PV204 Security technologies

JavaCard platform

Petr Švenda  ***svenda@fi.muni.cz***  ***@rngsec***
Centre for Research on Cryptography and Security, Masaryk University

CROCS
Centre for Research on Cryptography and Security

Please comment on slides with anything unclear, incorrect or suggestions for improvement
https://drive.google.com/file/d/1SkI1tOUh_KeqV2wkQh834WUy7ahfHri_/view?usp=sharing

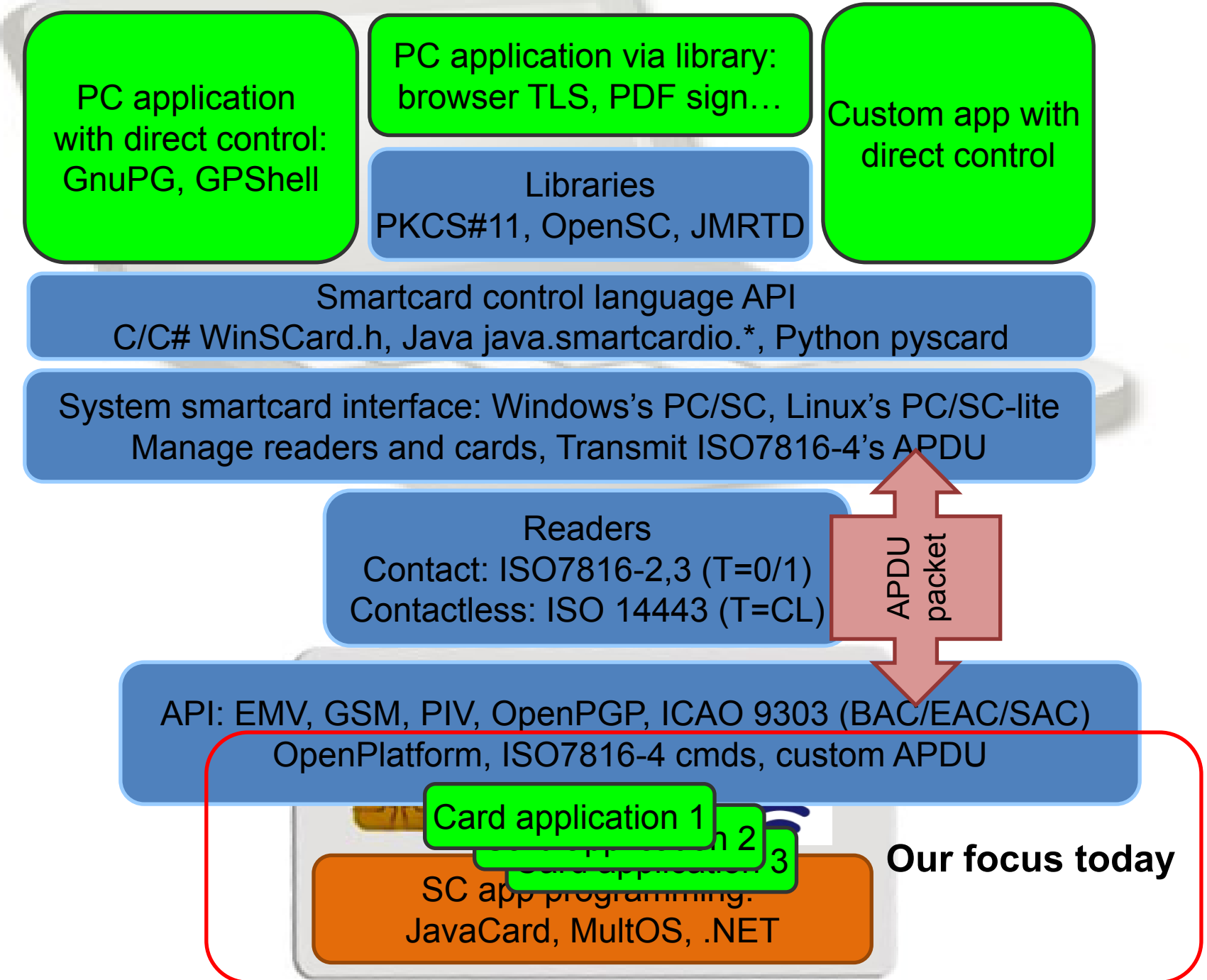
Overview

- JavaCard programming platform
- Skeleton of JavaCard applet
- How to upload and communicate with
- Best practices – security and performance
- Supplementary materials
  - Simple signature applet
  - Simple symmetric cryptography applet

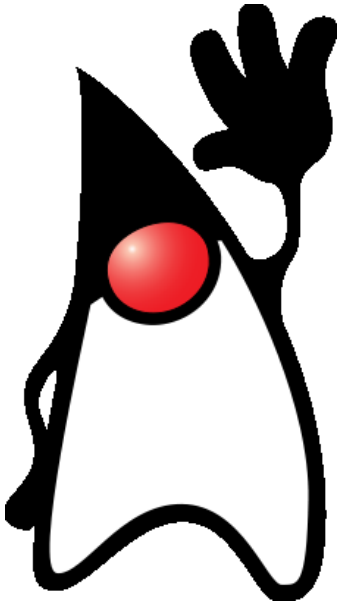
Old vs. multi-application smart cards

- One program only
- Stored persistently in ROM or 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)

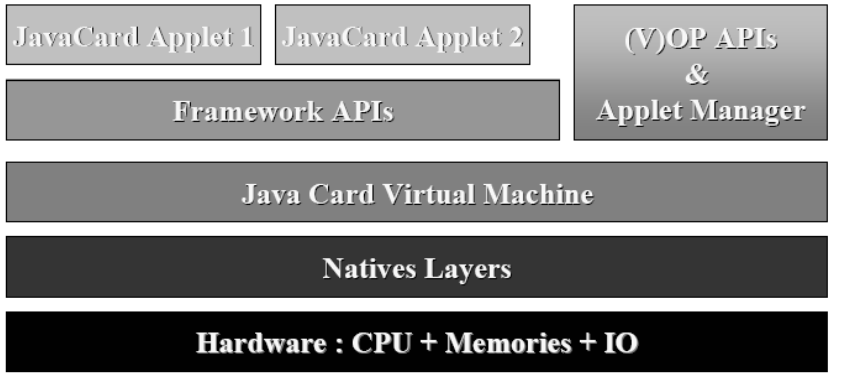

PC application with direct control: GnuPG, GPShell
PC application via library: browser TLS, PDF sign…
Custom app with direct control
Libraries PKCS#11, OpenSC, JMRTD
Smartcard control language API C/C# WinSCard.h, Java java.smartcardio.*, Python pyscard
System smartcard interface: Windows's PC/SC, Linux's 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
SC app programming: JavaCard, MultOS, .NET
Our focus today

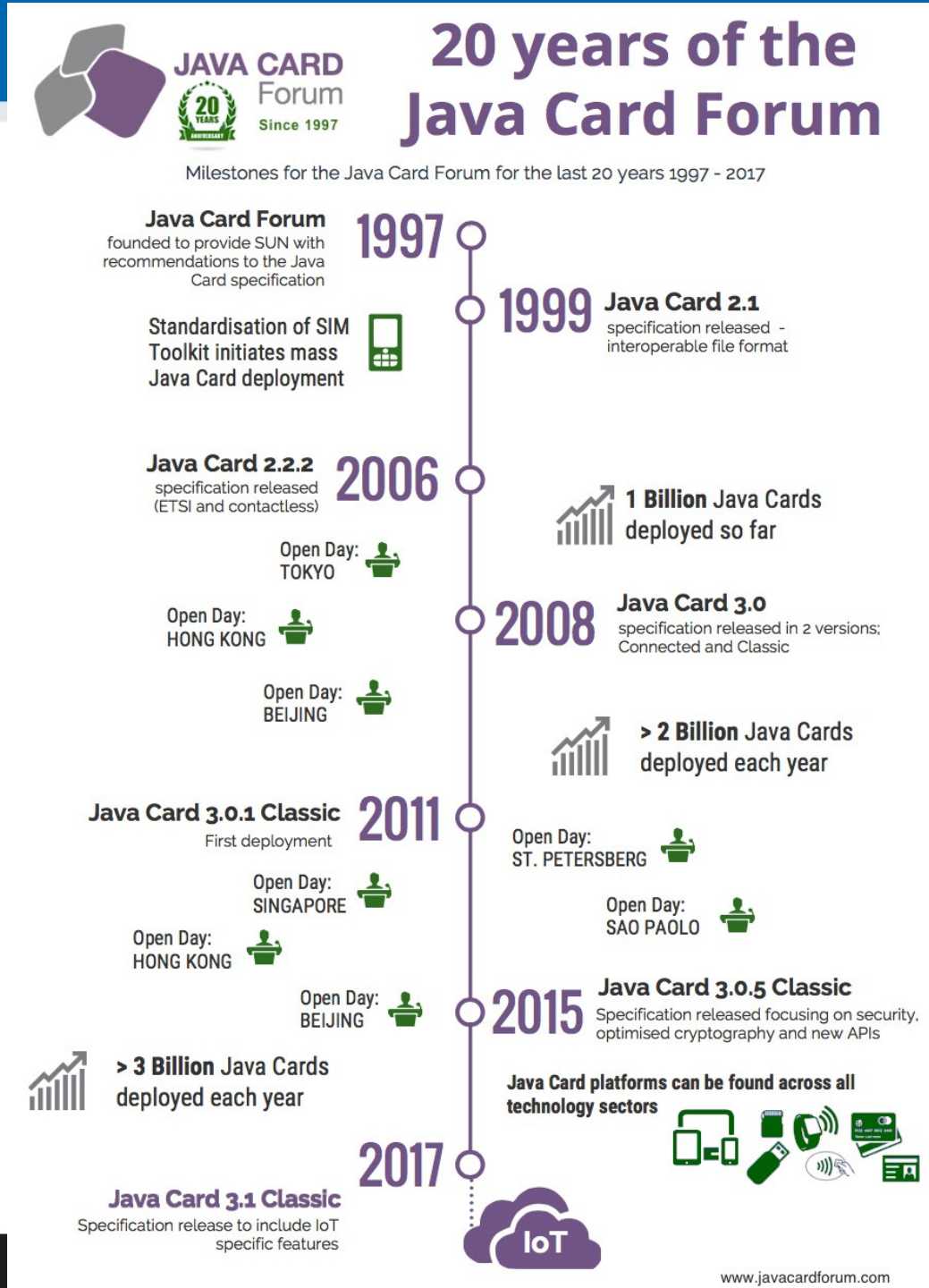
JavaCard basics



JavaCard

- Maintained by Java Card Forum (since 1997)
- 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




JAVA CARD Forum
20 YEARS
Since 1997
20 years of the Java Card Forum
Milestones for the Java Card Forum for the last 20 years 1997 - 2017
Java Card Forum
founded to provide SUN with recommendations to the Java Card specification
1997
Standardisation of SIM Toolkit initiates mass Java Card deployment
1999
Java Card 2.1
specification released - interoperable file format
Java Card 2.2.2
specification released (ETSI and contactless)
2006
1 Billion Java Cards deployed so far
Open Day: TOKYO
Open Day: HONG KONG
2008
Java Card 3.0
specification released in 2 versions; Connected and Classic
Open Day: BEIJING
> 2 Billion Java Cards deployed each year
Java Card 3.0.1 Classic
First deployment
2011
Open Day: ST. PETERSBERG
Open Day: SINGAPORE
Open Day: SAO PAOLO
Open Day: HONG KONG
Open Day: BEIJING
2015
Java Card 3.0.5 Classic
Specification released focusing on security, optimised cryptography and new APIs
> 3 Billion Java Cards deployed each year
Java Card platforms can be found across all technology sectors
2017
Java Card 3.1 Classic
Specification release to include IoT specific features
IoT
www.javacardforum.com

JavaCard 2.x not supporting

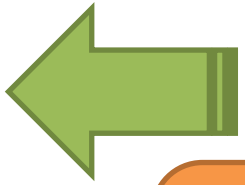
- No dynamic class loading
- No Security manager
- No Threads and synchronization
- No Object cloning, finalization
- No Large primitive data types
  - float, double, long and char
  - usually not even int (4 bytes) data type
- Most of std. classes missing
  - most of java.lang, Object and Throwable in limited form
- Limited garbage collection
  - Newer cards supports, but slow and not always 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 3.0.x (most recent 3.0.5 from 2015)

- 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

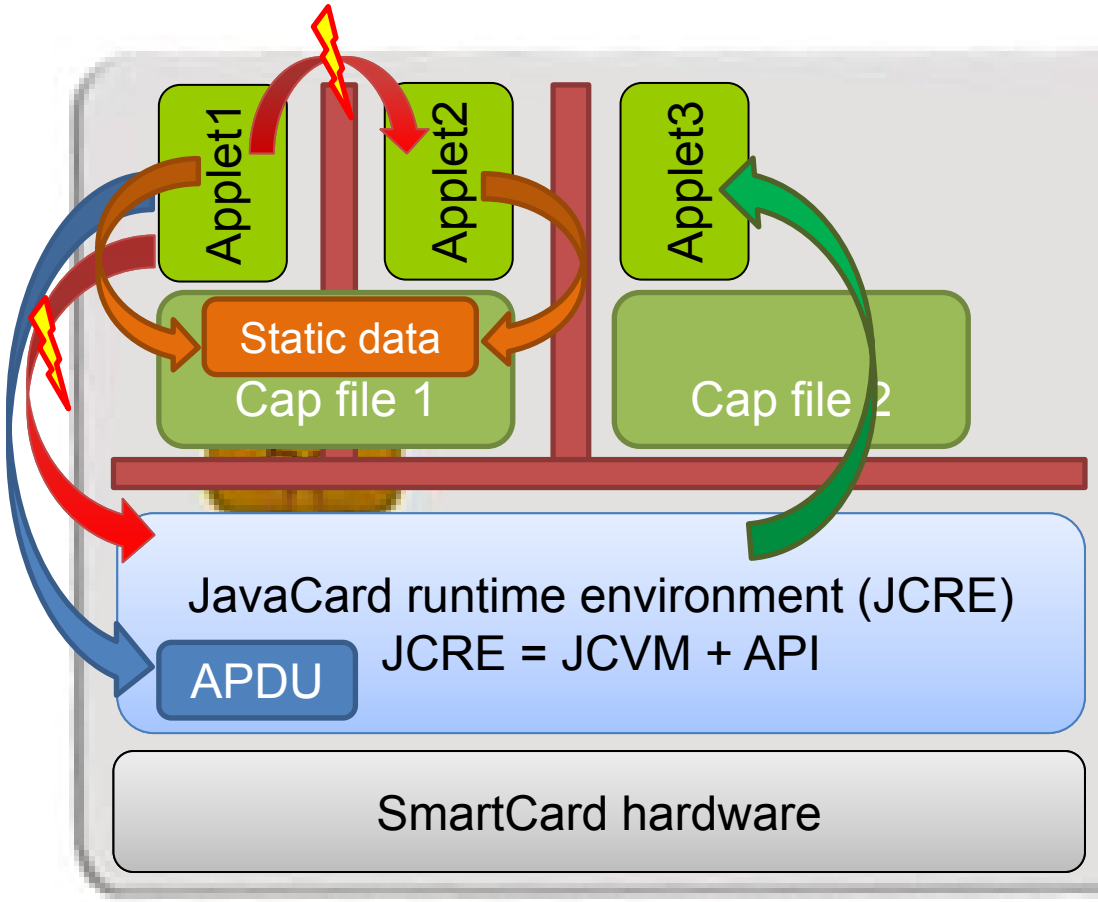


Connected edition is not used so far (potentially dead?)

Version support

- Need to know supported version for your card
  - convertor adds version identification of packages used to binary cap file
  - 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)

JavaCard applet firewall – runtime checks



- Access to other applet's methods and attributes prevented
  - Even if `public`
- Applets can access specific JCRE objects
- JCRE can access all applets (no restriction)
- Static attributes of package accessible by all its applets!

Inspired by http://ekladata.com/IHWNXUB-yernblD2sdiK1zxxQco/5_javacard.pdf

DEVELOPING JAVACARD APPS

Group activity: complex development

- Write down all steps that take place during development
  - Write code, compile, CI…
- Separate groups: Development, Testing, Delivery, Support
  - Pre-printed structure
- (7 minutes)
- Collect areas into mind-mapping software
- Add and discuss relevant tools/steps from JavaCard world
- (10 minutes)

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

```
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_SUPPOR
   }
   public void MyMethod(APDU apdu) { /* ... */ }
}
```

include packages from javacard.*

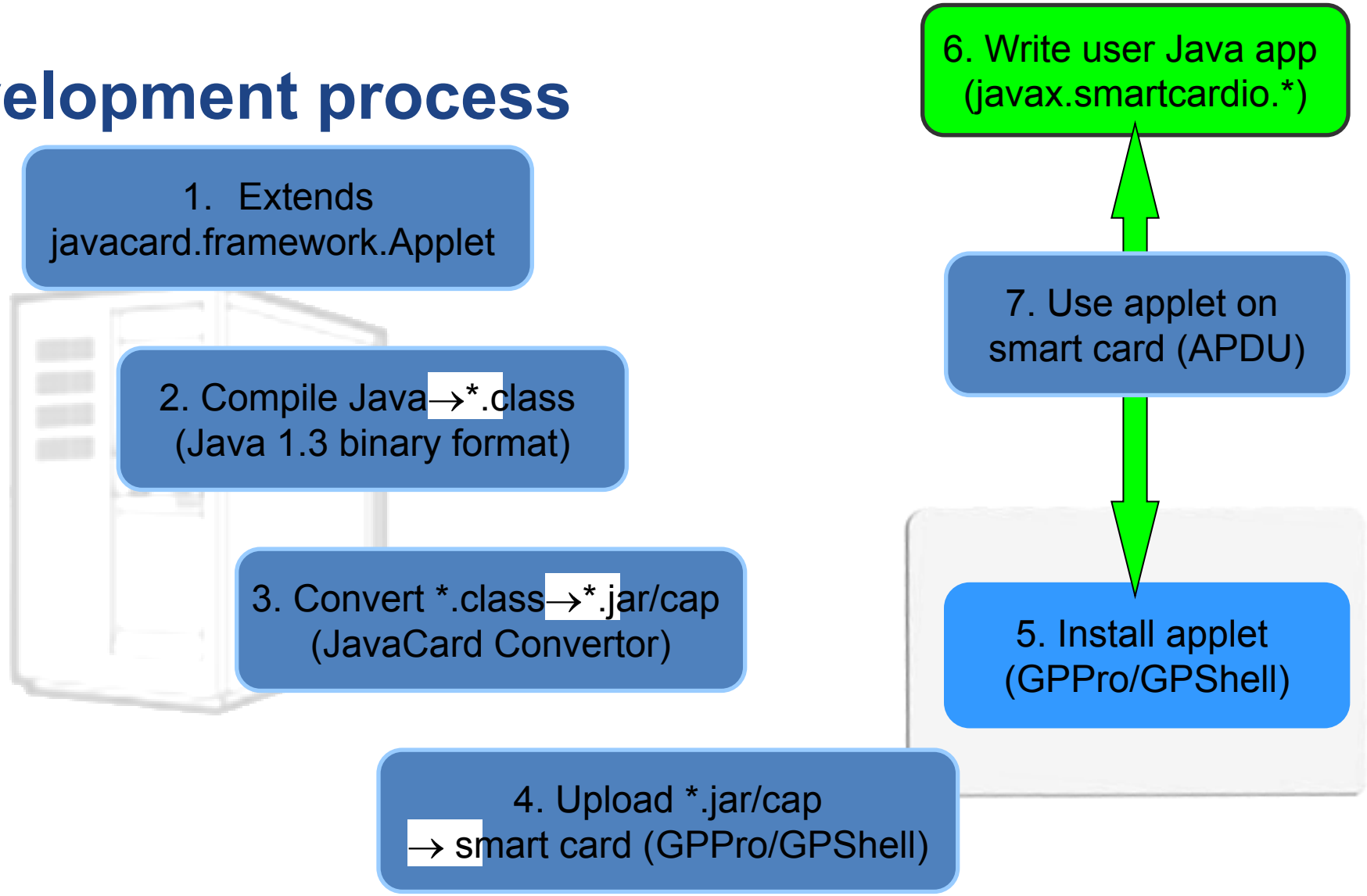
extends Applet

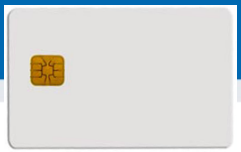
Called only once, do all allocations&init HERE

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





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

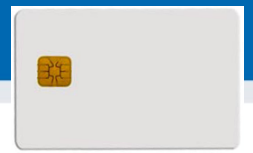
On-card, off-card code verification

- 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

OpenPlatform Package/applet upload

A. Security domain selection
B. Secure channel establishment – security domain
C. Package (cap file) 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

DEVELOPING SIMPLE APPLET

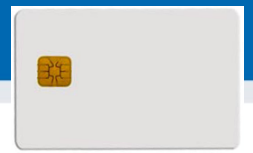


JavaCard – My first applet

- 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

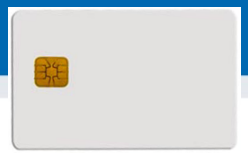
Necessary tools

- 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
  - ant-javacard ant task for building JC applets
    - Apache Ant 1.7 or later, JavaCard Development Kit 2.2.2
  - NetBeans 6.8 or later as IDE
  - GlobalPlatformPro for applets management



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())

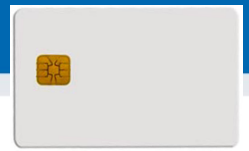


select() method

- 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…)

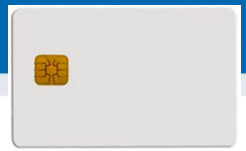
```
public void select() { // CLEAR ALL SESSION DATA
      chv1.reset(); // Reset OwnerPIN verification status
      remainingDataLength = 0; // Set states etc.
      // If card is not blocked, return true.
      // If false is returned, applet is not selectable
     if (!blocked) return true;
     else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

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



Sending and receiving data

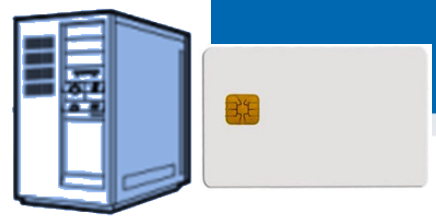
- `javacard.framework.APDU`
  - incoming and outgoing data in APDU object
- Obtaining just apdu header
  - APDU.getBuffer()
- Receive data from terminal
  - APDU.setIncomingAndReceive()
- Send outgoing data
  - APDU.setOutgoingAndSend()



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, APPLET_AID)
2. PC: Reset card (plug smart card in, software reset)
3. PC: Send SELECT command (00 a4 04 00 APPLET_AID)
   - 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)

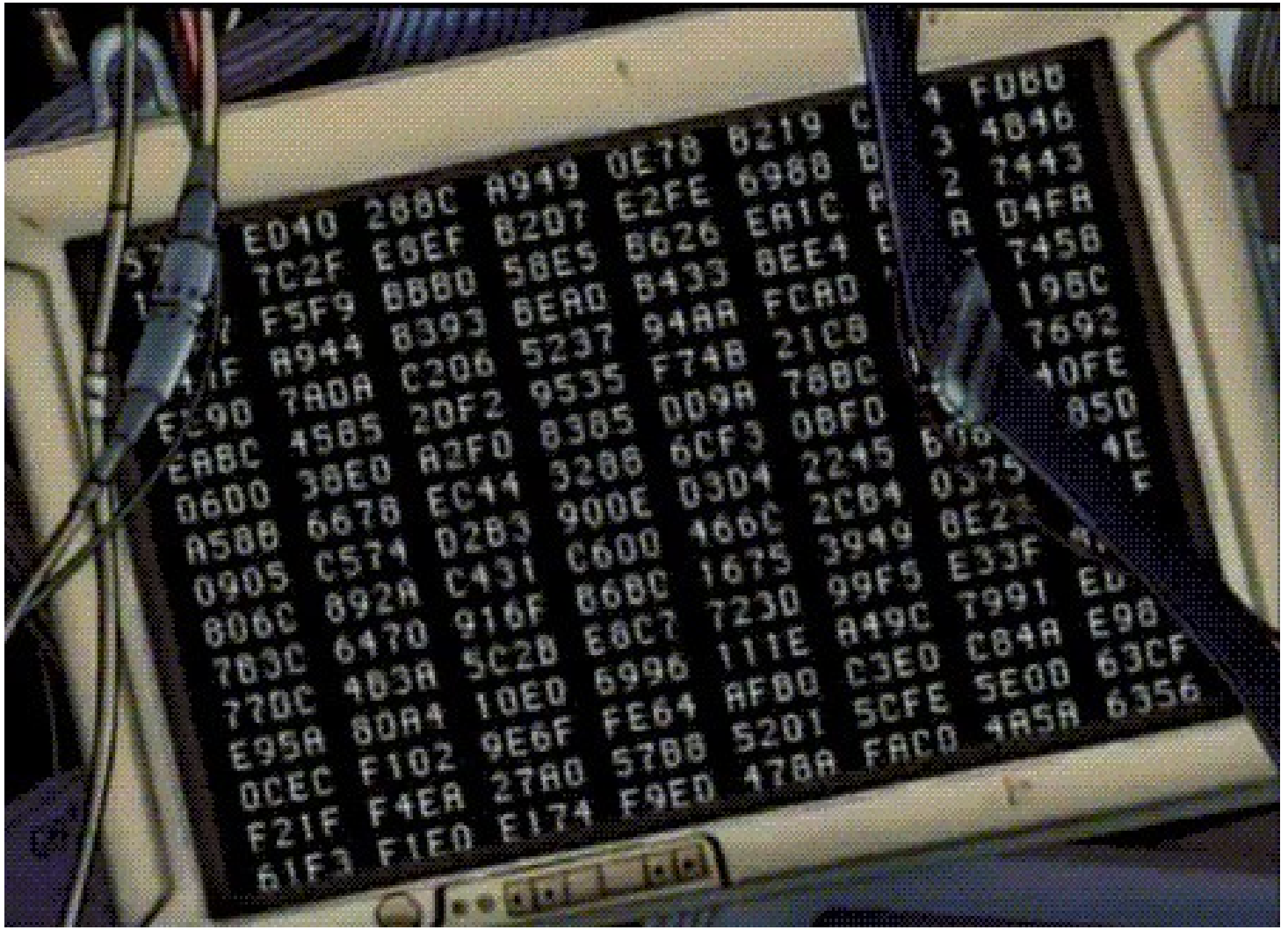
- Select applet on card
  - send APDU with header 00 a4 04 00 LC APPLET_AID
- Send APDU to invoke method
  - prepare APDU buffer (byte array)
  - create CommandAPDU from byte array
  - send CommandAPDU via CardChannel.transmit()
  - check for response data (getSW1() == 0x61)
  - read available response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

DEBUGGING APPLET

Martin Paljak
@martinpaljak

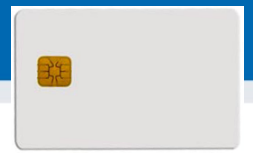
Following

How does smart card programming look like in real life? Here's a typical scenario...



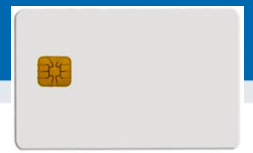
1. Debugging applets: simulator

- 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 and integration tests!
- Problem: Real limitations of cards are missing
  - supported algorithms, memory, execution speed…



2. 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 errors tell you where problem (might) happened
- Problem: operation may end with unspecific 0x6f00
  - Any uncaught exception on card (other than ISOException)
  - Solution1: Capture on card, translate to ISOException
  - Solution2: Locate problematic command by insertion of ISOException.throwIt(0x666); and recompile



Possible causes for exception on card

- 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()
- Too many nested calls, no free space on stack for arguments
- … and many more ☺

Getting more than 0x6f00

```
public void process(javacard.framework.APDU apdu) {
   // ignore the applet select command dispatched to the process
   if (selectingApplet()) return;
   try {
      //
      // ... Standard APDU command dispatching...
      //
   } catch (ISOException e) {
      throw e; // Our exception from code, just re-emit
   } catch (ArrayIndexOutOfBoundsException e) {
      ISOException.throwIt(SW_ArrayIndexOutOfBoundsException);
   } catch (ArithmeticException e) {
      ISOException.throwIt(SW_ArithmeticException);
   } catch (ArrayStoreException e) {
      ISOException.throwIt(SW_ArrayStoreException);
   } catch (NullPointerException e) {
      ISOException.throwIt(SW_NullPointerException);
   } catch (NegativeArraySizeException e) {
      ISOException.throwIt(SW_NegativeArraySizeException);
   } catch (CryptoException e) {
      ISOException.throwIt((short) (SW_CryptoException_prefix | e.getReason()));
   } catch (SystemException e) {
      ISOException.throwIt((short) (SW_Sy
   } catch (PINException e) {
      ISOException.throwIt((short) (SW_PI
   } catch (TransactionException e) {
      ISOException.throwIt((short) (SW_Tr
   } catch (CardRuntimeException e) {
      ISOException.throwIt((short) (SW_Ca
   } catch (Exception e) {
      ISOException.throwIt(Consts.SW_Exce
   }
}
```

Our exception, just re-emit

Some exceptions provide additional information (code). Propagate it further

```
final short SW_Exception                        = (short) 0xff01;
final short SW_ArrayIndexOutOfBoundsException = (short) 0xff02;
final short SW_ArithmeticException              = (short) 0xff03;
final short SW_ArrayStoreException              = (short) 0xff04;
final short SW_NullPointerException             = (short) 0xff05;
final short SW_NegativeArraySizeException       = (short) 0xff06;
final short SW_CryptoException_prefix           = (short) 0xf100;
final short SW_SystemException_prefix           = (short) 0xf200;
final short SW_PINException_prefix              = (short) 0xf300;
final short SW_TransactionException_prefix      = (short) 0xf400;
final short SW_CardRuntimeException_prefix      = (short) 0xf500;
```

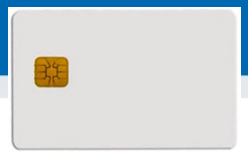
Debugging using custom commands

- Addition of custom commands to “dump” interesting parts of data
  - Intermediate values of internal arrays, unwrapped keys…
- Should obey to *Secure by default principle*
  - Debugging possibility should be enabled only on intention
  - E.g., specific flag in installation data which cannot be enabled later (by an attacker)
  - Don’t let debugging code into release!

BEST PRACTICES (FOR APPLET DEVELOPERS)

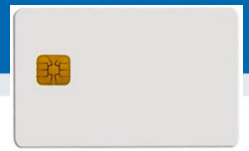
Quiz

1. Expect that your device is leaking in time/power channel. Which option will you use?
   - AES from hw coprocessor or software re-implementation?
   - Short-term sensitive data stored in EEPROM or RAM?
   - Persistent sensitive data in EEPROM or encrypted object?
   - Conditional jumps on sensitive value?
2. Expect that attacker can successfully induct faults (random change of bit(s) in device memory).
   - Suggest defensive options for applet's source code
   - Change in RAM, EEPROM, instruction pointer, CPU flags…



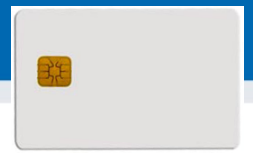
Security hints (1)

- Use API algorithms/modes rather than your own
  - API algorithms fast and protected in cryptographic hardware
  - general-purpose processor leaks more information (side-channels)
- 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, PINs or sensitive data in primitive arrays
  - use specialized objects like OwnerPIN and Key
  - better protected against power, fault and memory read-out attacks
  - If not possible, generate random key in Key object, encrypt large data with this key and store only encrypted data
- Make checksum on stored sensitive data (=> detect fault)



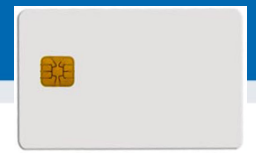
Security hints (2)

- Erase unused keys and sensitive arrays
  - use specialized method if exists (Key.clearKey())
  - or overwrite with random data (Random.generate())
  - Perform always before start of new session
- 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 => timing/power leakage



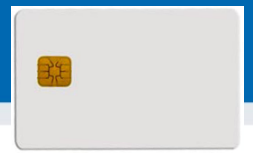
Security hints (3)

- Allocate all necessary resources in constructor
  - applet installation usually in trusted environment
  - prevent attacks based on limiting available resources
- Don’t use static attributes (except constants)
  - Static attribute is shared between multiple instances of applet (bypass applet firewall)
  - Static ptr to array/engine filled by dynamic allocation cannot be removed until package is removed from card (memory “leak”)
- Use automata-based programming model
  - well defined states (e.g., user PIN verified)
  - well defined transitions and allowed method calls



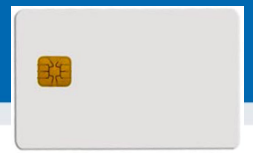
Security hints (4)

- Treat exceptions properly
  - Do not let uncaught native exceptions to propagate from the card
  - Do not let your code to cause basic exceptions like OutOfBoundsException or NullPointerExceptions…



Security hints: fault induction (1)

- Cryptographic algorithms are sensitive to fault induction
  - Single signature with fault from RSA-CRT may leak the private key
  - Perform operation twice and compare results
  - Perform reverse operation and compare (e.g., verify after sign)
- Use constants with large hamming distance
  - Induced fault in variable will likely cause unknown value
  - Use 0xA5 and 0x5A instead of 0 and 1 (correspondingly for more)
  - Don’t use values 0x00 and 0xff (easier to force all bits to 0 or 1)
- Check that all sub-functions were executed [Fault.Flow]
  - Fault may force program stack or stack to skip some code
  - Idea: Add defined value to flow counter inside target sub-function, check later for expected sum. Add also in branches.



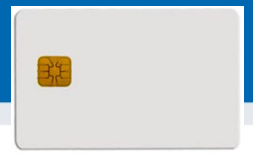
Security hints: fault induction (2)

- Replace single condition check by complementary check
  - **`conditionalValue`** is sensitive value
  - Do not use boolean values for sensitive decisions

```
if (conditionalValue == 0x3CA5965A) { // enter critical path
  // . . .
  if (~conditionalValue != 0xC35A69A5) {
    faultDetect(); // fail if complement not equal to 0xC35A69A5
  }
  // . . .
}
```

- Verify number of actually performed loop iterations

```
int i;
for ( i = 0; i < n; i++ ) { // important loop that must be completed
//. . .
}
if (i != n) { // loop not completed
  faultDetect();
}
```



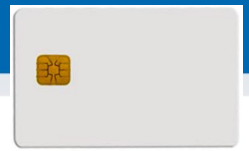
Security hints: fault induction (3)

- Insert random delays around sensitive operations
  - Randomization makes targeted faults more difficult
  - for loop with random number of iterations (for every run)
- Monitor and respond to detected induced faults
  - If fault is detected (using previous methods), increase fault counter.
  - Erase keys / lock card after reaching some threshold (~10)
    - Natural causes may occasionally cause fault => > 1

How and when to apply protections

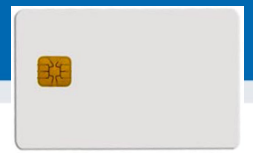
- ✓ Does the device need protection?
- ✓ Understand the resistance of the hardware
- ✓ Identify potential weakness in design
- ✓ Select patterns to use
- ✓ Understand your compiler
- ✓ Code it
- ✓ Test the resistance of the device

Riscure



Execution speed hints (1)

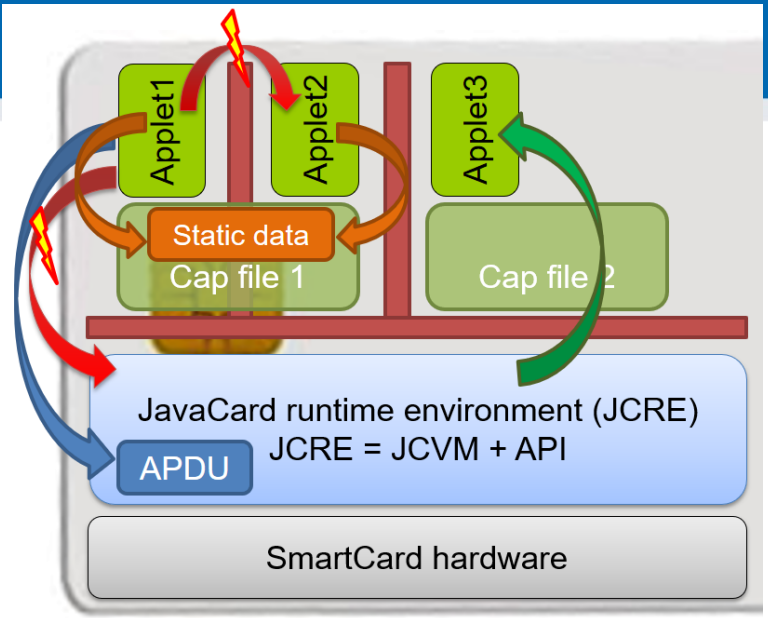
- Big difference between RAM and EEPROM memory
  - new allocates in EEPROM (persistent, but slow)
    - do not use EEPROM for temporary data
    - do not use for sensitive data (keys)
  - JCSystem.getTransientByteArray() for RAM buffer
  - local variables automatically in RAM
- Use algorithms from JavaCard API and utility methods
  - much faster, cryptographic co-processor
- Allocate all necessary 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 for subsequent derivation)
  - initialization with key takes non-trivial time

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
  - http://www.sos.cs.ru.nl/applications/smartcards/firewalltester/

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

Relevant open-source projects

- Easy building of applets
  - https://github.com/martinpaljak/ant-javacard
  - https://github.com/ph4r05/javacard-gradle-template
- AppletPlayground (ready to “fiddle” with applets)
  - https://github.com/martinpaljak/AppletPlayground
- Card simulator https://jcardsim.org
- Profiling performance
  - https://github.com/crocs-muni/JCAlgTest
  - https://github.com/OpenCryptoProject/JCProfiler
- Curated list of JavaCard applets
  - https://github.com/EnigmaBridge/javacard-curated-list
- Low-level ECPoint library
  - https://github.com/OpenCryptoProject/JCMathLib

Mandatory reading

- Mandatory
  - Secure Application Programming in the presence of Side Channel Attacks, Riscure
    - IS, Riscure_Whitepaper_Side_Channel_Patterns.pdf
- Optional
  - Gemalto JavaCard developers guide
    - IS, Gemalto_JavaCard_DevelGuide.pdf
  - Java Card lecture, Erik Poll, Radboud Uni
    - http://ekladata.com/IHWNXUB-yernblD2sdiK1zxxQco/5_javacard.pdf

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

SUPPLEMENTARY MATERIALS

QUICK AND DIRTY START

Quick and dirty start – OpenPGP applet

1. Get JavaCard smart card and reader
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:40414243444546474849 4A4B4C4D4E4F
MAC: Ver:0 ID:0 Type:DES3 Len:16 Value:40414243444546474849 4A4B4C4D4E4F
KEK: Ver:0 ID:0 Type:DES3 Len:16 Value:40414243444546474849 4A4B4C4D4E4F
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:40414243444546474849 4A4B4C4D4E4F
MAC: Ver:0 ID:0 Type:DES3 Len:16 Value:40414243444546474849 4A4B4C4D4E4F
KEK: Ver:0 ID:0 Type:DES3 Len:16 Value:40414243444546474849 4A4B4C4D4E4F
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 D276000124010200000000000010000
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

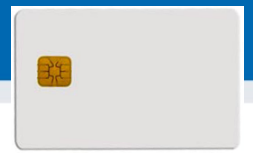
- > gp --apdu apdu_in_hex --debug
- Example for SimpleApplet.java
  - gp –-apdu B0541000 -d (generate random numbers)

```
>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

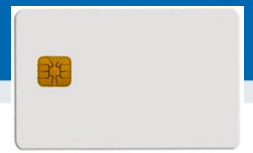
- > gp --delete D27600012401 --deletedeps
- (Verify that applet was deleted by gp –list)

DEVELOPING SIMPLE PKI APPLET



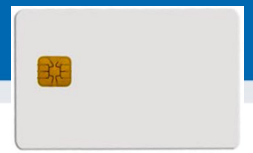
PKI-relevant JavaCard API

- Access controlled by PIN
  - `javacard.security.OwnerPIN`
- Asymmetric cryptography keys
  - javacard.security.KeyPair, PublicKey, PrivateKey
- Digital signatures
  - javacard.security.Signature
- Asymmetric encryption
  - javacard.security.Cipher



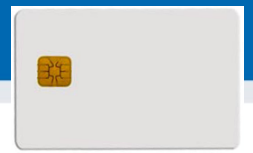
PIN verification functionality

- `javacard.framework.OwnerPIN`
- Management functions (available for “admin”)
  - Create PIN (new OwnerPIN())
  - Set initial PIN value (OwnerPIN.update())
  - Unblock PIN (OwnerPIN. resetAndUnblock())
- Common usage functions (available to user)
  - Verify supplied PIN (OwnerPIN.check())
  - Check if was verified (OwnerPIN.isValidated())
  - Get remaining tries (OwnerPIN.getTriesRemaining())
  - Set new value (OwnerPIN.update())



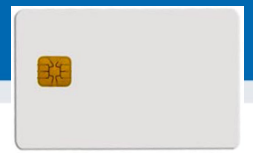
PIN code

```
// CREATE PIN OBJECT (try limit == 5, max. PIN length == 4)
OwnerPIN m_pin = new OwnerPIN((byte) 5, (byte) 4);
// SET CORRECT PIN VALUE
m_pin.update(INIT_PIN, (short) 0, (byte) INIT_PIN.length);
// VERIFY CORRECTNESS OF SUPPLIED PIN
boolean correct = m_pin.check(array_with_pin, (short) 0, (byte)
array_with_pin.length);
// GET REMAING PIN TRIES
byte j = m_pin.getTriesRemaining();
// RESET PIN RETRY COUNTER AND UNBLOCK IF BLOCKED
m_pin.resetAndUnblock();
```



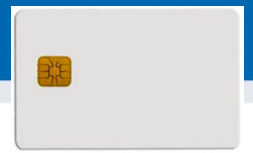
Digital signature

- Management functions
  - Generate new key pair (KeyPair().genKeyPair())
  - Export public key (KeyPair().getPublic())
  - (export private key) (KeyPair().getPrivate())
  - create Signature object (Signature.getInstance())
  - init with public/private key (Signature.init())
- Common usage functions
  - sign message (Signature.update(), Signature.sign())
  - verify signature (Signature.update(),verify())



On-card asymmetric key generation

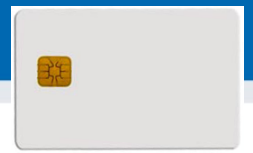
- `javacard.security.KeyPair`
- Key pair is generated directly on smart card
  - very good entropy source (TRNG)
  - private key never leaves the card (unless you allow in code)
  - fast sign/verify operation
- But who is sending data to sign/decrypt?
  - protect signature method by PIN.isValidated() check
  - use secure channel to prevent injection of attacker's message
  - terminal still must be trustworthy



Key generation - source code

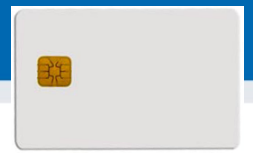
```
// CREATE RSA KEYS AND PAIR
m_keyPair = new KeyPair(KeyPair.ALG_RSA_CRT, KeyBuilder.LENGTH_RSA_1024);

// STARTS ON-CARD KEY GENERATION PROCESS
m_keyPair.genKeyPair();

// OBTAIN REFERENCES TO PRIVATE AND PUBLIC KEY OBJECT
m_publicKey = m_keyPair.getPublic();
m_privateKey = m_keyPair.getPrivate();
```

Example shows RSA 1024b – not recommended
Use KeyBuilder.LENGTH_RSA_2048 instead
(But 2 APDUs are required to transmit signature back)



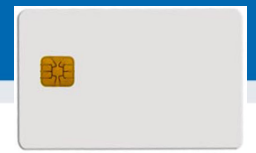
Public (private) key export/import

- Obtain algorithm-specific key object from KeyPair
  - e.g., RSAPublicKey pubKey = keyPair.getPublic();
  - get exponent and modulus
    - getExponent() & getModulus() methods
  - send it back to terminal via APDU
- Similar situation with key import
  - setExponent() & setModulus() methods
- Private key export
  - It is up to you if your code will allow private key export (usually not)
  - Otherwise similar as for RSAPublicKey
  - more parameters with RSAPrivateCrtKey (CRT mode)



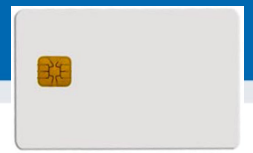
javacard.security.Signature

- Both symmetric and asymmetric crypto signatures
  - RSA_SHA_PKCS1 (always), ECDSA_SHA, DSA (less common)
  - DES_MAC8_NOPAD (always), ISO9797 (common), AES (common)
  - check in advance what your card supports (JCAlgTester)
- Message hashing done on card (asymmetric sign)
  - message received in single or multiple APDUs
  - Signature.update(), Signature.sign()
- If you need just sign of message hash
  - use Cipher object to perform asymmetric crypto operation



Signature – source code

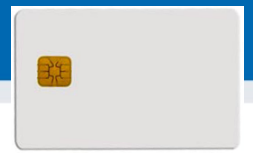
```
// CREATE SIGNATURE OBJECT
Signature m_sign = Signature.getInstance(Signature.ALG_RSA_SHA_PKCS1, false);
// INIT WITH PRIVATE KEY
m_sign.init(m_privateKey, Signature.MODE_SIGN);

// SIGN INCOMING BUFFER
signLen = m_sign.sign(apdubuf, ISO7816.OFFSET_CDATA, (byte) dataLen,
                      m_ramArray, (byte) 0);
```



Asymmetric encryption

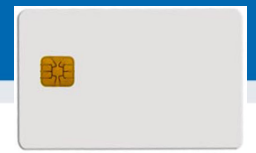
- javacardx.crypto.Cipher
- Usage similar to Signature object
  - generate key pair
  - export/import public key
  - initialize Key and set mode (MODE_ENCRYPT/DECRYPT)
  - process incoming data (Cipher.update(), doFinal())
- Supported algorithms
  - RSA_NOPAD (always), RSA_PKCS1 (almost always)

DEMO - SYMMETRIC CRYPTOGRAPHY APPLET



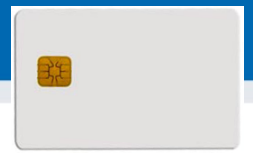
Random numbers

- `javacard.security.RandomData`
- Two versions of random generator
  - ALG_SECURE_RANDOM (truly random)
  - ALG_PSEUDO_RANDOM (deterministic from seed)
- Generate random block
  - RandomData.generateData()
- Very fast and high quality output
  - bottleneck is usually card-to-terminal link



RandomData – source code

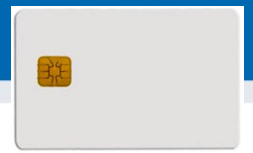
```
private RandomData     m_rngRandom = null;
// CREATE RNG OBJECT
m_rngRandom = RandomData.getInstance(RandomData.ALG_SECURE_RANDOM);
// GENERATE RANDOM BLOCK WITH 16 BYTES
m_rngRandom.generateData(array, (short) 0, ARRAY_ONE_BLOCK_16B);
```



Key generation and initialization

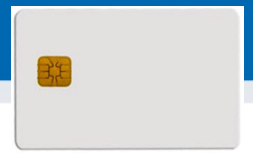
- Allocation and initialization of the key object (KeyBuilder.buildKey())
- Receive (or generate random) key value
- Set key value (AESKey.setKey())

```
// …. INICIALIZATION SOMEWHERE (IN CONSTRUCT)
// CREATE AES KEY OBJECT
AESKey m_desKey = (AESKey) KeyBuilder.buildKey(KeyBuilder.TYPE_AES,
    KeyBuilder.LENGTH_AES_256, false);
// Generate random data to be used as key
m_rngRandom.generateData(array, (short) 0,
    (short) KeyBuilder. KeyBuilder.LENGTH_AES_256/8);

// SET KEY VALUE
m_aesKey.setKey(array, (short) 0);
```



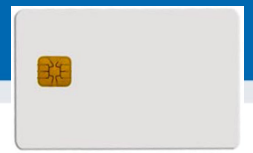
Symmetric cryptography encryption

- `javacard.security.Cipher`
- Allocate and initialize cipher object
  - Cipher.getInstance(), Cipher.init()
- Encrypt or decrypt data
  - Cipher.update(), Cipher.doFinal()



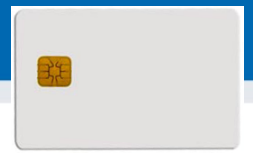
Encryption with 3DES – source code

```
// INIT CIPHER WITH KEY FOR ENCRYPT DIRECTION
m_encryptCipher.init(m_desKey, Cipher.MODE_ENCRYPT);
//….

// ENCRYPT INCOMING BUFFER
void Encrypt(APDU apdu) {
     byte[]    apdubuf = apdu.getBuffer();
     short     dataLen = apdu.setIncomingAndReceive();

     // CHECK EXPECTED LENGTH (MULTIPLY OF 64 bites)
     if ((dataLen % 8) != 0) ISOException.throwIt(SW_CIPHER_DATA_LENGTH_BAD);

     // ENCRYPT INCOMING BUFFER
     m_encryptCipher.doFinal(apdubuf, ISO7816.OFFSET_CDATA, dataLen, m_ramArray, (short) 0);

     // COPY ENCRYPTED DATA INTO OUTGOING BUFFER
     Util.arrayCopyNonAtomic(m_ramArray, (short) 0, apdubuf, ISO7816.OFFSET_CDATA, dataLen);

     // SEND OUTGOING BUFFER
     apdu.setOutgoingAndSend(ISO7816.OFFSET_CDATA, dataLen);
}
```



Message authentication code (MAC)

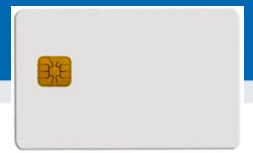
- `javacard.security.Signature`
- Usage similar to asymmetric signatures
- Create signature object for target MAC algorithm
- Initialize with symmetric cryptography key
- Supported algorithms
  - DES_MAC8 (always), AES_MAC8 (increasingly common)



MAC – source code

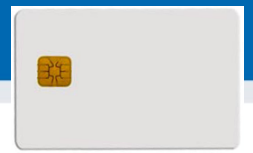
```
private  Signature     m_sessionCBCMAC = null;
private  DESKey        m_session3DesKey = null;

// CREATE SIGNATURE OBJECT
m_sessionCBCMAC = Signature.getInstance(Signature.ALG_DES_MAC8_NOPAD, false);
// CREATE KEY USED IN MAC
m_session3DesKey = (DESKey) KeyBuilder.buildKey(KeyBuilder.TYPE_DES,
KeyBuilder.LENGTH_DES3_3KEY, false);

// INITIALIZE SIGNATURE DES KEY
m_session3DesKey.setKey(m_ram, (short) 0);
// SET KEY INTO SIGNATURE OBJECT
m_sessionCBCMAC.init(m_session3DesKey, Signature.MODE_SIGN);

// GENERATE SIGNATURE OF buff ARRAY, STORE INTO m_ram ARRAY
m_sessionCBCMAC.sign(buff, ISO7816.OFFSET_CDATA, length, m_ram, (short) 0);
```

- Example based on 3DES, can be AES as well



Data hashing

- `javacard.security.MessageDigest`
- Create hashing object for target algorithm
  - MessageDigest.getInstance()
- Reset internal state of hash object
  - MessageDigest.reset()
- Process all parts of data
  - MessageDigest.update()
- Compute final hash digest
  - MessageDigest.doFinal()
- Supported algorithms
  - MD5, SHA-1 (always), SHA-256 (increasingly common)
  - related to supported Signature algorithms



Data hashing – source code

```
// CREATE SHA-1 OBJECT
MessageDigest m_sha1 = MessageDigest.getInstance(
   MessageDigest.ALG_SHA, false);

// RESET HASH ENGINE
m_sha1.reset();
// PROCESS ALL PARTS OF DATA
while (next_part_to_hash_available) {
 m_sha1.update(array_to_hash, (short) 0, (short) array_to_hash.length);
}
// FINALIZE HASH VALUE (WHEN LAST PART OF DATA IS AVAILABLE)
// AND OBTAIN RESULTING HASH VALUE
m_sha1.doFinal(array_to_hash, (short) 0, (short) array_to_hash.length,
   out_hash_array, (short) 0);
```

GPPro – M. Paljak

gp.exe -install applet.cap -verbose
Reader: OMNIKEY AG Smart Card Reader USB 0
ATR: 3BF81800008031FE450073C8401300900092
More information about your card:
http://smartcard-atr.appspot.com/parse?ATR=3BF81800008031FE450073C8401300900092

Auto-detected ISD AID: A000000003000000
Host challenge: 764D6A0982DC5E17
Card challenge: 0005112D5C02E152
Card reports SCP02 with version 1 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
Diversifiefd master keys:
Version 0
ENC: Ver:0 ID:0 Type:DES3 Len:16 Value:8D16CDB90D9A1BCB9C3B208FB491DFF6
MAC: Ver:0 ID:0 Type:DES3 Len:16 Value:D3A3DD0DB2C1F84F79E3BC0EF4B0A78E
KEK: Ver:0 ID:0 Type:DES3 Len:16 Value:F80C3E807D4C57293B651693ED999448
Sequnce counter: 0005
Derived session keys:
Version 0
ENC: Ver:0 ID:0 Type:DES3 Len:16 Value:6BCC8856C64D5A6090A603C5FFBA7F4F
MAC: Ver:0 ID:0 Type:DES3 Len:16 Value:3BDCE52AE932EFF43E506C498BAC9F21
KEK: Ver:0 ID:0 Type:DES3 Len:16 Value:FFC30797EFA7EC37A28E4485052EA21D
Verified card cryptogram: 37C4139407A2F0DD
Calculated host cryptogram: A9376B4721194AFA
CAP file (v2.1) generated on Tue Aug 04 14:34:51 CEST 2015
By Sun Microsystems Inc. converter 1.3 with JDK 1.8.0_31 (Oracle Corporation)
Package: AlgTest v1.0 with AID 6D797061636B616731
Applet: JCAlgTestApplet with AID 6D7970616330303031
Import: A0000000620001 v1.0
Import: A0000000620102 v1.2
Import: A0000000620101 v1.2
CAP loaded

DEMO: OPENPGP APPLET

OpenPGP

- Standard for PGP/GPG compliant applications
- Includes specification for card with private key(s)
  - openpgp-card-1.0.pdf
- Supported (to some extend) in GnuPG
- Pre-personalized OpenPGP cards available
  - http://www.g10code.de/p-card.html
- Open source Java Card applet available
  - JOpenPGPCard
  - http://sourceforge.net/projects/jopenpgpcard/
  - our card can be used

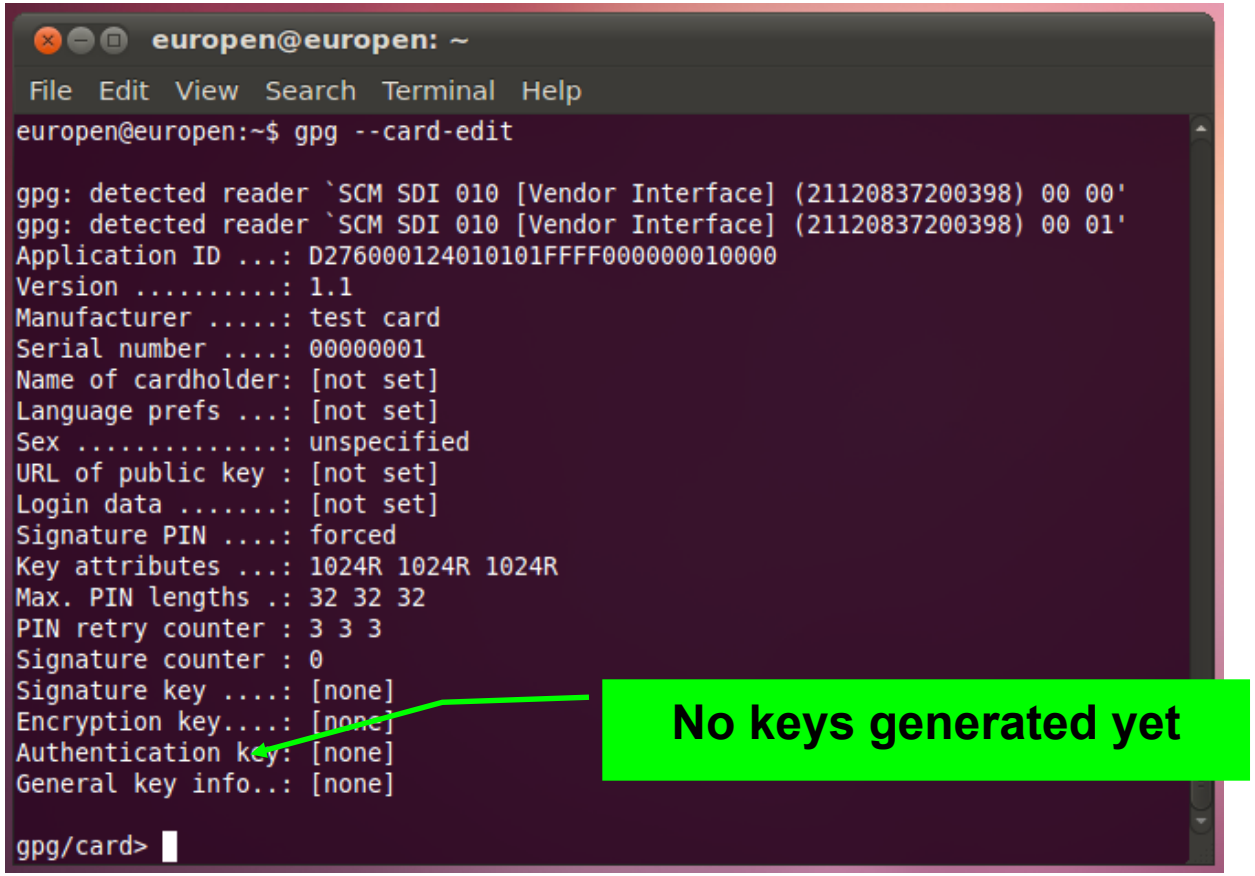
JOpenPGPCard applet

- Main parts
  - two level of PIN protection
  - on-card keys generation, public key export
  - on-card encryption/signature
- Compilation and upload
  - Project settings (preconfigured)
  - AID (given in OpenPGP specification)
  - GPShell script
- Compile and upload applet to card

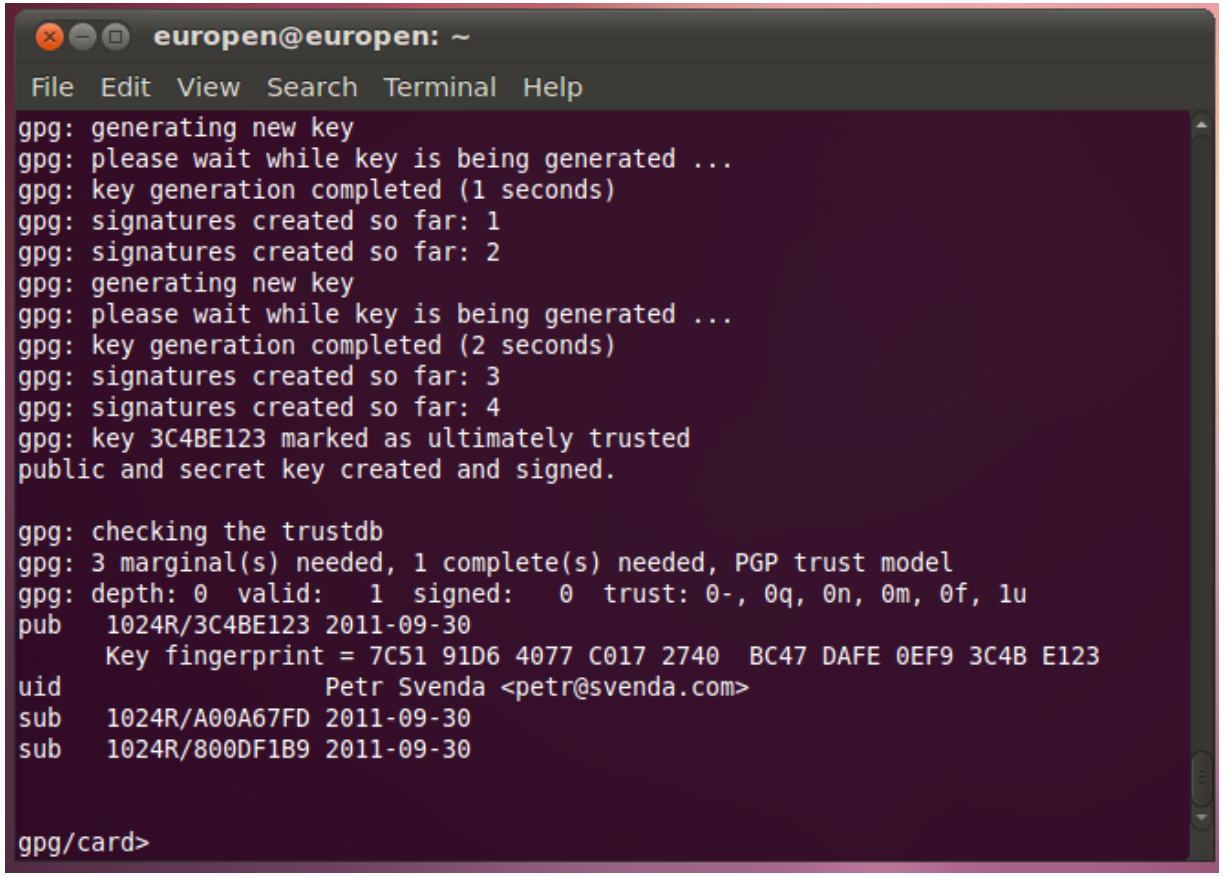
Compilation and upload

- gpg --card-edit
- Command> admin
- Command> help
- Command> generate
  - follow the instructions (default PINs)
  - signature, decryption and authentication key
  - private keys generated directly on the card
  - public keys exported to GPG keyring
- Change your PIN by Command> passwd

GPG --card-edit



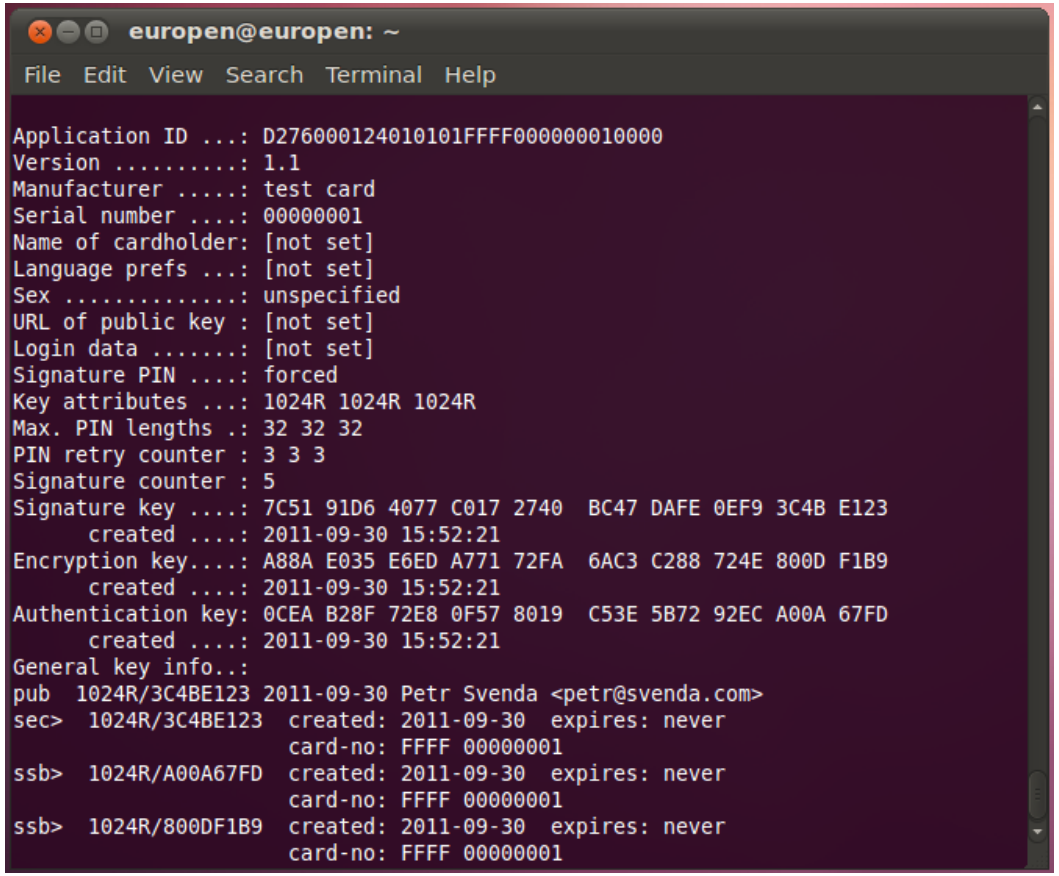
GPG – keys generation finished



What we have…

- Card with OpenPGP-compliant applet
- GPG generated private&public keypairs
  - sign, enc, auth
- Public keys exported from card and imported to local keyring
- Can be used to sign, encrypt message on command line
- Can be further integrated into applications
  - Thunderbird + Enigmail + GPG

(gpg –card-edit) Command> list



Using GPG with smart card

- gpg --clearsign --output myfile.sig --sign myfile
  - our public key is already imported to keyring
  - PIN is required to sign (notice signature count so far)
  - --clearsign causes output in BASE64
- gpg --verify myfile.sig
  - smart card not required, public key in keyring
- gpg --output gpshell.log.gpg --recipient petr@svenda.com --encrypt gpshell.log
  - smart card not required, public key in keyring
- gpg --decrypt gpshell.log.gpg

MORE DETAILS ABOUT JAVACARD

JavaCard – more to be discovered

- Recursion is slooow...
- Memory allocation issues
  - EEPROM vs. RAM allocations, *new* operator
  - No (real-time) garbage collector!
- Persistent objects
- Transactions, atomic operations
- JavaCard applet firewall

```
function f(…) {
    byte a[] = new byte[10];
    byte b[] = JCSystem.makeTransientByteArray(...);
    byte c;
}
```

GPShell upload&install

- Upload and install converted *.cap file
  - GPShell tool with script specific for target card
  - GP SCP channel version (mode_201, mode_211)
  - select CardManager by AID (various AIDs)
  - authenticate and open secure channel (open_sc)
  - delete previous applet version (1. applet, 2. package)
  - load and install (install command, many params)
  - install may pass personalization data (master key…)
- Check applet functionality
  - from GPShell script, no need for secure channel
  - select your applet by AID (select –AID xxx)
  - send test APDU (send_apdu -APDU xxx)

JavaCard – PIN verification

- Image/code for PIN verification
  - Vulnerable to transaction rollback

```
public class OwnerPIN implements PIN {
    byte triesLeft; // persistent counter

    boolean check(...) {
        ...
        triesLeft--;
        ...
    }
}
```

JavaCard – PIN verification done better

- Non-atomic operations

```
public class OwnerPIN implements PIN {
  byte[] triesLeft = new byte[1]; // persistent counter
  byte[] temps =
     JCSystem.makeTransientByteArray(1,
                 JCSystem.CLEAR_ON_RESET);

  boolean check(...) {
    ...
    temps[0] = triesLeft[0] - 1;
    // update the try counter non-atomically:
    Util.arrayCopyNonAtomic(temps, 0, triesLeft, 0, 1);
   ...
  }
}
```

JavaCard – Atomic vs. Non-Atomic

- Persistent memory updates
  - Two ways of updating
  - FillArrayNonAtomic, CopyArrayNonAtomic
- Code refactoring
  - Original short/byte values have to be converted to arrays[1]

JavaCard – Atomic vs. Non-Atomic

- Non-deterministic variable rollback

```
a[0] = 0
beginTransaction()
  a[0] = 1;
  arrayFillNonAtomic(a,0,1,2);
  // a[0] = 2;
abortTransaction()
```

```
a[0] = 0;
beginTransaction();
  arrayFillNonAtomic(a,0,1,2);
  // a[0] = 2;
  a[0] = 1;
abortTransaction();
```

- Result dependency on the commands order
  - **a[0] == 0 vs. a[0] == 2**