JavaCard development

JavaCard platform

Petr Švenda svenda@fi.muni.cz
Faculty of Informatics, Masaryk University

Centre for Research on Cryptography and Security

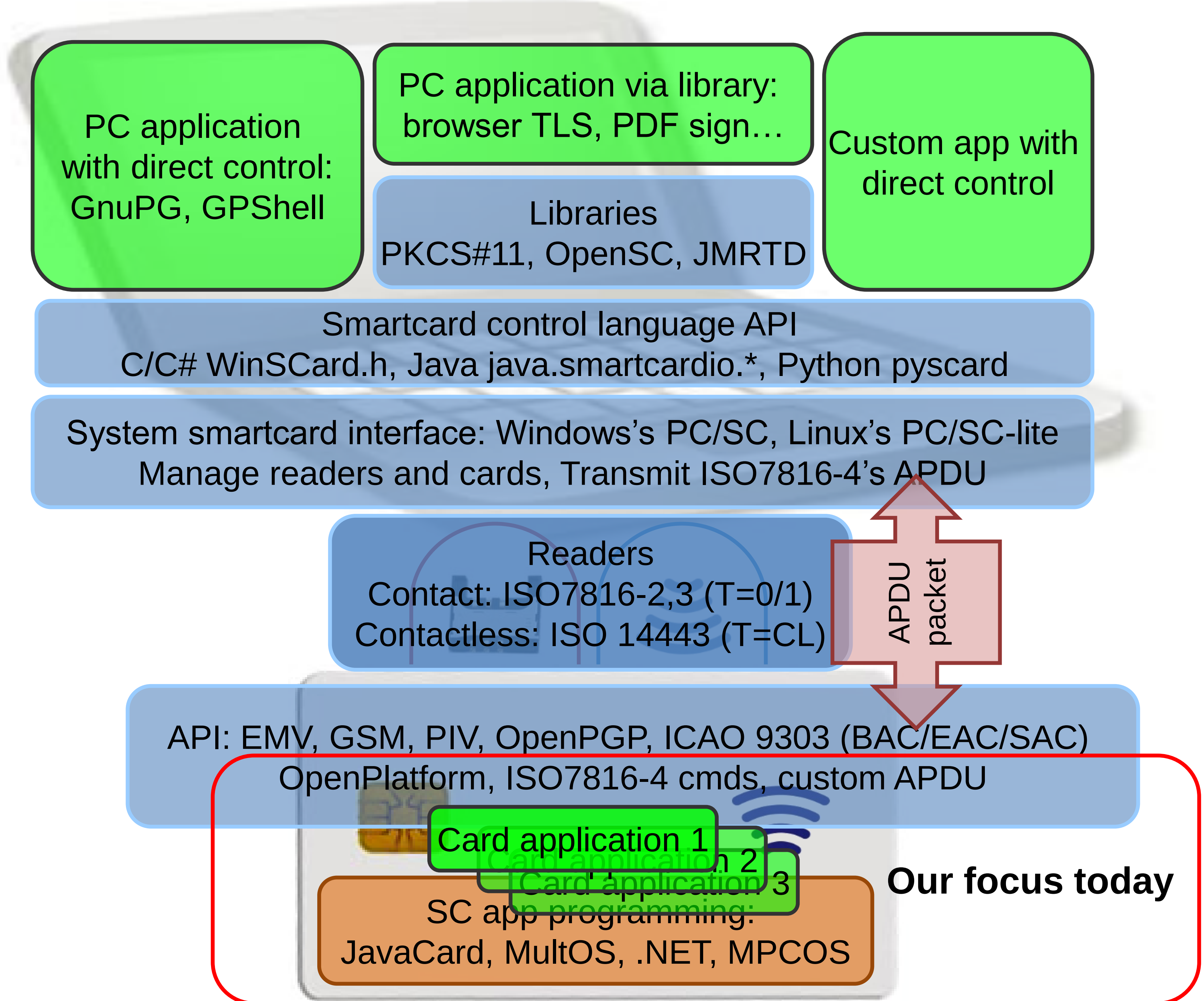
Overview

- JavaCard programming platform
- Skeleton of JavaCard applet
- How to upload and communicate with
- Best practices – performance, security

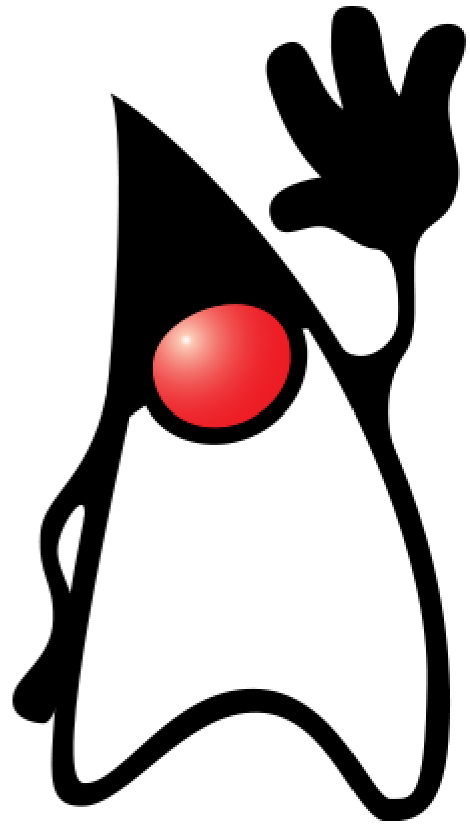
Old vs. multi-application smart cards

- One program only
- Stored persistently in ROM o EEPROM
- Written in machine code
  - Chip specific

- Multiple applications at the same time
- Stored in EEPROM
- Written in higher-level language
  - Interpreted from bytecode
  - Portable
- Application can be later managed (remotely)

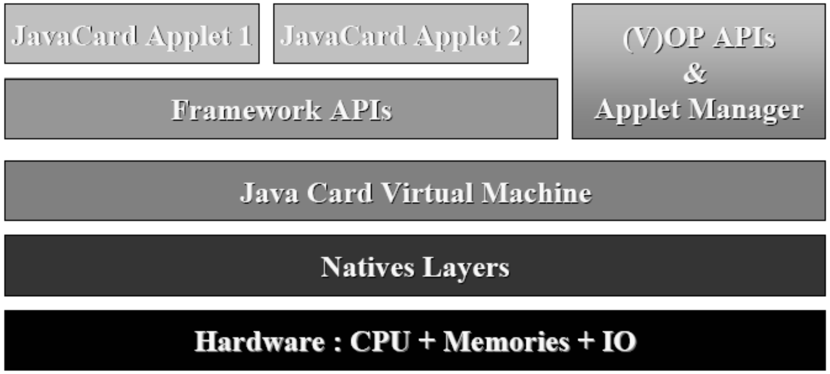

PC application with direct control: GnuPG, GPShell
PC application via library: browser TLS, PDF sign…
Custom app with direct control
Libraries
PKCS#11, OpenSC, JMRTD
Smartcard control language API
C/C# WinSCard.h, Java java.smartcardio.*, Python pyscard
System smartcard interface: Windows's PC/SC, Linux's PC/SC-lite
Manage readers and cards, Transmit ISO7816-4's APDU
Readers
Contact: ISO7816-2,3 (T=0/1)
Contactless: ISO 14443 (T=CL)
APDU packet
API: EMV, GSM, PIV, OpenPGP, ICAO 9303 (BAC/EAC/SAC)
OpenPlatform, ISO7816-4 cmds, custom APDU
Card application 1
Card application 2
Card application 3
SC app programming:
JavaCard, MultOS, .NET, MPCOS
Our focus today

JavaCard basics



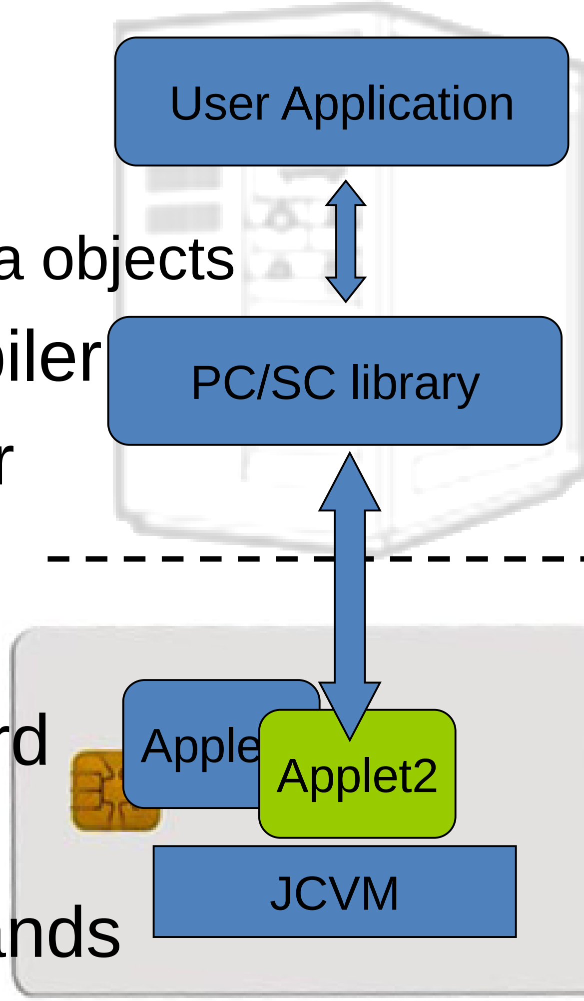
JavaCard

- Maintained by Sun Microsystems (now Oracle)
- Cross-platform and cross-vendor applet interoperability
- Freely available specifications and development kits
  - http://www.oracle.com/technetwork/java/javacard/index.html
- JavaCard applet is Java-like application
  - uploaded to a smart card
  - executed by the JCVM



JavaCard applets

- Written in restricted Java syntax
  - byte/short (int) only, missing most of Java objects
- Compiled using standard Java compiler
- Converted using JavaCard converter
  - check bytecode for restrictions
  - can be signed, encrypted…
- Uploaded and installed into smartcard
  - executed in JC Virtual Machine
- Communication using APDU commands
  - small packets with header



JavaCard versions

- JavaCard 2.1.x/2.2.x (2001-2003)
  - widely supported versions
  - basic symmetric and asymmetric cryptography algorithms
  - PIN, hash functions, random number generation
  - transactions, utility functions
- JavaCard 2.2.2 (2006)
  - last version from 2.x series
  - significantly extended support for algorithms and new concepts
    - long “extended” APDUs, BigNumber support, biometrics
    - external memory usage, fast array manipulation methods…
- JavaCard 3.x (2009)
  - classic and connected editions, later

JavaCard 2.x not supporting

- Dynamic class loading
- Security manager
- Threads and synchronization
- Object cloning, finalization
- Large primitive data types
  - float, double, long and char
  - usually not even int (4 bytes) data type
- Most of std. classes
  - most of java.lang, Object and Throwable in limited form
- Limited garbage collection
  - Newer cards supports, but slow and unreliable

JavaCard 2.x supports

- Standard benefits of the Java language
  - data encapsulation, safe memory management, packages, etc.
- Applet isolation based on the JavaCard firewall
  - applets cannot directly communicate with each other
  - special interface (Shareable) for cross applets interaction
- Atomic operations using transaction mode
- Transient data (buffer placed in RAM)
  - fast and automatically cleared
- A rich cryptography API
  - accelerated by cryptographic co-processor
- Secure (remote) communication with the terminal
  - if GlobalPlatform compliant (secure messaging, security domains)

JavaCard 3.x (most recent is 3.0.4 (2011))

- Relatively recent major release of JavaCard specification
  - significant changes in development logic
  - two separate branches – Classic and Connected edition
- JavaCard 3.x Classic Edition
  - legacy version, extended JC 2.x
  - APDU-oriented communication
- JavaCard 3.x Connected Edition
  - smart card perceived as web server (Servlet API)
  - TCP/IP network capability, HTTP(s), TLS
  - supports Java 6 language features (generics, annotations…)
  - move towards more powerful target devices
  - focused on different segment then classic smart cards

Version support

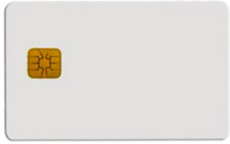
- Need to know supported version for your card
  - convertor adds version identification to package
  - If converted with unsupported version, upload to card fails
- Supported version can be obtained from card
  - JCSystem.getVersion() → [Major.Minor]
  - See https://www.fi.muni.cz/~xsvenda/jcsupport.html
- Available cards supports mostly 2.x specification or 3.x (newer cards)

DEVELOPING JAVACARD APPS

Desktop vs. smart card

- Following slides will be marked with icon based on where it is executed

Process executed on host (PC/NTB…)



Process executed inside smart card

CROCS

```
package example;
import javacard.framework.*;

public class HelloWorld extends Applet {
   protected HelloWorld()  {
     register();
   }
   public static void install(byte[] bArray, short bOffset, byte bLength) {
      new HelloWorld();
   }
   public boolean select() {
      return true;
   }
   public void process(APDU apdu) {
      // get the APDU buffer
      byte[] apduBuffer = apdu.getBuffer();
      // ignore the applet select command dispached to the process
      if (selectingApplet()) return;
      // APDU instruction parser
      if (apduBuffer[ISO7816.OFFSET_CLA] == CLA_MYCLASS) &&
         apduBuffer[ISO7816.OFFSET_INS] == INS_MYINS)) {
         MyMethod(apdu);
      }
      else ISOException.throwIt( ISO7816.SW_INS_NOT_SUPPORTED);
   }
   public void MyMethod(APDU apdu) { /* … */ }
}
```

include packages from javacard.*

extends Applet

Called only once, do all allocations&init HERE

Called repeatedly on application select, do all temporaries preparation HERE

Called repeatedly for every incoming APDU, parse and call your code HERE

JC development process

1. Extends javacard.framework.Applet

2. Compile Java→*.class (Java 1.3 binary format)

3. Convert *.class→*.jar/cap (JavaCard Convertor)

4. Upload *.jar/cap → smart card (GPPro/GPShell)

5. Install applet (GPPro/GPShell)

6. Write user Java app (javax.smartcardio.*)

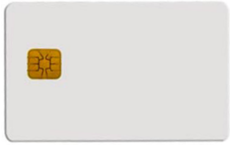
7. Use applet on smart card (APDU)

JavaCard application running model

1. Uploaded package – application binary
2. Installed applet from package – running application
3. Applet is “running” until deleted from card
4. Applet is suspended when power is lost
   – Transient data inside RAM are erased
   – Persistent data inside EEPROM remain
   – Currently executed method is interrupted
5. When power is resumed
   – Unfinished transactions are rolled back
   – Applet continues to run with the same persistent state
   – Applet waits for new command (does *not* continue with interrupted method)
6. Applet is deleted by service command

On-card, off-card code verification

- Off-card verification
  - Basic JavaCard constraints
  - Possibly additional checks (e.g., type consistency when using Shareable interface)
  - Full-blown static analysis possible
  - Applet can be digitally signed
- On-card verification
  - Limited resources available
  - Proprietary checks by JC platform implementation



QUICK AND DIRTY START

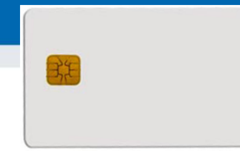
Quick and dirty start – OpenPGP applet

1. Get JavaCard smart card and reader
   – Our example card: NXP JCOP J2A081 80K
2. Install Java SDK and ant build environment
   – Don’t forget to set proper paths (javac, ant)
3. Download AppletPlayground project
   – https://github.com/martinpaljak/AppletPlayground
4. Download GlobalPlatformPro uploader
   – https://github.com/martinpaljak/GlobalPlatformPro

1. Compile and convert applets

- > ant toys
  - 'toys' is *ant* build target inside build.xml
  - Compiles source with Java compiler (javac)
  - Convert with javacard convertor
- (use > ant simpleapplet to build only our applet)

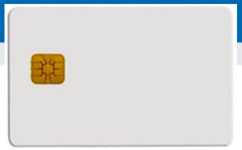
PLAID.cap	03/10/2015 14:27	CAP File	5 KB
PKIApplet.cap	03/10/2015 14:27	CAP File	10 KB
PassportApplet.cap	03/10/2015 14:27	CAP File	18 KB
OpenPGPApplet.cap	03/10/2015 14:26	CAP File	13 KB
OpenEMV.cap	03/10/2015 14:27	CAP File	7 KB
OATH.cap	03/10/2015 14:27	CAP File	7 KB
NDEF.cap	03/10/2015 14:27	CAP File	4 KB
MuscleApplet.cap	03/10/2015 14:26	CAP File	17 KB
ISOApplet.cap	03/10/2015 14:27	CAP File	47 KB
FluffyPGPApplet.cap	03/10/2015 14:27	CAP File	9 KB
DriversLicense.cap	03/10/2015 14:27	CAP File	16 KB




2. Manage applets on smart card

- GlobalPlatformPro tool
  - Authenticates against CardManager
  - Establish secure channel with CM
  - Manage applets (list/upload/delete)

```
Auto-detected ISD AID: A000000003000000
Host challenge: BD525E5585006202
Card challenge: 05211C9591C58232
Card reports SCP02 with version 255 keys
Master keys:
Version 0
ENC: Ver:0 ID:0 Type:DES3 Len:16 Value:404142434445464748494A4B4C4D4E4F
MAC: Ver:0 ID:0 Type:DES3 Len:16 Value:404142434445464748494A4B4C4D4E4F
KEK: Ver:0 ID:0 Type:DES3 Len:16 Value:404142434445464748494A4B4C4D4E4F
Sequence counter: 0521
```



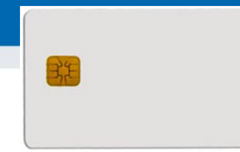
>gp -list –verbose

Reader: Gemplus USB SmartCard Reader 0
ATR: 3BF81300008131FE454A434F5076323431B7
More information about your card:
    http://smartcard-atr.appspot.com/parse?ATR=3BF81300008131FE454A434F507632343
1B7

Auto-detected ISD AID: A000000003000000
Host challenge: 10FFA96848D9EB62
Card challenge: 0520E372F35B4818
Card reports SCP02 with version 255 keys
Master keys:
Version 0
ENC: Ver:0 ID:0 Type:DES3 Len:16 Value:404142434445464748494A4B4C4D4E4F
MAC: Ver:0 ID:0 Type:DES3 Len:16 Value:404142434445464748494A4B4C4D4E4F
KEK: Ver:0 ID:0 Type:DES3 Len:16 Value:404142434445464748494A4B4C4D4E4F
Sequnce counter: 0520
Derived session keys:
Version 0
ENC: Ver:0 ID:0 Type:DES3 Len:16 Value:654E72AAADA31F0A7B5567160DE4C5A7
MAC: Ver:0 ID:0 Type:DES3 Len:16 Value:C6883A00AB6E56384B845A5A6F68CA6C
KEK: Ver:0 ID:0 Type:DES3 Len:16 Value:3875213C9F2123EB01AA420DC83C18F0
Verified card cryptogram: 62CBE443B3F4FB80
Calculated host cryptogram: 9AAC671F9B1E0630

AID: A000000003000000 (|........|)
ISD OP_READY: Security Domain, Card lock, Card terminate, Default selected, CVM (PIN) management

AID: A0000000035350 (|.....SP|)
ExM LOADED: (none)
A000000003535041 (|.....SPA|)

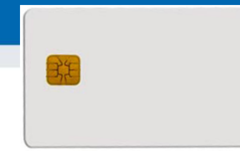


3. Upload applet to smart card

- (already converted applet *.cap is assumed)
- > gp --instal OpenPGPApplet.cap –verbose

CAP file (v2.1) generated on Sat Oct 03 15:13:58 CEST 2015
By Sun Microsystems Inc. converter 1.3 with JDK 1.8.0_60 (Oracle Corporation)
Package: openpgpcard v0.0 with AID D27600012401
Applet: OpenPGPApplet with AID D276000124010200000000000000010000
Import: A0000000620101 v1.3
Import: A0000000620201 v1.3
Import: A0000000620102 v1.3
Import: A0000000620001 v1.0
Cap loaded

- Hint: test with gpg --card-edit



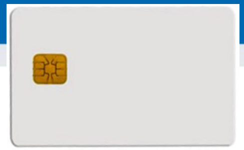
OpenPlatform Package/applet upload

A. Security domain selection
B. Secure channel establishment – security domain
C. Package upload
   - Local upload in trusted environment
   - Remote upload with relayed secure channel
D. Applet installation
   - Separate instance from package binary with unique AID
   - Applet privileges and other parameters passed
   - Applet specific installation data passed

4. Communicate with smart card

- > gp --apdu apdu_in_hex --debug
- Example for SimpleApplet.java
  - gp –-apdu B0541000 -d (generate random numbers)

```
>gp --apdu B0541000 -d
[*] Gemplus USB SmartCard Reader 0
SCardConnect("Gemplus USB SmartCard Reader 0", T=*) -> T=1, 3BF81300008131FE454A
434F5076323431B7
SCardBeginTransaction("Gemplus USB SmartCard Reader 0")
A>> T=1 (4+0000) B0541000
A<< (0016+2) (32ms) 801D52307393AC0AB1CC242F6905B7C5 9000
```



5. Delete applet

- > gp --delete D27600012401 --deletedeps

- (Verify that applet was deleted by gp –list)

DEVELOPING SIMPLE APPLET

JavaCard – My first applet

- Desktop Java vs. JavaCard
  - PHP vs. C ☺
- No modern programming features
  - No threads, no generics, no iterators…
- Limited type system
  - Usually no ints (short int and byte only), no floats, no Strings
- Fun with signed 16-bits values
  - JavaCard is usually 16-bit platform (short)
  - (short) typecast must be performed on intermediate results
  - Shorts are signed => to obtain unsigned byte
    - Convert to short with & 0x00ff

Necessary tools

- Several tool chains available
  - both commercial (RADIII, JCOPTools, G&D JCS Suite)
  - and free (Sun JC SDK, AppletPlayground…)
- We will use:
  - Java Standard Edition Development Kit 1.3 or later
  - Apache Ant 1.7 or later, JavaCard Development Kit 2.2.2
  - JavaCard Ant Tasks (from JC SDK 2.2.2)
  - NetBeans 6.8 or later as IDE
  - GlobalPlatformPro for applets management

```java
package example;
import javacard.framework.*;

public class HelloWorld extends Applet {
   protected HelloWorld()  {
     register();
   }
   public static void install(byte[] bArray, short bOffset, byte bLength) {
      new HelloWorld();
   }
   public boolean select() {
      return true;
   }
   public void process(APDU apdu) {
      // get the APDU buffer
      byte[] apduBuffer = apdu.getBuffer();
      // ignore the applet select command dispached to the process
      if (selectingApplet()) return;
      // APDU instruction parser
      if (apduBuffer[ISO7816.OFFSET_CLA] == CLA_MYCLASS) &&
         apduBuffer[ISO7816.OFFSET_INS] == INS_MYINS)) {
         MyMethod(apdu);
      }
      else ISOException.throwIt( ISO7816.SW_INS_NOT_SUPPORTED);
   }
   public void MyMethod(APDU apdu) { /* … */ }
}
```

include packages from javacard.*

extends Applet

Called only once, do all allocations&init HERE

Called repeatedly on application select, do all temporaries preparation HERE

Called repeatedly for every incoming APDU, parse and call your code HERE

Simple JavaCard applet - code

1. Subclass javacard.framework.Applet
2. Allocate all necessary resources in constructor
3. Select suitable CLA and INS for your method
4. Parse incoming APDU in Applet.process() method
5. Call your method when your CLA and INS are set
6. Get incoming data from APDU object (getBuffer(), setIncomingAndReceive())
7. Use/modify data
8. Send response (setOutgoingAndSend())

select() method

- Method called when applet is set as active
  - for subsequent APDU commands
  - begin of the session
  - use for session data init (clear keys, reset state…)

```
public void select() { // CLEAR ALL SESSION DATA
      chv1.reset(); // Reset OwnerPIN verification status
      remainingDataLength = 0; // Set states etc.
      // If card is not blocked, return true.
      // If false is returned, applet is not selectable
     if (!blocked) return true;
     else return false;
  }
```

- deselect()
  - similar, but when applet usage finish
  - may not be called (sudden power drop) => clear in select

Sending and receiving data

- `javacard.framework.APDU`
  - incoming and outgoing data in APDU object
- Obtaining just apdu header
  - APDU.getBuffer()
- Receive data from terminal
  - APDU.setIncomingAndReceive()
- Send outgoing data
  - APDU.setOutgoingAndSend()

Sending and receiving data – source code

```
private void ReceiveSendData(APDU apdu) {
    byte[]    apdubuf = apdu.getBuffer();     // Get just APDU header (5 bytes)
    short     dataLen = apdu.setIncomingAndReceive(); // Get all incoming data
    // DO SOMETHING WITH INPUT DATA
    // STARTING FROM apdubuf[ISO7816.OFFSET_CDATA]
    // ...
    // FILL SOMETHING TO OUTPUT (apdubuf again)
    Util.arrayFillNonAtomic(apdubuf, ISO7816.OFFSET_CDATA, 10, (byte) 1);
    // SEND OUTGOING BUFFER
    apdu.setOutgoingAndSend(ISO7816.OFFSET_CDATA, 10);
}
```

COMMUNICATION WITH SMART CARD

JavaCard communication lifecycle

1. (Applet is already installed)
2. PC: Reset card (plug smart card in, software reset)
3. PC: Send SELECT command (00 0a 04 00 xxx)
    - received by Card Manager application
    - SC: sets our applet active, select() method is always called
4. PC: Send any APDU command (any of your choice)
    - SC: received by process() method
5. SC: Process incoming data on card, prepare outgoing data
    - encryption, signature…
6. PC: Receive any outgoing data
    - additional special readout APDU might be required
7. PC: Repeat again from step 4
8. PC: (Send DESELECT command)
    - SC: deselect() method might be called

Java javax.smartcardio.* API

- List readers available in system
  - TerminalFactory.terminals()
  - identified by index CardTerminal.get(index)
  - readable string (Gemplus GemPC Card Reader 0)
- Connect to target card
  - Check for card (CardTerminal.isCardPresent())
  - connect to Card (CardTerminal.connect("*"))
  - get channel (Card.getBasicChannel())
  - reset card and get ATR (Card.getATR())

Already used in labs last week – SimpleAPDU project

Java javax.smartcardio.* API (2)

- Select applet on card
  - send APDU with header 00 a4 04 00 LC APPLET_AID
- Send APDU to invoke method
  - prepare APDU buffer (byte array)
  - create CommandAPDU from byte array
  - send CommandAPDU via CardChannel.transmit()
  - check for response data (getSW1() == 0x61)
  - read available response data by 00 C0 00 00 SW2
- Process response
  - status should be ResponseAPDU.getSW() == 0x9000
  - returned data ResponseAPDU.getData()

DEBUGGING APPLET

Debugging applets: simulator

- The smartcard is designed to protect application
  - Debugger cannot be connected to running application
- Option 1: use card simulator (jcardsim.org)
  - Simulation of JavaCard 2.2.2 (based on BouncyCastle)
  - Very helpful, allows for direct debugging (labs)
  - Catch of logical flaws etc.
  - Allows to write automated unit tests!
- Problem: Real limitations of cards are missing
  - supported algorithms, memory, execution speed…

Debugging applets: real cards

- Option 2: use real cards
  - Cannot directly connect debugger, no logging strings…
- Debugging based on error messages
  - Use multiple custom errors rather than ISO7816 errors
  - Distinct error tells you more precisely, where problem happened
- Problem: operation may end with unspecific 0x6f00
  - define specific error code and use ISOException.throwIt(0x666);
  - Insert into method causing 0x6f00, compile, convert, upload, run
  - Localize exact line where 0x6f00 is emitted
- Debugging based on additional custom commands
  - Output current values of arrays, keys…
  - Important: Secure by default principle: debugging possibility should be enabled only on intention (e.g., specific flag in installation data, cannot be enabled later (by an attacker)). Don't let debugging code into release!

Possible causes for unspecific 0x6f00

- Writing behind allocated array
- Using Key that was Key.clear() before
- Insufficient memory to complete operation
- Cipher.init() with uninitialized Key
- Import of RSA key into real card generated by software outside card (e.g., getP() len == 64 vs. 65B for RSA1024)
- Storing reference of APDU object localAPDU = origAPDU;
- Decryption of value stored in byte[] array with raw RSA with most significant bit == 1 (set first byte of array to 0xff to verify)
- Set CRT RSA key using invalid values for given part - e.g. setDP1()
- … and many more ☺

BEST PRACTICES

Execution speed hints (1)

- Difference between RAM and EEPROM memory
  - new allocates in EEPROM (persistent, but slow)
    - do not use EEPROM for temporary data
    - do not use for sensitive data (keys)
  - JCSystem.getTransientByteArray() for RAM buffer
  - local variables automatically in RAM
- Use API algorithms and utility methods
  - much faster, cryptographic co-processor
- Allocate all resources in constructor
  - executed during installation (only once)
  - either you get everything you want or not install at all

Execution speed hints (2)

- Garbage collection limited or not available
  - do not use new except in constructor
- Use copy-free style of methods
  - foo(byte[] buffer, short start_offset, short length)
- Do not use recursion or frequent function calls
  - slow, function context overhead
- Do not use OO design extensively (slow)
- Keep Cipher or Signature objects initialized
  - if possible (e.g., fixed master key)
  - initialization with key takes non-trivial time

CROCS

How many cryptographic engines?

Type of object	NXP CJ2A081	NXP CJ2D081 80K	NXP JCOP21 v2.4.2R3 145KB
AESKey 128	877	729	678
AESKey 256	658	607	565
DESKey 196	748	607	565
Cipher AES	79	74	74
Cipher DES	147	136	136
RSA CRT PRIVATE 1024	72	93	86
RSA PRIVATE 1024	203	152	141
RSA CRT PRIVATE 2048	61	51	47
RSA PRIVATE 2048	108	82	77

Security hints (1)

- Use API algorithms/modes rather than your own
  - API algorithms fast and protected in cryptographic hardware
  - general-purpose processor leaks more information
- Store session data in RAM
  - faster and more secure against power analysis
  - EEPROM has limited number of rewrites (10^5 - 10^6 writes)
- Never store keys and PINs in primitive arrays
  - use specialized objects like OwnerPIN and Key
  - better protected against power, fault and memory read-out attacks

Security hints (2)

- Erase unused keys and sensitive arrays
  - use specialized method if exists (Key.clearKey())
  - or overwrite with random data (Random.generate())
- Use transactions to ensure atomic operations
  - power supply can be interrupted inside code execution
  - be aware of attacks by interrupted transactions - rollback attack
- Do not use conditional jumps with sensitive data
  - branching after condition is recognizable with power analysis

Security hints (3)

- Allocate all necessary resources in constructor
  - applet installation usually in trusted environment
  - prevent attacks based on limiting available resources
- Use automata-based programming model
  - well defined states (e.g., user PIN verified)
  - well defined transitions and allowed method calls
- Some additional hints
  - Gemalto_JavaCard_DevelGuide.pdf
  - http://developer.gemalto.com/fileadmin/contrib/downloads/pdf/Java%20Card%20%26%20STK%20Applet%20Development%20Guidelines.pdf

JavaCard applet firewall issues

- Main defense for separation of multiple applets
- Platform implementations differ
  - Usually due to the unclear and complex specification
- If problem exists then is out of developer's control
- Firewall Tester project (W. Mostowski)
  - Open and free, the goal is to test the platform

```
short[] array1, array2; // persistent variables
short[] localArray = null; // local array
JCSystem.beginTransaction();
       array1 = new short[1];
       array2 = localArray = array1; // dangling reference!
JCSystem.abortTransaction();
```

Summary

- Smart cards are programmable (JavaCard)
  - reasonable cryptographic API
  - coprocessor for fast cryptographic operations
  - multiple applications coexist securely on single card
  - Secure execution environment
- Standard Java 6 API for communication exists
- PKI applet can be developed with free tools
  - PIN protection, on-card key generation, signature…
- JavaCard is not full Java – optimizations, security