

Tutorial: JavaCards



Programming cryptographic smart cards

Petr Švenda svenda@fi.muni.cz

Faculty of Informatics, Masaryk University

CRCS

Centre for Research on
Cryptography and Security

Outline

- Short intro to multi-application smart cards
- Typical usage scenarios
- Programming smart cards
- Platform performance and capabilities

Slides and source codes for tutorial available here:

[HTTP://CROCS.CZ/SC](http://CROCS.CZ/SC)

Tutorial slides and sources

- <http://crcs.cz/sc>
- 7 readers and cards available here
- You can try card programming during conference
- Don't forget to return reader and card back please
- Hint: You can start downloading Java SDK now

INTRO TO SMART CARDS

Basic types of (smart) cards



- Contactless “barcode”
 - Fixed identification string (RFID, < 5 cents)
- Simple memory cards (magnetic stripe, RFID)
 - Small write memory (< 1KB) for data, (~10 cents)
- Memory cards with PIN protection
 - Memory (< 5KB), simple protection logic (<\$1)

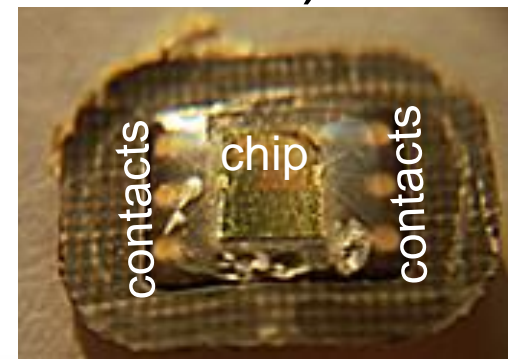
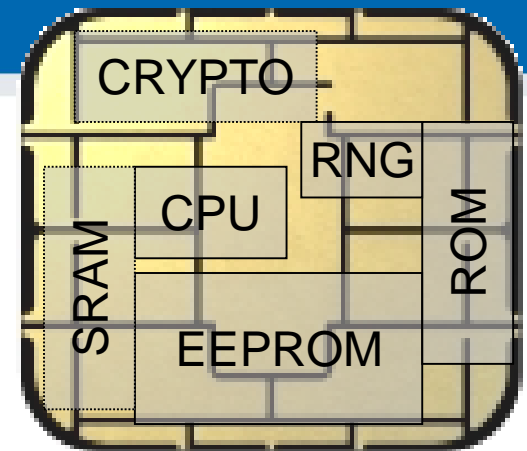
Basic types of (smart) cards (2)

- Cryptographic smart cards
 - Support for (real) cryptographic algorithms
 - Mifare Classic (\$1), Mifare DESFire (\$3)
- User-programmable smart cards
 - Java cards, .NET cards, MULTOS cards (\$10-\$30)



Cryptographic smart cards

- SC is quite powerful device
 - 8-32 bit processors @ 5-20MHz
 - persistent memory 32-100kB (EEPROM)
 - volatile fast RAM, usually $\ll 10$ kB
 - truly random generator
 - cryptographic coprocessor (3DES, RSA-2048,...)
- 8.8 billion units shipped in 2014 (ABI Research)
 - mostly smart cards
 - telco, payment and loyalty...



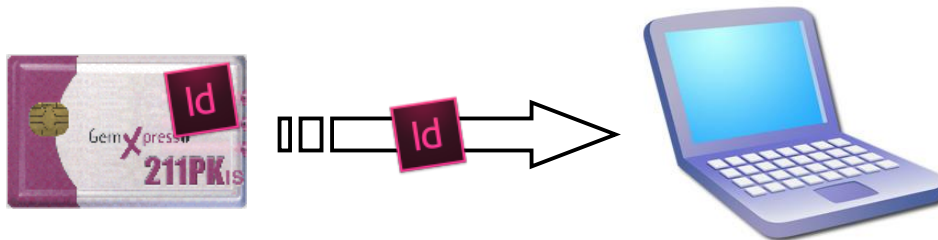
Main advantages of crypto smart cards

- High-level of security (CC EAL4 and higher)
- Fast cryptographic coprocessor
- Programmable secure execution environment
- Secure memory and storage
- On-card asymmetric key generation
- High-quality and very fast RNG
- Possibility for secure remote card control

MODES OF USAGE

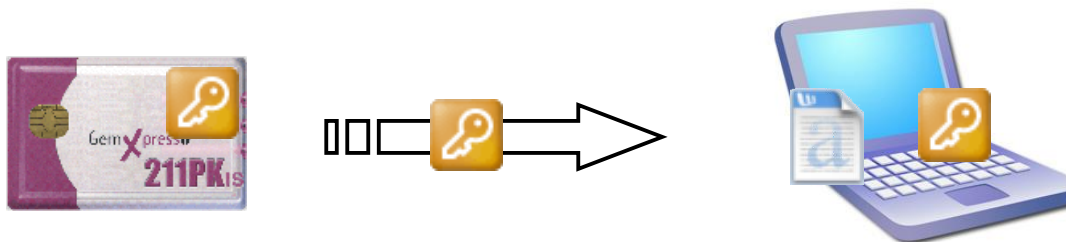
Smart card carries fixed information

- Fixed information ID transmitted, no secure channel
- Low cost solution (nothing “smart” needed)
- Problem: Attacker can eavesdrop and clone chip



Smart card as a secure carrier

- Key(s) stored on a card, loaded to a PC before encryption/signing/authentication, then erased
- High speed usage of key possible (\gg MB/sec)
- Attacker with an access to PC during operation will obtain the key
 - key protected for transport, but not during the usage



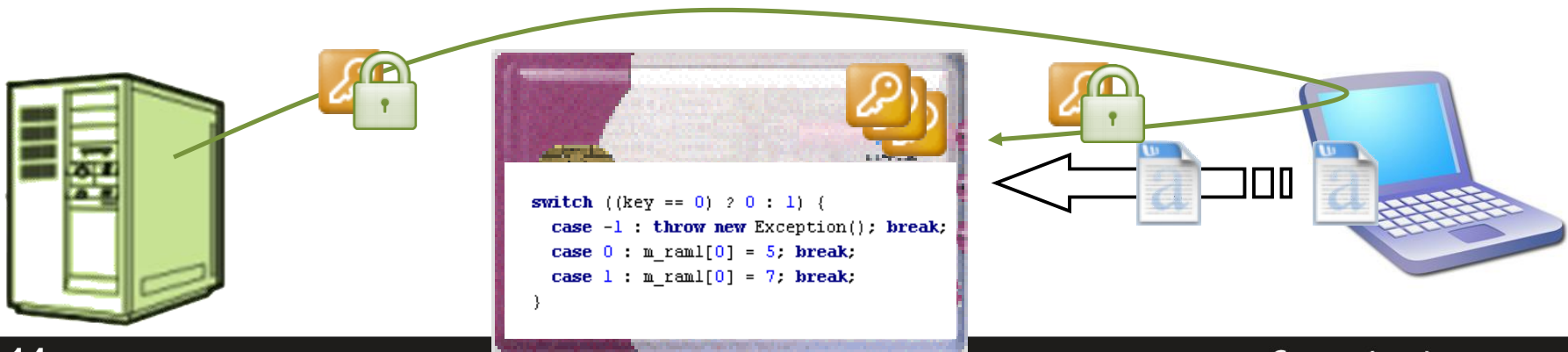
Smart card as encryption/signing device

- PC just sends data for encryption/signing...
- Key never leaves the card
 - personalized in secure environment
 - protected during transport and usage
- Attacker must attack the smart card
 - or wait until card is inserted and PIN entered!
- Low speed encryption (\sim kB/sec)
 - low communication speed / limited card performance



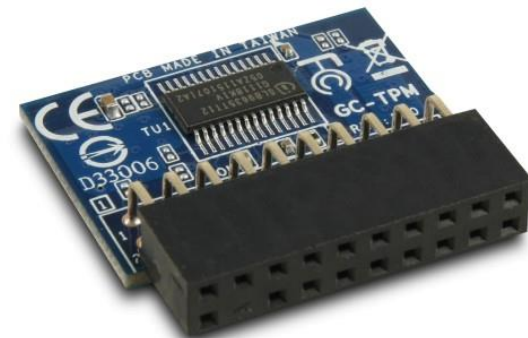
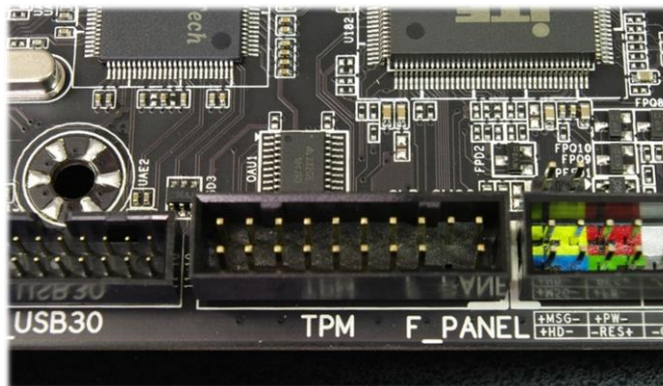
Smart card as computational device

- PC just sends input for application on smart card
- Application code & keys never leave the card
 - smart card can do complicated programmable actions
 - can open secure channels to other entity
 - PC act as a transparent relay only (no access to data)
- Attacker must attack the smart card

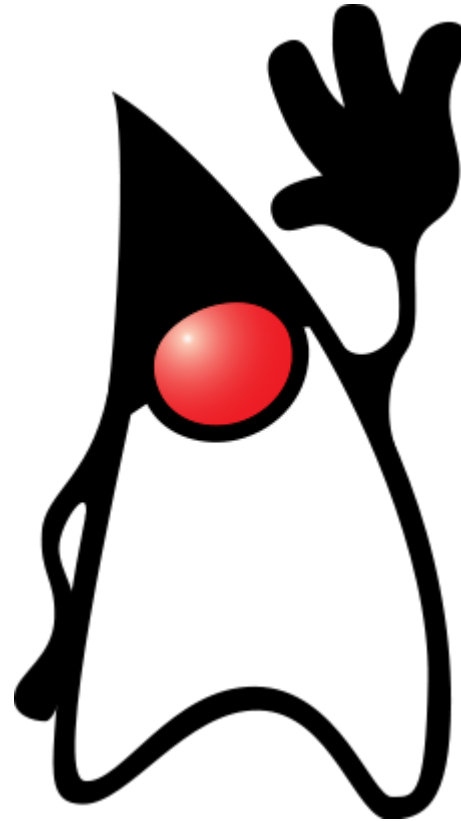


Smart card as root of trust

- Secure boot process, remote attestation
- Smart card provides robust store with integrity
- Application can verify before pass control (measured boot)
- Computer can authenticate with remote entity...

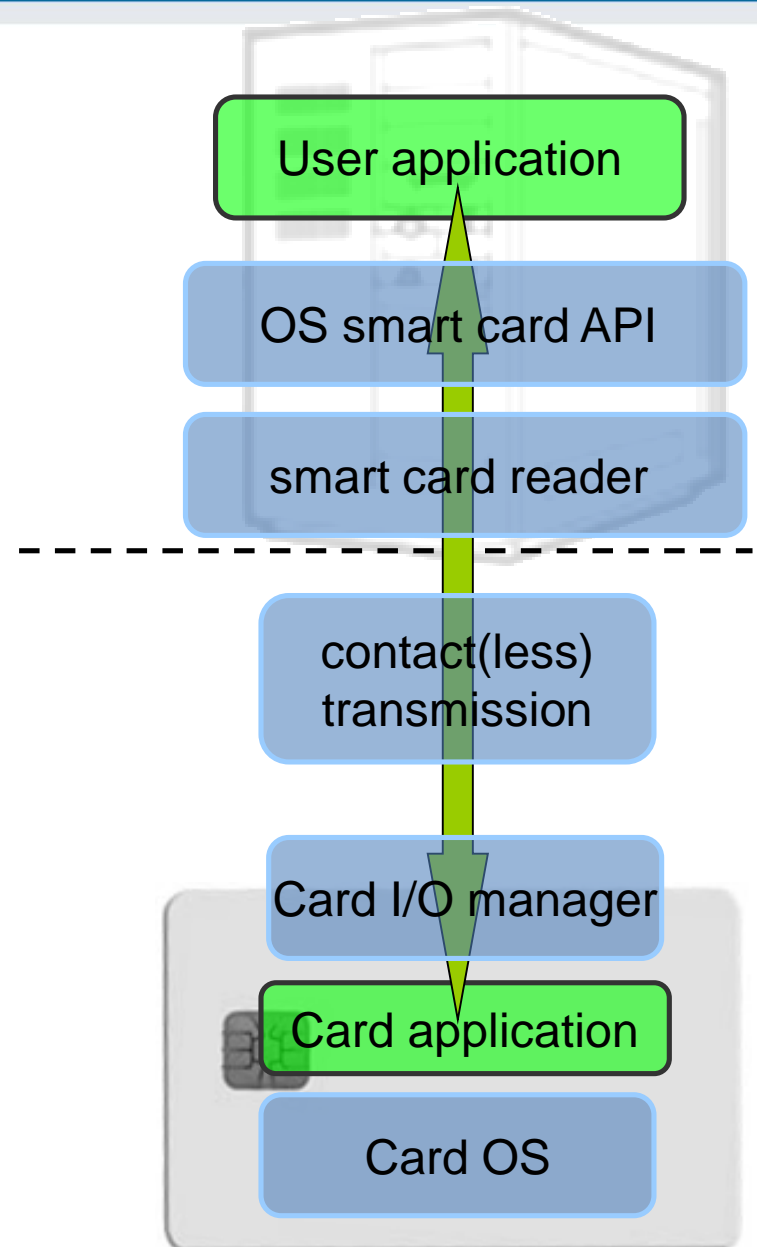


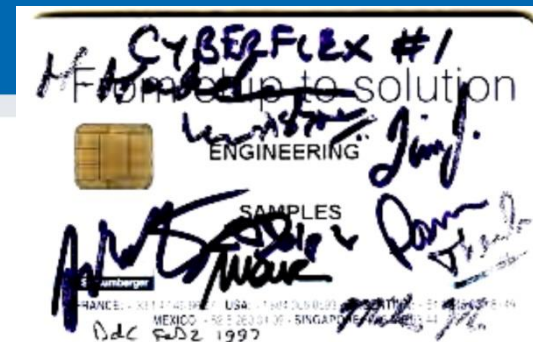
Java Card basics



Main standards

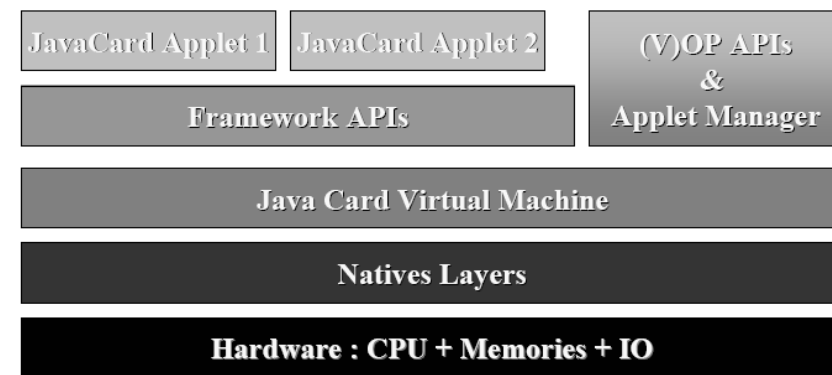
- ISO7816
 - card physical properties
 - physical layer communication protocol
 - packet format (APDU)
- PC/SC, PKCS#11
 - standardized interface on host side
 - card can be proprietary
- GlobalPlatform
 - remote card management interface
 - secure installation of applications
- JavaCard
 - open programming platform from Sun
 - applets portable between cards





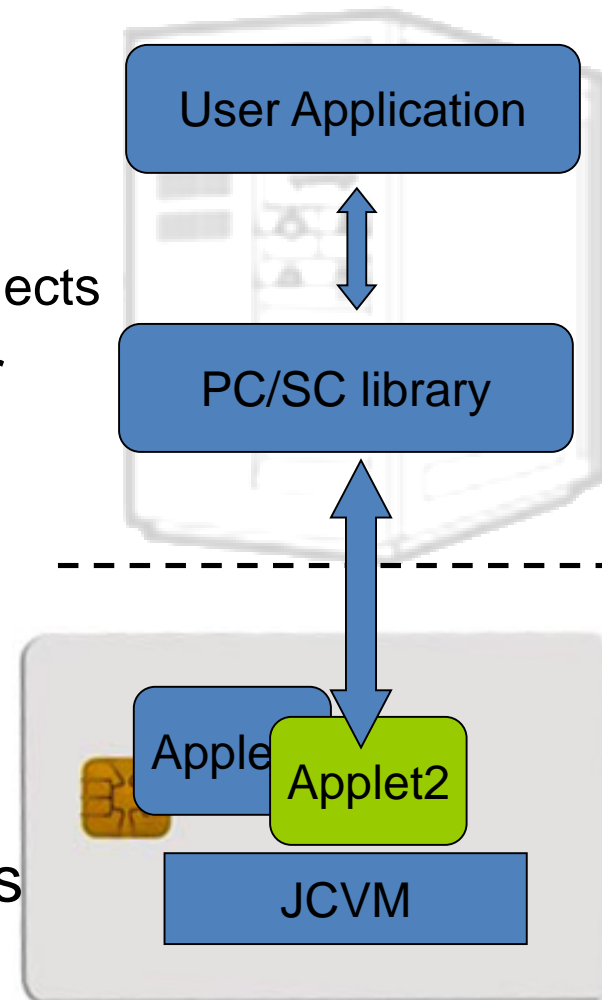
JavaCard specification (1996)

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



Java Card applets

- Writing in restricted Java syntax
 - byte/short (int) only, missing most of Java objects
- Compiled using standard Java compiler
- Converted using Java Card 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



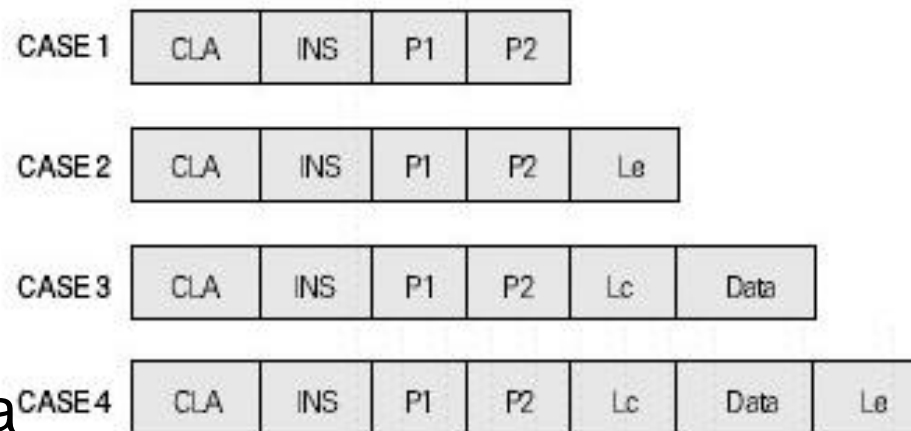
JavaCard API versions

- Java Card 2.1.x/2.2.x
 - widely supported versions
 - basic symmetric and asymmetric cryptography algorithms
 - PIN, hash functions, random number generation
 - transactions, utility functions
- Java Card 2.2.2
 - last version from 2.x series
 - significantly extended support for algorithms and new concepts
 - long “extended” APDUs, BigInteger support
 - biometric capability
 - external memory usage, fast array manipulation methods...
- JavaCard 3.x (classic vs. connected editions)

DEVELOPING JAVACARD APPS

APDU (Application Protocol Data Unit)

- APDU is basic logical communication datagram
 - header (5 bytes) and up to ~256 bytes of user data
- Header format
 - CLA – instruction class
 - INS – instruction number
 - P1, P2 – optional data
 - Lc – length of incoming data
 - Data – user data
 - Le – length of the expected output data



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

include packages
from javacard.*

```
public class HelloWorld extends Applet {
```

extends Applet

```
    protected HelloWorld() {
        register();
    }
```

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

Called only once, do
all allocations&init
HERE

```
    public boolean select() {
        return true;
    }
```

Called repeatedly on
application select, do
all temporaries
preparation HERE

```
    public void process(APDU apdu) {
```

```
        // get the APDU buffer
```

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

```
        // ignore the applet select command dispatched to the process
```

```
        if (selectingApplet()) return;
```

```
        // APDU instruction parser
```

```
        if (apduBuffer[ISO7816.OFFSET_CLA] == CLA_MYCLASS) &&
            apduBuffer[ISO7816.OFFSET_INS] == INS_MYINS) {
            MyMethod(apdu);
        }
```

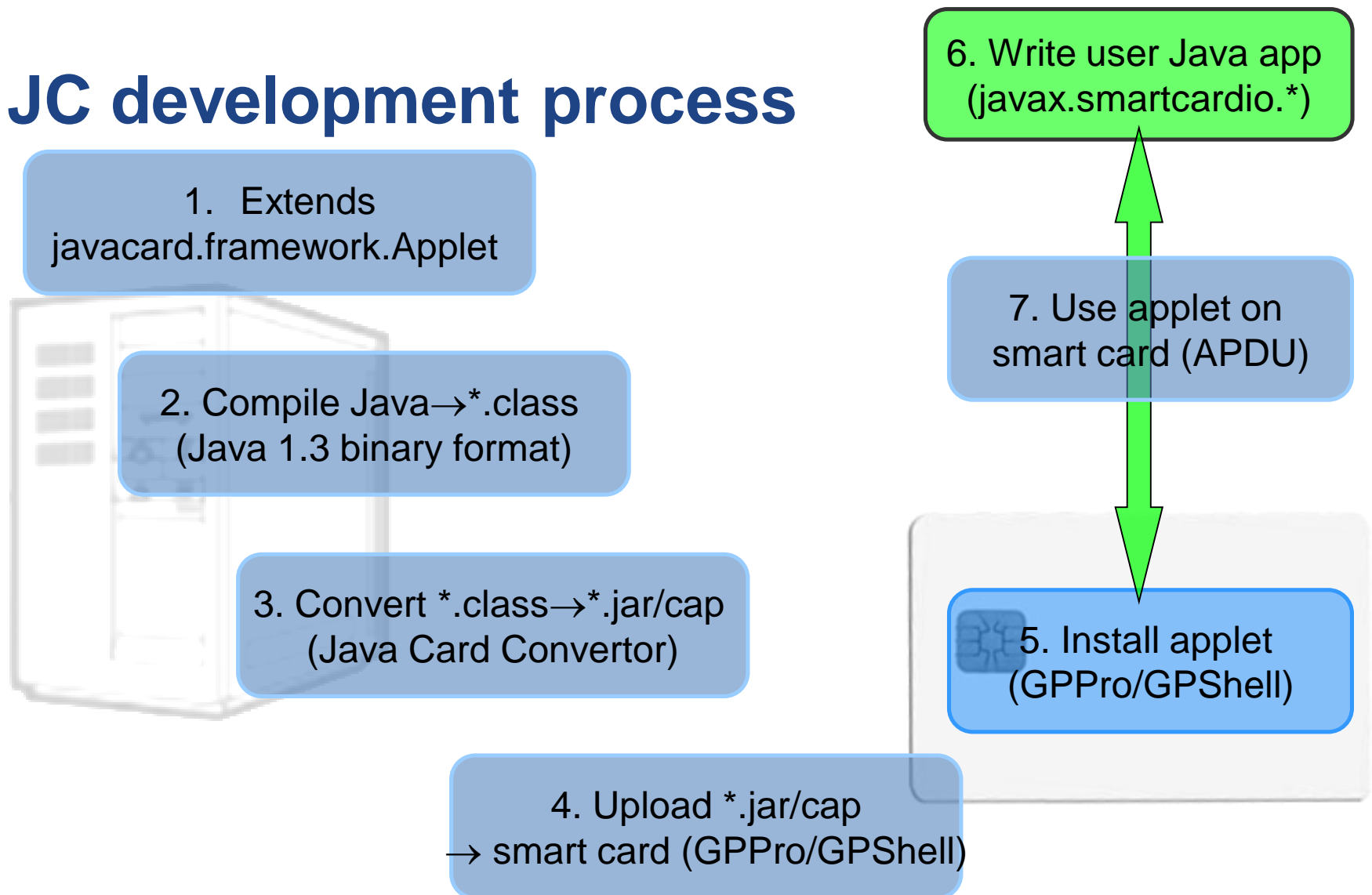
Called repeatedly for
every incoming APDU,
parse and call your
code HERE

```
        else ISOException.throwIt( ISO7816.SW_INS_NOT_SUPPORTED);
```

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

```
}
```

JC development process














Quick start

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
 - Compiles source with Java compiler (javac)
 - Convert with javacard convertor
- (for all projects)

 PLAlD.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=3BF81300008131FE454A434F5076323431B7>

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
 Sequence 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: A00000000620102 v1.3

Import: A0000000620001 v1.0

Cap loaded

- Hint: test with `gpg --card-edit`

OpenPlatform Package/applet upload

- Security domain selection
- Secure channel establishment – security domain
- Package upload
 - Local upload in trusted environment
 - Remote upload with relayed secure channel
- 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
- Example for LabakApplet.java
 - gp –apdu B0541000 (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`)

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
6. Applet is deleted by service command

COMMUNICATION WITH SMART CARD

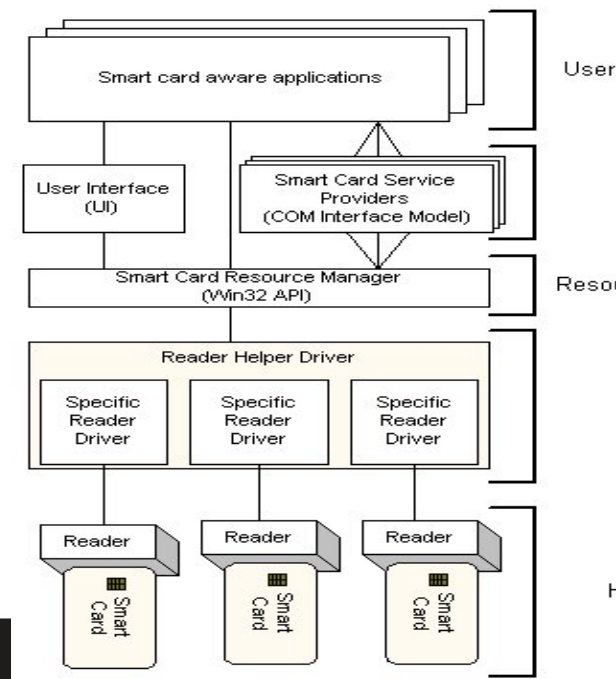
How to communicate with our applet?

1. Various existing tools


- PGP/GPG, S-MIME (PKCS11 lib), signature application...
- Low-level communication (GPShell, GPPro...)

2. Possibility to send APDU from our own program

- PC/SC standard (PC/SC-lite on Linux)
- SCardxxx Win32 API (winscard.dll)
- javax.smartcardio.* API for Java 6



JavaCard communication lifecycle

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

SYMMETRIC CRYPTO APPLET

Random numbers

- `javacard.security.RandomData`
- Two versions of random generator
 - `ALG_SECURE_RANDOM` (truly random)
 - `ALG_PSEUDO_RANDOM` (usually same as `SECURE`)
- 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 (DESKey.setKey())

```
// .... INICIALIZATION SOMEWHERE (IN CONSTRUCT)
// CREATE DES KEY OBJECT
DESKey m_desKey = (DESKey) KeyBuilder.buildKey(KeyBuilder.TYPE_DES,
    KeyBuilder. LENGTH_DES3_3KEY, false);
// Generate from RNG
m_rngRandom.generateData(array, (short) 0,
    (short) KeyBuilder. LENGTH_DES3_3KEY/8);

// SET KEY VALUE
m_desKey.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 – 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);

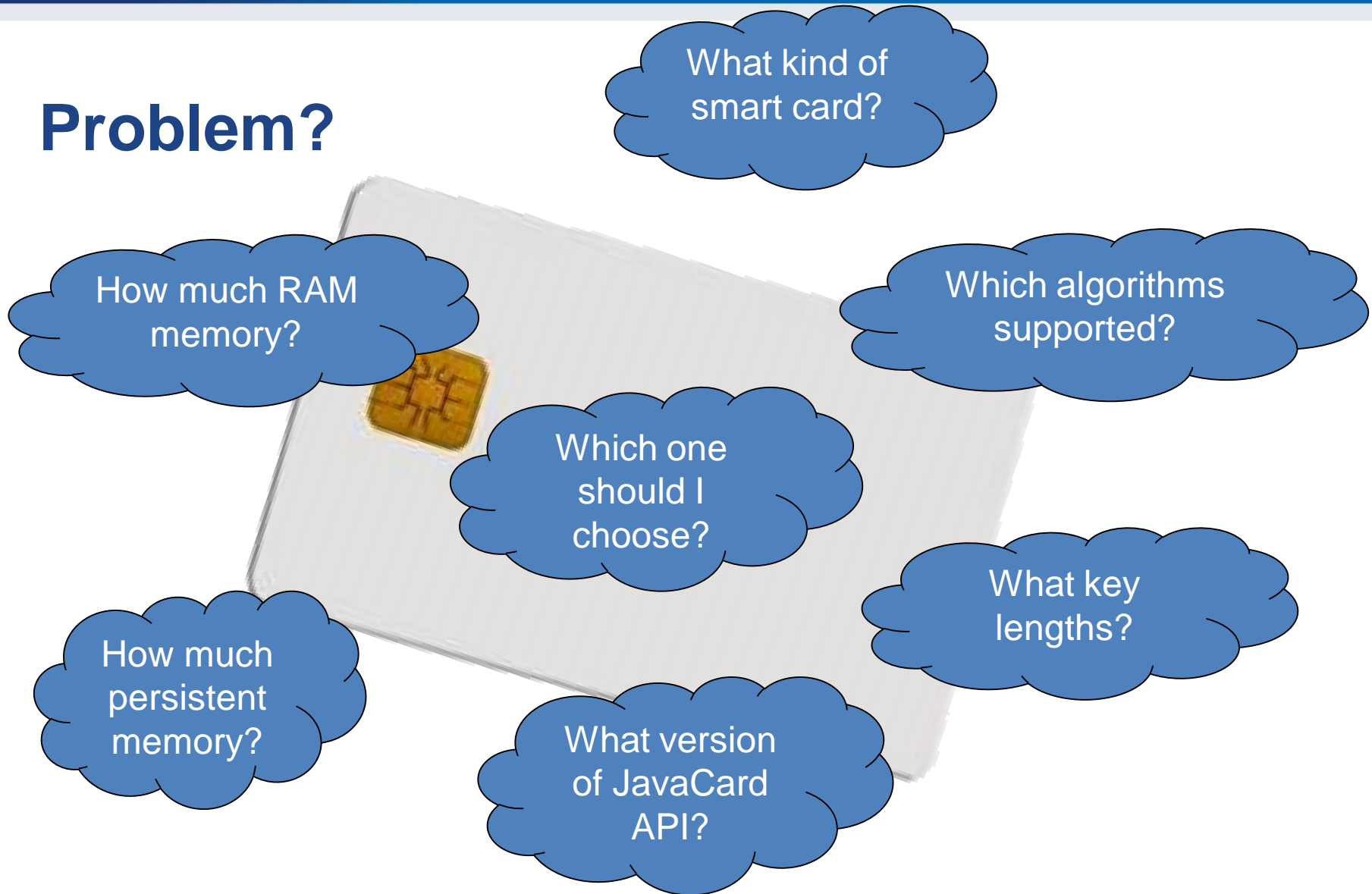
    // COPY ENCRYPTED DATA INTO OUTGOING BUFFER
    Util.arrayCopyNonAtomic(m_ramArray, (short) 0, apdubuf, ISO7816.OFFSET_CDATA, dataLen);

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

Algorithms offered, performance of the current hardware

ALGORITHMS, PERFORMANCE

Problem?



Supported algorithms for JavaCard smart cards

- Same hw sells in several configurations
 - e.g., AES present, but disabled
 - additional software libraries in later versions of card
- ATR alone is not sufficient identification
 - hard to get product description just from ATR
 - ATR can be changed via service command
 - seller not always aware of details
 - <http://smartcard-atr.appspot.com/>
- More details from certification reports like NIST FIPS 140
 - <http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm>

JCAlgTester project (test app&database, 43+)

javacard.crypto.Signature	introduced in JavaCard version	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14
ALG_DES_MAC4_NOPAD	<=2.1	no	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes
ALG_DES_MAC8_NOPAD	<=2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
ALG_DES_MAC4_ISO9797_M1	<=2.1	no	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	no	yes
ALG_DES_MAC8_ISO9797_M1	<=2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
ALG_DES_MAC4_ISO9797_M2	<=2.1	no	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	no	yes
ALG_DES_MAC8_ISO9797_M2	<=2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
ALG_DES_MAC4_PKCS5	<=2.1	no	no	no	yes	yes	no	yes	no	no	no	no	no	no	no	yes
ALG_DES_MAC8_PKCS5	<=2.1	no	no	no	yes	yes	no	yes	no	no	no	no	no	no	no	yes
ALG_RSA_SHA_ISO9796	<=2.1	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
ALG_RSA_SHA_PKCS1	<=2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
ALG_RSA_MD5_PKCS1	<=2.1	no	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	no	yes	yes
ALG_RSA_RIPEDMD160_ISO9796	<=2.1	no	no	no	yes	yes	no	no	no	no	no	no	no	no	no	yes
ALG_RSA_RIPEDMD160_PKCS1	<=2.1	no	no	no	yes	yes	no	no	no	no	no	no	no	no	no	yes
ALG_DSA_SHA	<=2.1	no	no	no	yes	yes	no	no	no	no	no	no	no	no	no	no
ALG_RSA_SHA_RFC2409	<=2.1	no	no	no	yes	yes	no	no	no	no	no	no	no	no	no	yes
ALG_RSA_MD5_RFC2409	<=2.1	no	no	no	yes	yes	no	no	no	no	no	no	no	no	no	yes
ALG_ECDSA_SHA	2.2.0	no	no	no	no	no	no	yes	no	no	no	no	no	no	no	no
ALG_AES_MAC_128_NOPAD	2.2.0	yes	no	yes	yes	yes	yes	yes	yes	yes	no	no	no	no	no	no
ALG_DES_MAC4_ISO9797_1_M2_ALG3	2.2.0	no	no	no	yes	yes	yes	yes	yes	yes	no	no	no	no	no	yes
ALG_DES_MAC8_ISO9797_1_M2_ALG3	2.2.0	no	no	no	yes	yes	yes	yes	yes	yes	no	no	no	no	no	yes
ALG_RSA_SHA_PKCS1_PSS	2.2.0	no	no	no	no	no	no	yes	no	no	no	no	no	no	no	no
ALG_RSA_MD5_PKCS1_PSS	2.2.0	no	no	no	no	no	no	no	no	no	no	no	no	no	no	no

Supported algorithms - summary

- Always supported: TRNG, 3DES, RSA-1024b, SHA-1, MD5, on-card key generation
- Supported by newer cards: AES-128/196/256, RSA-2048b, ECC
- Usually supported by newer cards: SHA2-256, EC-DH key agreement

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)
 - JCSys::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 usually not available
 - do not use new except in constructor
- Keep Cipher or Signature objects initialized
 - if possible (e.g., fixed master key)
 - initialization with key takes non-trivial time
- 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)

Security hints (1)

- Use API algorithms/modes rather than your own
 - API algorithms fast and protected in cryptographic hardware
 - general-purpose processor leaking 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)

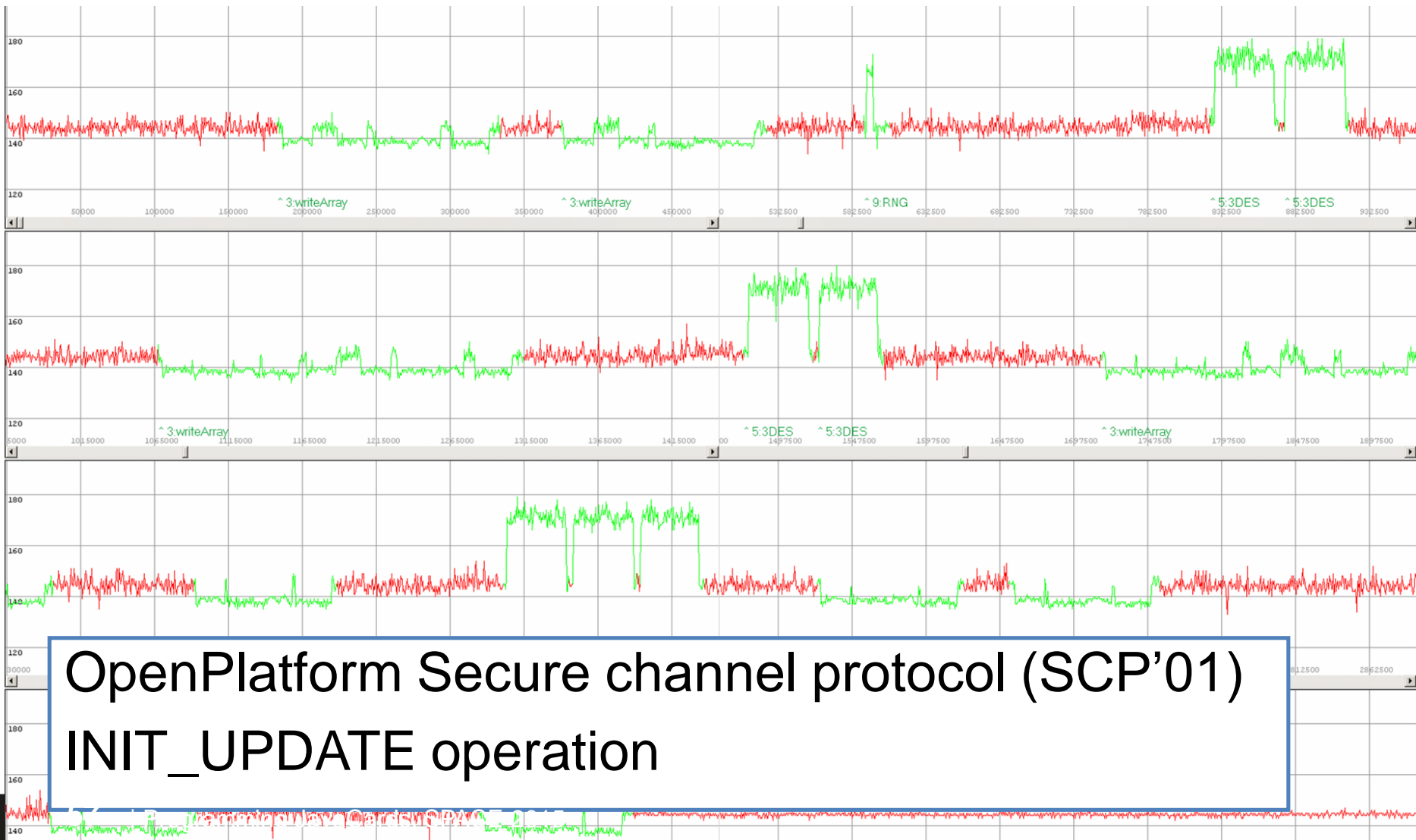
- Erase unused keys and sensitive arrays
 - use specialized method if exists (`Key::clearKey()`)
 - or overwrite with random data (`Random::generate()`)
- Use transactions to ensure atomic operations
 - power supply can be interrupted inside code execution
 - be aware of attacks by interrupted transactions - 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

POWER ANALYSIS

Analyzing implementations



OpenPlatform Secure channel protocol (SCP'01)
INIT_UPDATE operation

Reverse engineering of Java Card bytecode

- Goal: obtain code back from smart card
 - JavaCard defines around 140 bytecode instructions
 - JVM fetch instruction and execute it

(source code)

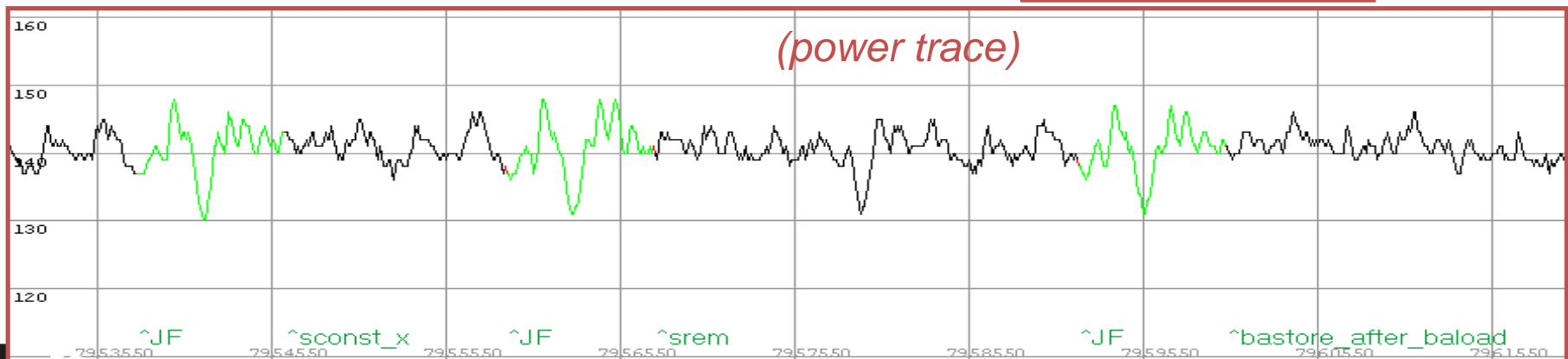
```
m_ram1[0] = (byte) (m_ram1[0] % 1);
```

compiler

(bytecode)

```
getfield_a_this 0;  
sconst_0;  
baload;  
sconst_1;  
srem;  
bastore;
```

oscilloscope



Conditional jumps

- may reveal sensitive info
- keys, internal branches...

(source code)

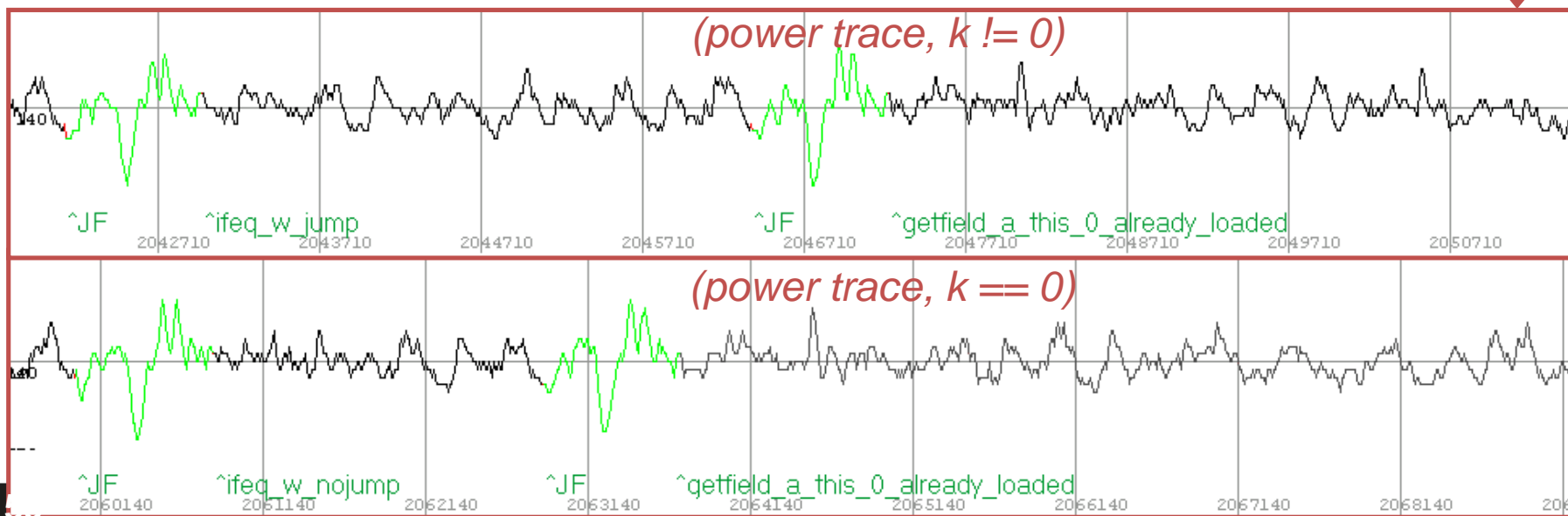
```
if (key == 0) m_ram1[0] = 1;
else m_ram1[0] = 0;
```

compiler →

(bytecode)

```
sload_1;
ifeq_w L2;
L1: getfield_a_this 0;
sconst_0;
sconst_0;
bastore;
goto L3;
L2: getfield_a_this 0;
sconst_0;
sconst_1;
bastore;
goto L3;
L3: ...
```

oscilloscope



Thank you for your attention!

Questions 

