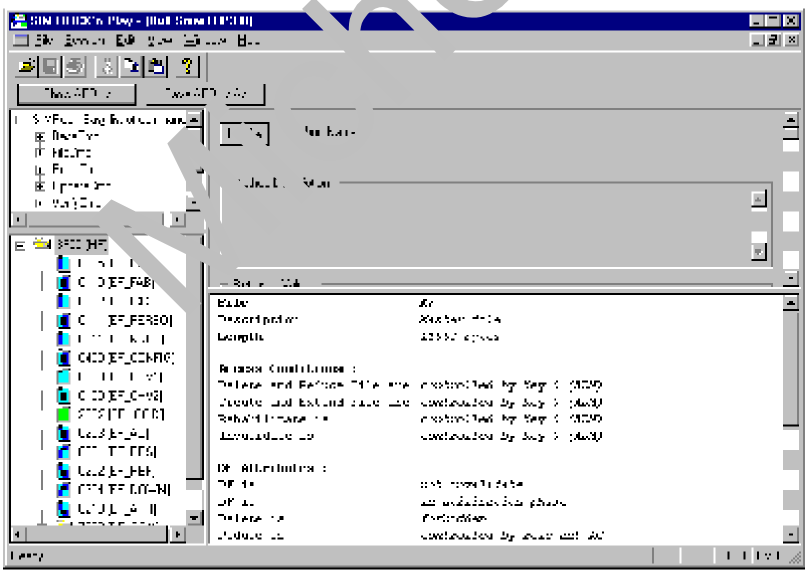


- The SIM contains a certain number of "files" grouped into "directories"
- Terminology:
  - Element File: file
  - Dedicated file: directory

File hierarchy



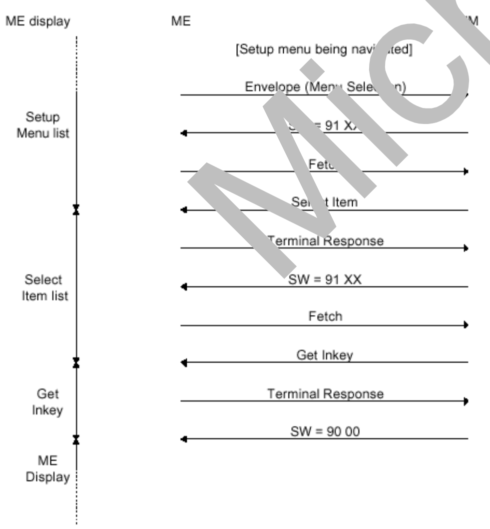
File hierarchy



Proactive SIM

- The ISO7816 standard does not permit that the card starts talking first
  - A card is waiting for an APDU and responds whenit 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



- The Mobile Equipment (the phone) send a command (**Envelope**)
  - Containing the menu selection
- The card answers using the status word **91xx**
  - Xx is the length of the command that the SIM wants to send back to the ME
- The ME sends the command **Fetch** to get the command from the SIM
- …

Allowed commands for the SIM

```
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)
```

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

Conclusion

- In this chapter, we have seen
  - The standards around the SIM card
  - What is a proactive SIM card
  - How works a proactive SIM card

Appendix 2

Logical channels

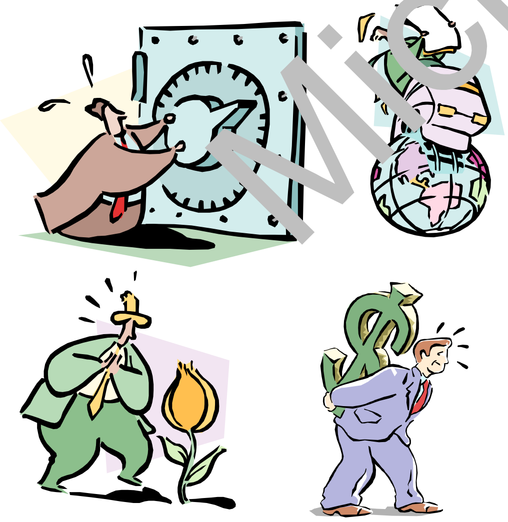
Objectives of this appendix

- In this appendix, we'll see
  - What is a channel, and what it is for
  - The standard ISO7816-4 about channels
  - How Java Card 2.2 takes in account this standard
  - An example how to use the channels

What is a channel

- Within the smart card, several applets could be installed at the same time
  - An e-purse
  - A loyalty program
  - A credit-debit application
  - An e-ticket application
- For what we have seen, only **one** can be selected at once

What is a channel (continued)



- **Example** : buying a transportation ticket using the epurse and getting loyalty points
- **Security** must be the same during all the operations
- We cannot expect the same **OwnerPIN** for the three applets

What is a channel (continued)

- It is a virtual link between the CAD and a selected element within the card
- The APDU are redirected to the corresponding element according to the channel number which is specified in the APDU

Channels in ISO7816-4

- ISO7816-4 allows up to 4 channels to be used
  – Channel 0 : default (or basic) channel
  – Channels 1, 2, 3
- Two commands are dedicated for channel handling
  – **SELECT FILE**
  – **MANAGE CHANNEL**

Selecting a channel

- Channel information can be held only with APDU starting with a **CLA** byte of the following type
  - **0x0X**, **0x8X**, **0x9X** and **0xAX**
  - The X nibble is responsible for
    - Channel encoding
      - Least significant bits
    - Secure message encoding

MANAGE CHANNEL APDU

- This command will not be processed by any applet from the card
- It is processed by the underlying operating system
- It is used to
  - Open a channel
  - Close a channel

Opening a channel

CLA	INS	P1	P2	Lc	data	Le
0x0**Q**	**0x70**	0x00	0x00	0x00		**0x01**

data	SW1	SW2
0x0**R**	**0x90**	**0x00**

- The new open channel will be **R**
- If **Q=0**, the default applet will be selected on channel **R**
- If **Q≠0**, the selected applet on channel **Q** will become the current applet selected on channel **R**

Opening a selected channel

CLA	INS	P1	P2	Lc	data	Le
0x0**Q**	**0x70**	0x00	0x0**R**	0x00		**0x00**

SW1	SW2
0x90	**0x00**

- If **Q=0**, the default applet will be selected on channel **R**
- If **Q≠0**, the selected applet on channel **Q** will become the current applet selected on channel **R**

Closing a channel

CLA	INS	P1	P2	Lc	data	Le
0x0**Q**	**0x70**	**0x80**	0x0**R**	0x00		**0x00**

SW1	SW2
0x90	**0x00**

- Channel **R** will be closed
  - Channel R must not be the basic channel

Select an applet

CLA	INS	P1	P2	Lc	data	Le
0x0**R**	**0xA4**	**0x04**	0x0**R**	length	**AID**	**0x00**

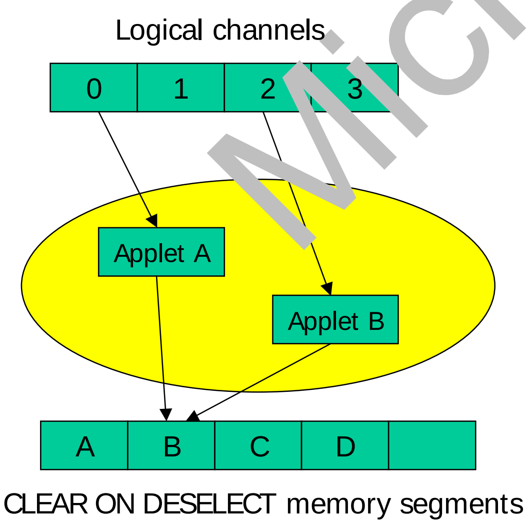
SW1	SW2
0x90	**0x00**

- Channel **R** can be any (opened or unopened) channel, including the basic channel
  - The applet identified by AID will become the selected applet in channel R
  - If channel R is not open, the command open it
  - If the channel is open, the command will change the selected applet in the channel to the one specified

Multiselectable applets

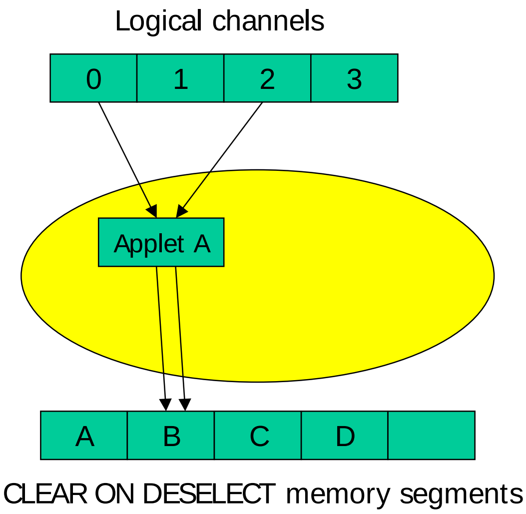
- The same applet can handle APDU on several channels
- The applet must implement the **javacard.framework.Multiselectable** interface
  - Must implement methods **select** and **deselect**
- Classpath must include **apduio.jar**

Memory usage



- If applets A and B from the same package are **multiselected**
  - They share the same **CLEAR_ON_DESELECT** memory segment
  - This segment will be clear **ONLY** when both applet will be deselected

Memory usage (continued)



- The applet can work with an internal two-fold data structure
  - One for the first channel
  - One for the second channel
- This avoid duplicating code in memory
  - Only data is duplicated

Multiselectable interface

- This interface supports two methods :
  - public boolean select(boolean allreadySelected);
    - Indicates if the applet (or one of the same package) has already been selected on different channels
    - Important to know for initialization process
  - public void deselect(boolean stillSelected);
    - Indicates if the applet (or one of the same package) remains selected on different channels
    - Important to know for CLEAR_ON_DESELECT memory usage

Multiselectable applet facts

- If one applet within a package is **Multiselectable**
  - All the applets in the same package must be selectable
- Each time an applet is proposed to be selected
  - It must take in account if other applets within the same package are selected or not
- Same thing for deselection
- During the process of an APDU
  - Only **one** applet is active

Other features

- Some methods are provided to help developer to handle correctly the channels

  static byte APDU.getCLAChannel();
- According to the selected channel, the applet can process differently the data

Example

- Suppose a wireless device able to handle several communications at the same time :
  - Phone call
  - Far Internet connection
  - Local internet connection
- This device needs to activate counters for each of the connection started and to stop counters when the connection is held

Example

- Possible usage :
- Channel 0 is dedicated to the global clock
  - To increment started counters
- Channel 1 is for the phone call connection
- Channel 2 is for the far Internet connection
- Channel 3 is for the local Internet connection

Example

- We can use a unique applet which
  - Create an array of short counters (3) on selection on channel 0
  - APDU to start, to stop or to get counters differ only on the channel number
  - APDU to increment counters is on channel 0

Conclusion

- In these appendices, we have seen
  - The SIM cards
    - Internal organization
    - The proactive commands
  - The channels
    - What is a channel, and what it is for
    - The standard ISO7816-4 about channels
    - How Java Card 2.2 takes in account this standard
    - An example how to use the channels