

PV204 Security technologies



JavaCard - programming secure elements

Petr Švenda  svenda@fi.muni.cz  [@rngsec](https://twitter.com/rngsec)

Centre for Research on Cryptography and Security, Masaryk University

CRCS

Centre for Research on
Cryptography and Security

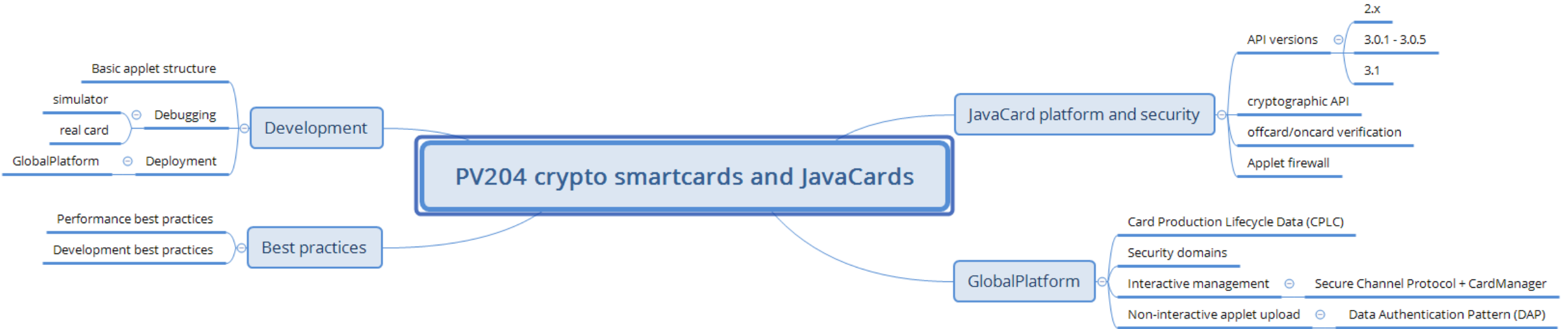
Please comment on slides with anything unclear, incorrect or suggestions for improvement
https://drive.google.com/file/d/1vHkqpJH3MU0zjJ3FQFuNiJbNUV_9VgHB/view?usp=sharing

www.fi.muni.cz/crocs

Project (Tonda)

Prerequisites

- Knowledge of basic smartcards technology is assumed (PV079)
- If you are not familiar yet, please read slides PV204_03___PV079_2021_smartcards.pdf from IS (uploaded for this course)



Motivation

- Usage security-relevant scenarios
 - Subscriber modules (SIMs), merchant payments, hardware wallets, authentication tokens, electronic IDs...
- Why not as another software application on your laptop?
 - Laptop not well portable, large trusted code base, many other applications (malware), lack of secure storage for cryptographic keys, user/attacker control platform, expensive to own...
- Mobile phone fixes only some of these issues
 - Is portable, some have better platform security (but not all!), still somewhat expensive...

Properties of “Ideal” platform

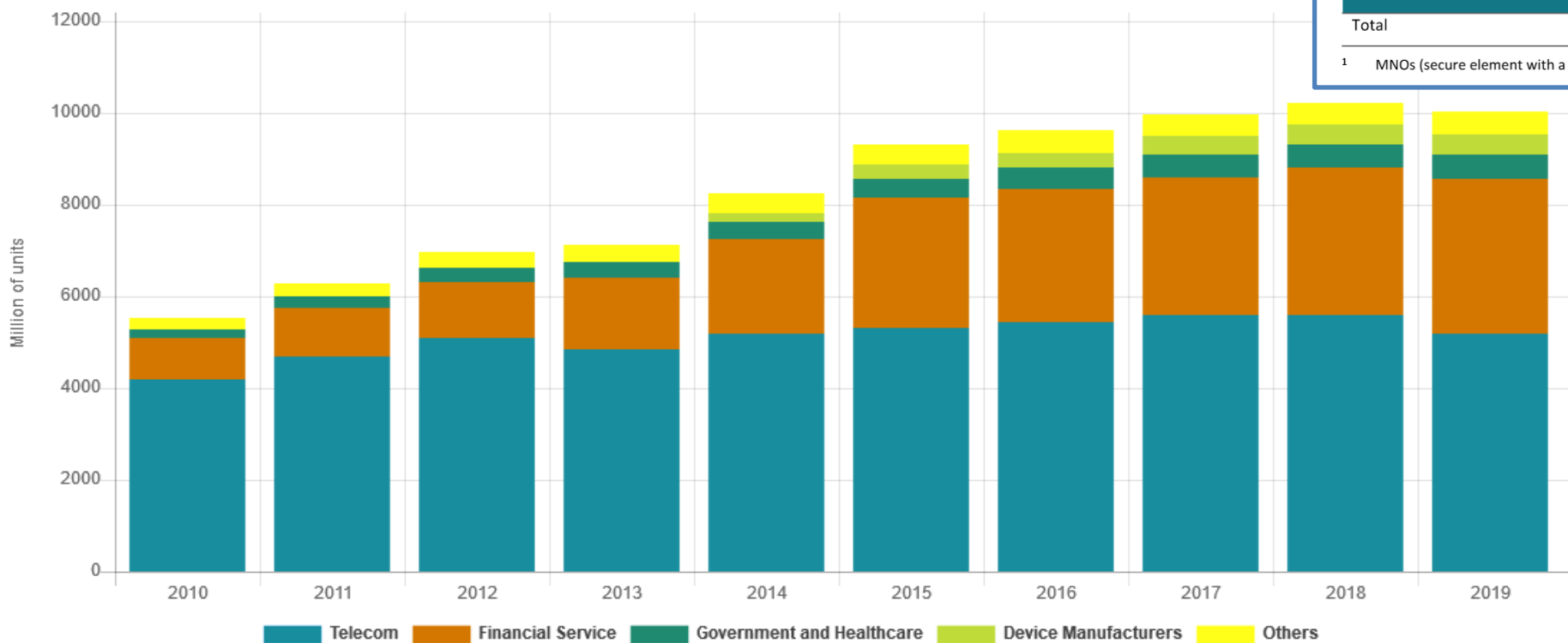
- Cheap, portable, no battery
- Good support from outer environment
- Fast enough for a task
- Easy to develop (securely)
- Apps portable between platform manufacturers
- Secure, even with physical access (keys extraction)
- Multiple apps from distrusting providers securely
- Secure remote management (new apps, update)
- ...

Technology

crypto smartcards
PC/SC, phones with NFC
main CPU + crypto coprocessors
JavaCard API, tools, best practices
JavaCard bytecode, JCVM
tamper resistant, CC, FIPS140-2/3
Applet firewall, Security Domains
GlobalPlatform, SCP, DAP

Primary markets for smartcards

Secure Elements Shipments From 2010 To 2019



Eurosmart estimated WW μ P market size - (Mu)

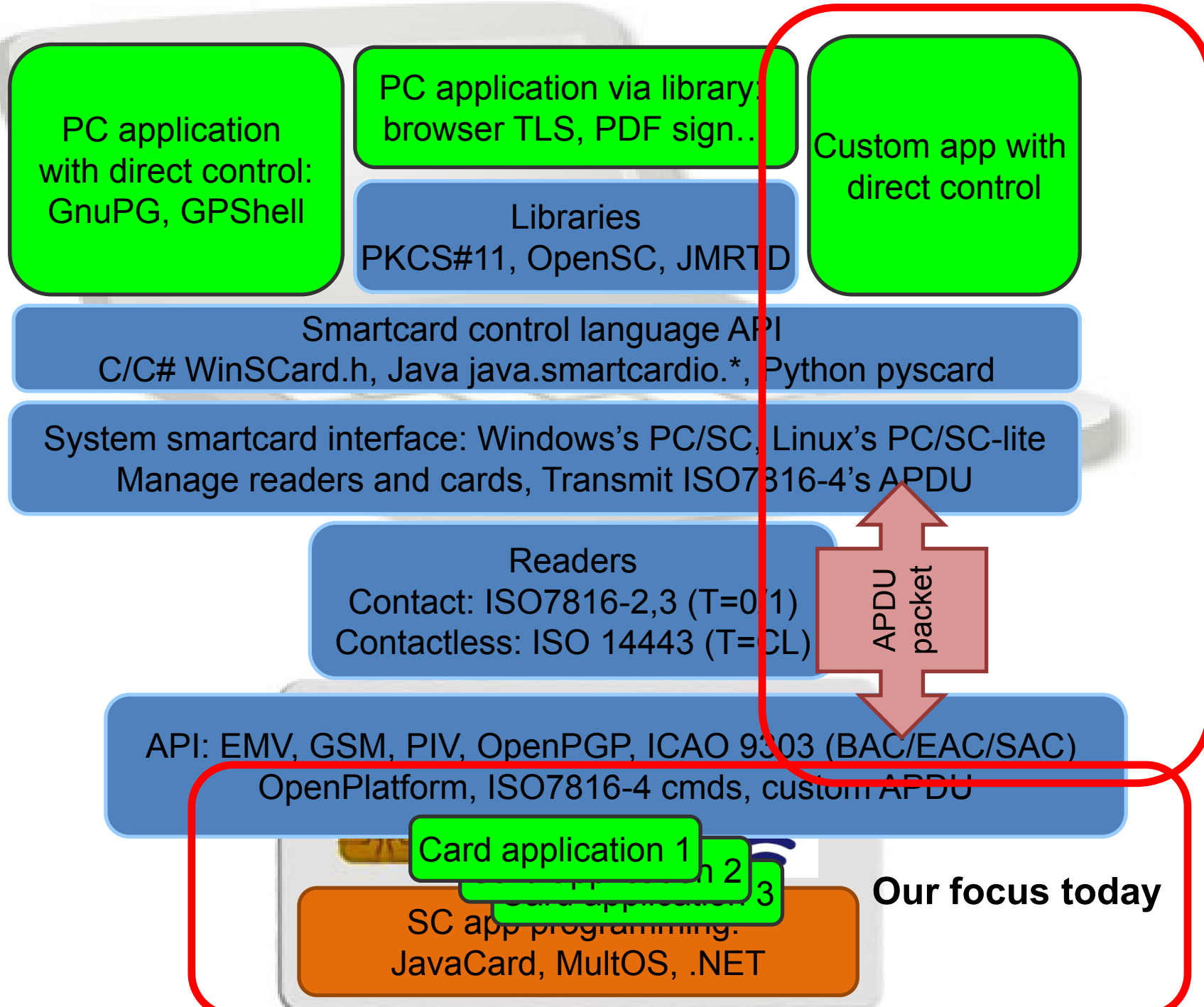
| | 2020 | 2021 |
|-----------------------------------|-------------|-------------|
| Telecom ¹ | 5100 | 4900 |
| Financial services | 3170 | 3250 |
| Government- Healthcare | 425 | 490 |
| Device manufacturers ² | 450 | 490 |
| Transport | 230 | 220 |
| Pay-TV | 75 | 65 |
| Others ³ | 90 | 90 |
| Total | 9540 | 9505 |

¹ MNOs (secure element with a SIM application)

<https://www.eurosmart.com/eurosmarts-secure-elements-market-analysis-and-forecasts/>

Old vs. current multi-application smart cards

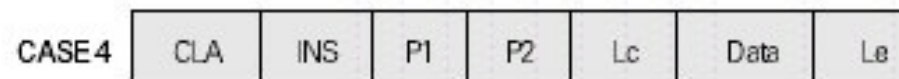
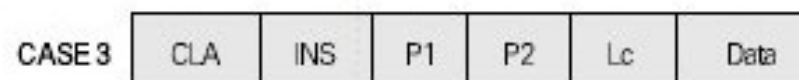
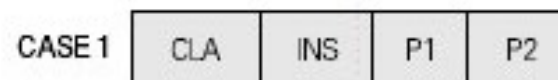
- One program only
- Stored persistently in ROM or EEPROM
- Written in machine code
 - Chip specific
- Multiple applications at the same time
- Stored in EEPROM
- Written in higher-level language
 - Interpreted from bytecode
 - Portable
- Application can be later managed (remotely)



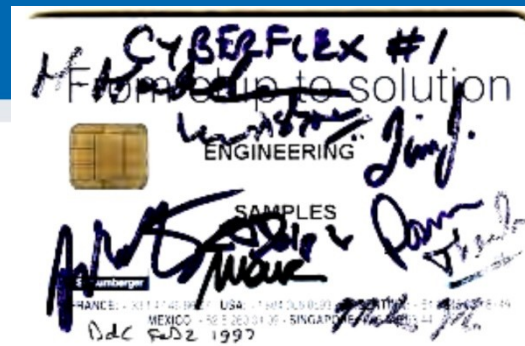
APDU (Application Protocol Data Unit)

- APDU is basic logical communication datagram
 - header (5 bytes) and up to ~256 bytes of user data
- Format specified in ISO7816-4
- Header/Data 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
- Some values of CLA/INS/P1/P2 standardized (better interoperability)
 - <https://web.archive.org/web/20180721010834/http://techmeonline.com/most-used-smart-card-commands-apdu/>
- Custom values used by application developer (your own API)

| Command | Function | Instruction (INS) | Standard |
|---------------------|--|-------------------|----------------|
| ACTIVATE FILE | Reversibly unblock a file. | '44' | ISO/IEC 7816-9 |
| APPEND RECORD | Insert a new record in a file with a linear fixed structure. | 'E2' | ISO/IEC 7816-4 |
| APPLICATION BLOCK | Reversibly block an application. | '1E' | EMV |
| APPLICATION UNBLOCK | Unblock an application. | '18' | EMV |
| ASK RANDOM | Request a random number from the smart card. | '84' | EN 726-3 |

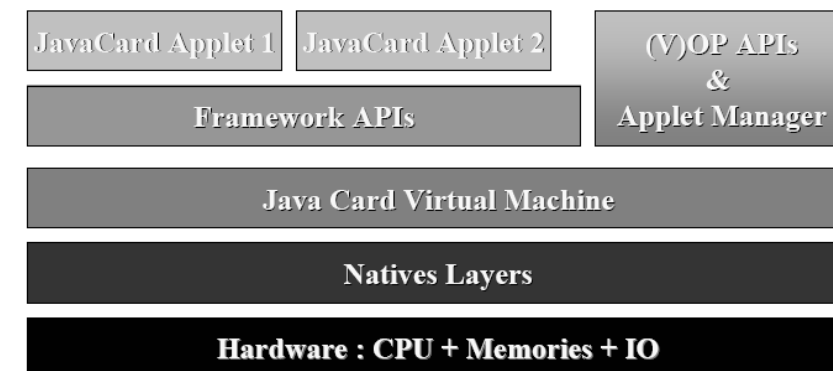


| | | | |
|------------------|---|------|----------------|
| | | '24' | TS 51.011 |
| | | '24' | ISO/IEC 7816-8 |
| | | 'AC' | EN 726-3 |
| | | '56' | EN 1546-3 |
| | | 'E0' | ISO/IEC 7816-9 |
| | | 'E2' | EN 726-3 |
| | | '52' | EN 1546-3 |
| CREDIT PSAM | Pay from IEP to the PSAM. | '72' | EN 1546-3 |
| DEACTIVATE FILE | Reversibly block a file. | '04' | ISO/IEC 7816-9 |
| DEBIT IEP | Pay from the purse | '54' | EN 1546- |
| DECREASE | Reduce the value of a counter in a file. | '30' | EN 726-3 |
| DECREASE STAMPED | Reduce the value of a counter in a file that is stamped using a | '34' | EN 726-3 |



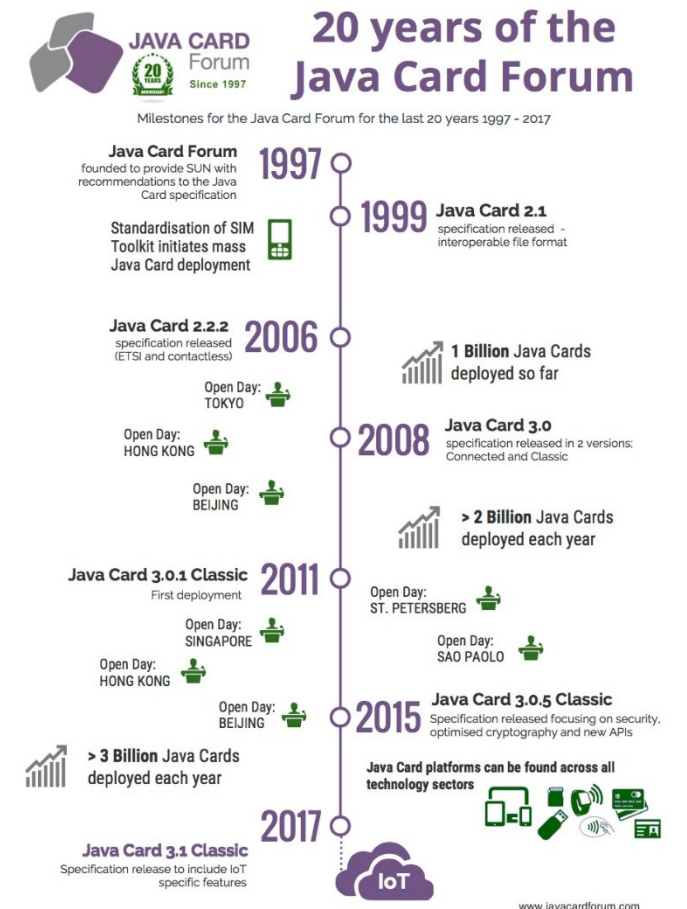
JavaCard

- 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



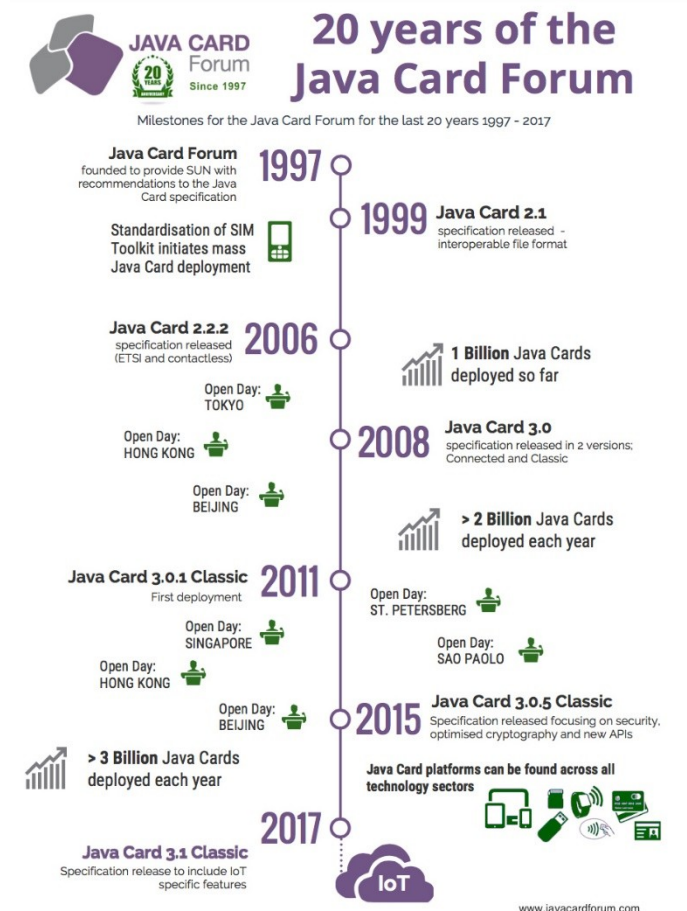
JavaCard 2.x is like Java but not supporting...

- 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 by default
 - specialized package `javacardx.framework.util.intx` for support
- 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



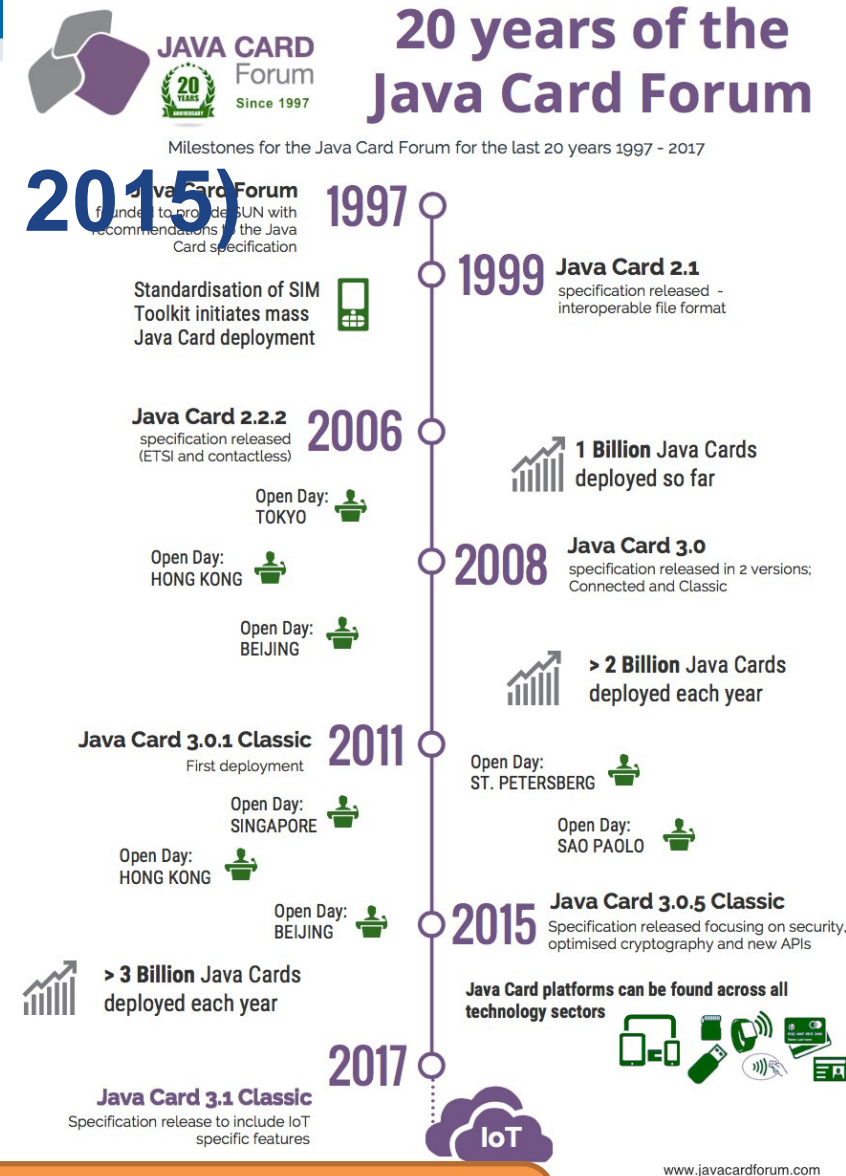
JavaCard 2.x supports

- 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)

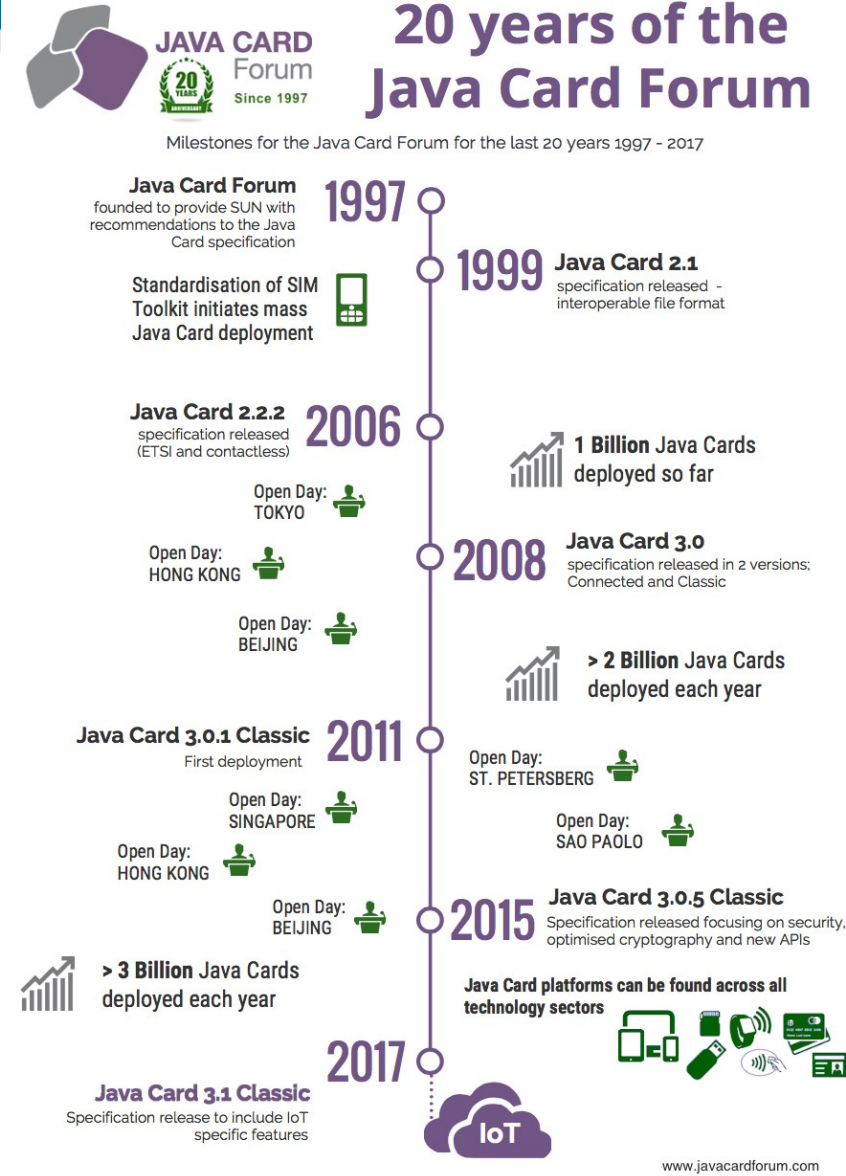
- Major release of JavaCard specification
 - significant changes in development logic
 - two separate branches – Classic and Connected edition
- JavaCard 3.x Classic Edition
 - legacy version, extended JC 2.x
 - APDU-oriented communication
- JavaCard 3.x Connected Edition
 - smart card perceived as web 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 than classic smart cards



Connected edition is not used so far (likely dead)

JavaCard 3.1 (2018)

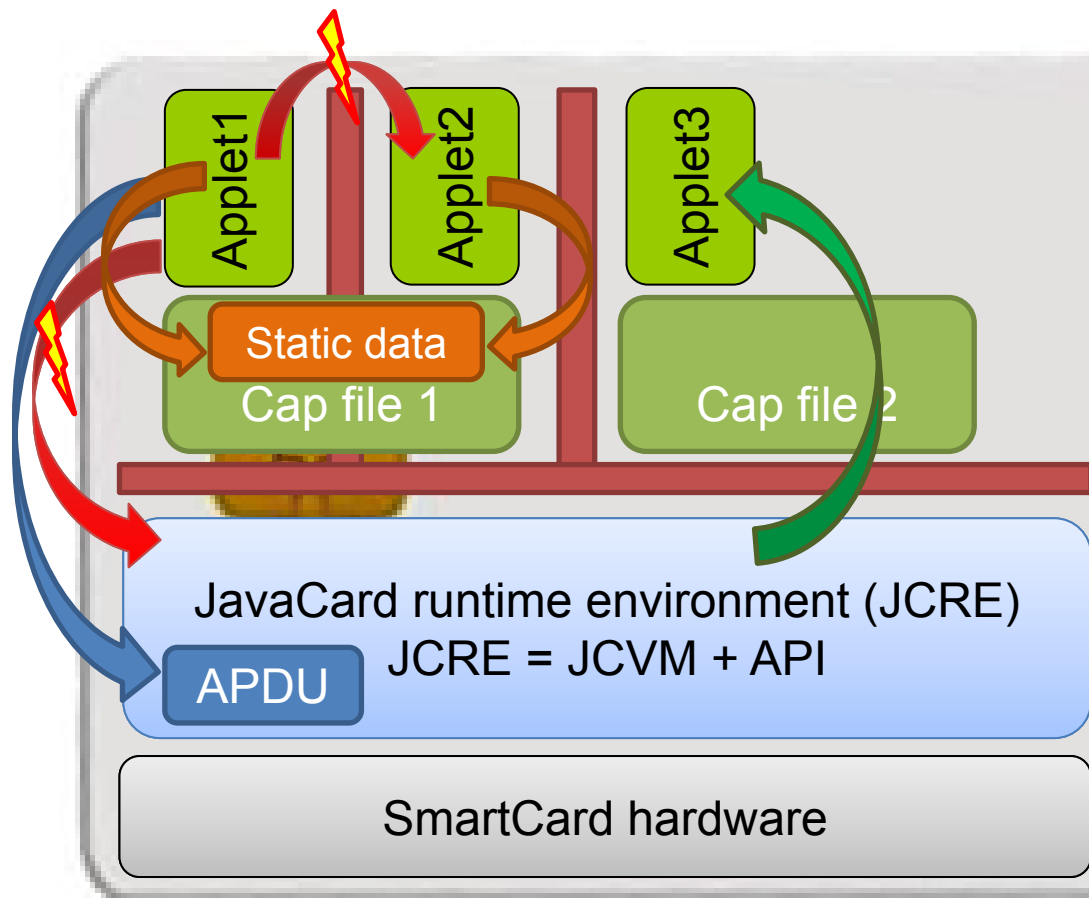
- Focus on IoT
- Not much experience yet (no devices)
- Additional cryptographic algorithms, named curves.



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 (somewhat) obtained from card
 - JCSystem.getVersion() → [Major.Minor]
 - <https://github.com/petrs/jcAIDScan>
 - See <https://www.fi.muni.cz/~xsvenda/jcsupport.html>
- Available cards supports mostly 2.x specification or 3.0.x (newer cards)

JavaCard applet firewall – runtime checks

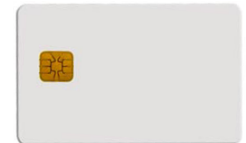


- **Access** to other applet's methods and attributes **prevented**
 - Even if **public**
- Applets can access specific JCRC objects
- JCRC can access all applets (no restriction)
- **Static attributes** of package accessible by all its applets!

Inspired by http://ekldata.com/IHWNXUB-yernbID2sdiK1zxxQco/5_javacard.pdf

On-card, off-card code verification

- How to upload only “correct” applets?
- 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 (and enforced by DAP – shown later)
- On-card verification
 - Limited resources available
 - Proprietary checks by JC platform implementation



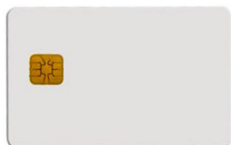
DEVELOPING JAVACARD APPS

Desktop vs. smart card

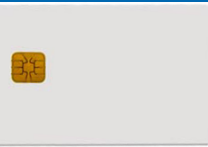
- Following slides will be marked with icon based on where it is executed



Process executed on host (PC/NTB...)



Process executed inside smart card



```

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 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);
        }
        else ISOException.throwIt( ISO7816.SW_INS_NOT_SUPPORTED);
    }
    public void MyMethod(APDU apdu) { /* ... */ }
}

```

include packages from
javacard.*

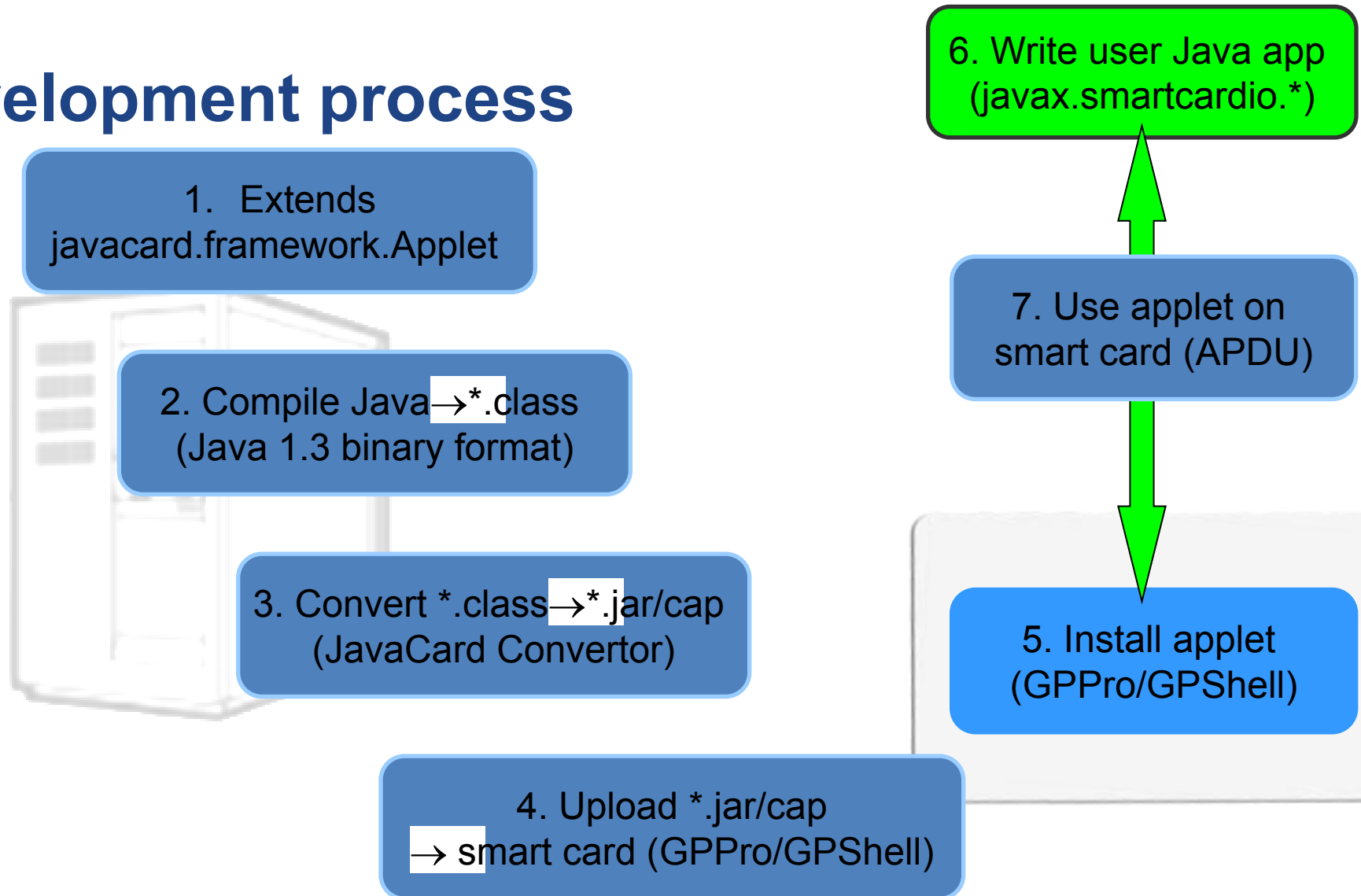
extends Applet

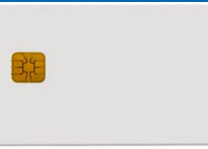
Called only once, do
all allocations&init
HERE

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





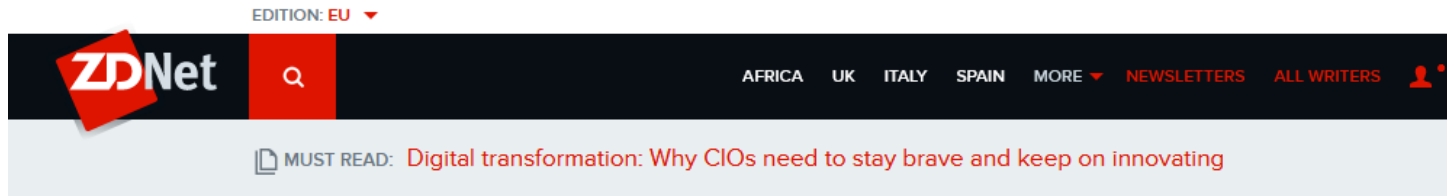
JavaCard application running model

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

Managing applets on card



Motivation: Fix bug in electronic IDs for half of population



Estonia's ID card crisis: How e-state's poster child got into and out of trouble

Estonia is built on secure state e-systems, so the world was watching when it hit a huge ID-card problem.

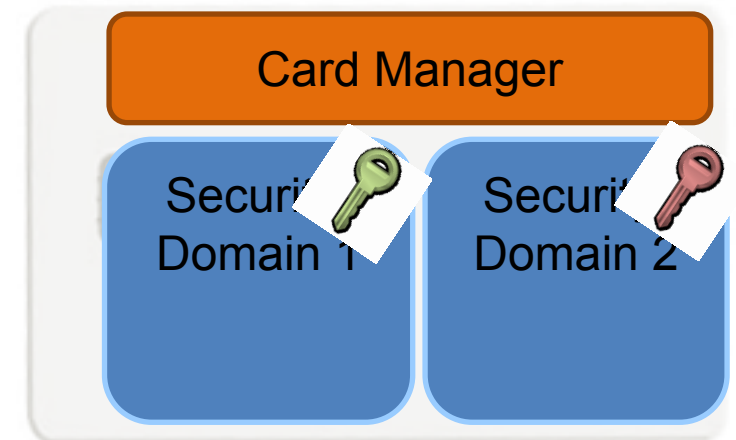
- Problem: how to remotely manage administrative access to token?
 - Smartcards, TEE (TrustZone) - same basic issues, but also some specifics
- Local/remote upload, configuration and removal of applications
- Authentication of manager, online vs. offline operations

GlobalPlatform

- Specification of API for card administration
 - Upload/install/delete applications
 - Card lifecycle management
 - Card security management
 - Security mechanisms and protocols
- Newest is GlobalPlatform Card Specification v2.3.1
 - March 2018
 - Previous versions also frequently used
 - <http://www.globalplatform.org/specificationscard.asp>
- Primary open API for Trusted Execution Environment (TEE)
 - ARM TrustZone...

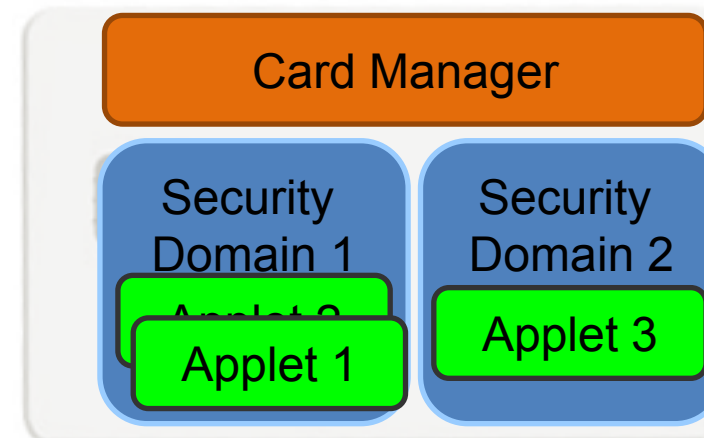
GlobalPlatform – main terms

- Smart card life cycle
 - OP_READY, INITIALIZED (prepared for personalization)
 - SECURED (issued to user, use phase)
 - CARD_LOCKED (temporarily locked (attack), unlock to SECURED)
 - TERMINATED (logically destroyed)
- Card Manager (CM)
 - Special card component responsible for administration and card system service functions (cannot be removed)
- Security Domain (SD)
 - Logically separated area on card with own access control
 - Enforced by different authentication keys



GlobalPlatform – main terms

- Card Content (apps,data) Management
 - Content verification, loading, installation, removal
- Security Management
 - Security Domain locking, Application locking
 - Card locking, Card termination
 - Application privilege usage, Security Domain privileges
 - Tracing and event logging
- Command Dispatch
 - Application selection
 - (Optional) Logical channel management



Smart card life cycles

- OP_READY – card is ready for uploading of key diversification data, any application and issuer specific structures.
- INITIALIZED – card is fully prepared but not yet issued to card holder.
- SECURED – card is issued to card holder. Card management is possible only through Security domain (installation of signed applets etc.).
- CARD_LOCKED – card is locked due to some security policy and no data management can be performed. Card can be locked by Security domain and later unlocked as well (switch back to SECURED state).
- TERMINATED – card is logically “destroyed” due to card expiration or detection of the severe security thread.

Card Production Life Cycle (CPLC)

- Manufacturing metadata
- Dates (OS, chip)
- Circuit serial number
- (not mandatory)
- GlobalPlatform APDU
 - 80 CA 9F 7F 00
 - gppro --info
- ISO7816 APDU
 - 00 CA 9F 7F 00

CPLC info

IC Fabricator: 4790

IC Type: 5167

OS ID: 4791

OS Release Date: 2081

OS Release Level: 3b00

IC Fabrication Date ((Y DDD) date in that year): 4126

IC Serial Number: 00865497

IC Batch Identifier: 3173

IC Module Fabricator: 4812

IC Module Packaging Date: 4133

IC Manufacturer: 0000

IC Embedding Date: 0000

IC Pre Personalizer: 1017

IC Pre Personalization Equipment Date: 4230

IC Pre Personalization Equipment ID: 38363534

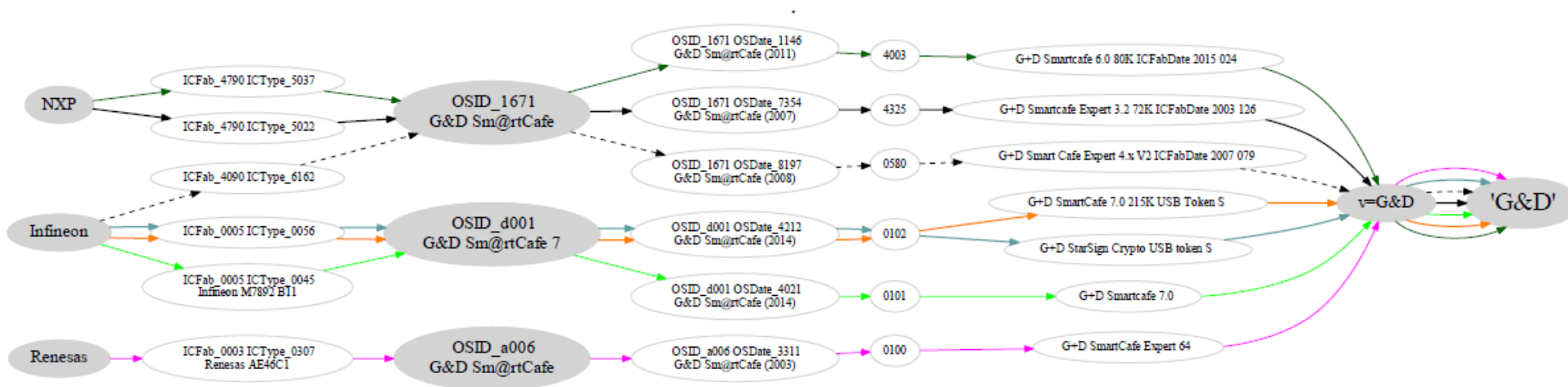
IC Personalizer: 0000

IC Personalization Date: 0000

IC Personalization Equipment ID: 00000000

Example CPLC results from several G&D cards

ICFabricator → ICFab_ICType → OperatingSystemID → OperatingSystemID_OSReleaseDate → OSReleaseLevel → CardName → Original vendor → Current vendor





GlobalPlatform package/applet upload - SCP

- A. Security domain selection
 - B. Secure channel establishment (SCP) – 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
- **`gp --install file_with_applet.cap`**

GlobalPlatform package/applet upload - Data Authentication Pattern (DAP)

- Generate cap signing keypair (RSA, OpenSSL)
- Sign applet (file with cap, capfile tool)
- Create policy domain (SSD) with MandatedDAPVerification
- Set personalization keys for the SSD (secret symmetric crypto keys)
- Upload verification key for this domain (key version 0x73, public key of your signing keypair)
- Verify that SSD is prepared (DOM, DAPVerification privilege)
- Upload signed applet (*.cap file)
- <https://github.com/martinpaljak/GlobalPlatformPro/blob/next/tests/sce70.sh>

DEBUGGING APPLET

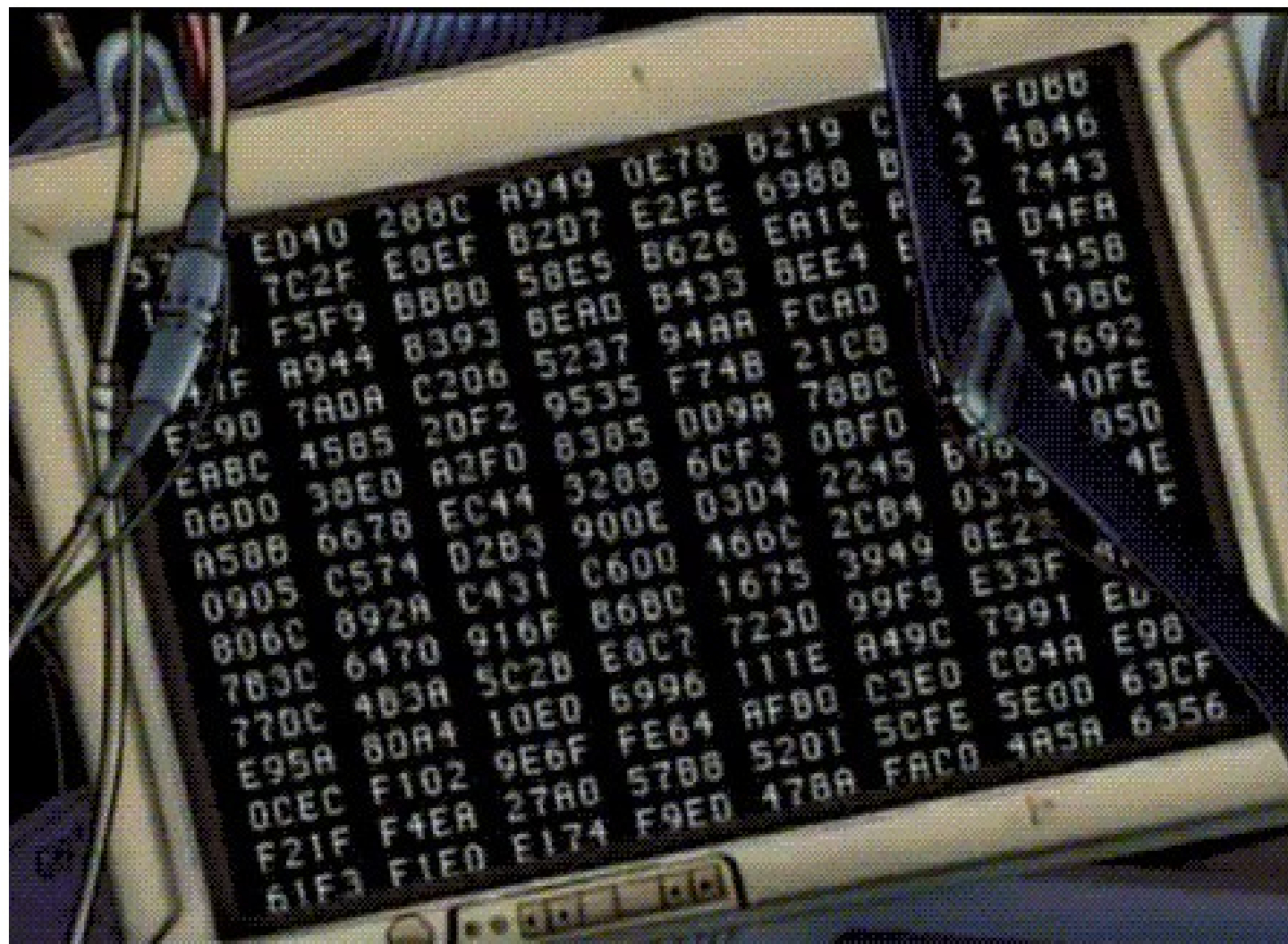
**Martin Paljak**

@martinpaljak

Following



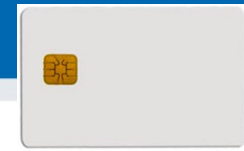
How does smart card programming look like in real life? Here's a typical scenario...





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 😊

```

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_SystemException_prefix | e.getReason()));
    } catch (PINException e) {
        ISOException.throwIt((short) (SW_PINException_prefix | e.getReason()));
    } catch (TransactionException e) {
        ISOException.throwIt((short) (SW_TransactionException_prefix | e.getReason()));
    } catch (CardRuntimeIOException e) {
        ISOException.throwIt((short) (SW_CardRuntimeIOException_prefix | e.getReason()));
    } catch (Exception e) {
        ISOException.throwIt(Constants.SW_Exception_prefix | e.getReason());
    }
}

```

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_CardRuntimeIOException_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!

**NEXT WEEK:
BEST PRACTICES FOR JAVACARD
(SECURE MULTIPARTY COMPUTATION)**

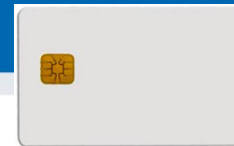
Summary

- 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

BEST PRACTICES (FOR APPLET DEVELOPERS)

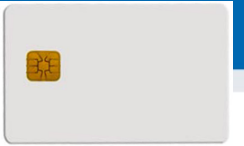
Quiz

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...



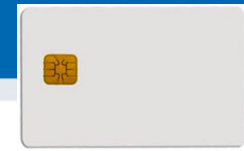
Security hints (1)

- 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)



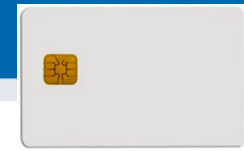
Security hints (2)

- 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



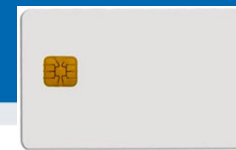
Security hints (4)

- 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 `NullPointerException`s...



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();
}
```



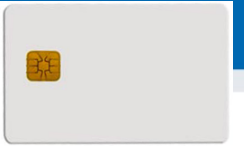
Security hints: fault induction (3)

- 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

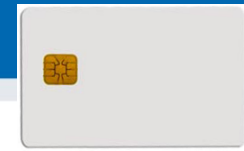
- ✓ Does the device need protection?
- ✓ Understand the resistance of the hardware
- ✓ Identify potential weakness in design
- ✓ Select patterns to use
- ✓ Understand your compiler
- ✓ Code it
- ✓ Test the resistance of the device

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

JCPROFILERNEXT – PERFORMANCE PROFILING, NON-CONSTANT TIME DETECTION

JCProfilerNext: on-card performance profiler

- Open-source on-card performance profiler (L. Zaoral)
 - <https://github.com/lzaoral/JCProfilerNext>
- Automatically instrumentation of provided JavaCard code
 - Conditional exception emitted on defined line of code
 - Spoon tool used <https://spoon.gforge.inria.fr/>
- Measures time to reach specific line (measured on client-side)
- Fully automatic, no need for special setup (only JavaCard + reader)
- Goals:
 - Help developer to identify parts for performance optimizations
 - Help to detect (significant) timing leakages
 - Insert “triggers” visible on side-channel analysis
 - Insert conditional breakpoints...

Instrumented code (Spoon)

```
private void example(APDU apdu) {
```

```
    PM.check(PMC.TRAP_example_Example_example_argb_javacard_framework_APDU_arge_1);
    short count = Util.getShort(apdu.getBuffer(), ISO7816.OFFSET_CDATA);
    PM.check(PMC.TRAP_example_Example_example_argb_javacard_framework_APDU_arge_2);
    for (short i = 0; i < count; i++) {
        PM.check(PMC.TRAP_example_Example_example_argb_javacard_framework_APDU_arge_3);
        short tmp = 0;
        PM.check(PMC.TRAP_example_Example_example_argb_javacard_framework_APDU_arge_4);
        for (short k = 0; k < 50; k++) {
            PM.check(PMC.TRAP_example_Example_example_argb_javacard_framework_APDU_arge_5);
            tmp++;
            PM.check(PMC.TRAP_example_Example_example_argb_javacard_framework_APDU_arge_6);
        }
        PM.check(PMC.TRAP_example_Example_example_argb_javacard_framework_APDU_arge_7);
    }
    PM.check(PMC.TRAP_example_Example_example_argb_javacard_framework_APDU_arge_8);
}
```

```
// if m_perfStop equals to stopCondition, exception is thrown (trap hit)
public static void check(short stopCondition) {
    if (PM.m_perfStop == stopCondition) {
        ISOException.throwIt(stopCondition);
    }
}
```

JCProfilerNext – timing profile of target line of code

example.Example.example2(javacard.framework.APDU)

TRAP_example_Example_example2_argb_javacard_framework_APDU_argc_12

Card ATR: [3BFA1800008131FE454A434F50333315632333298](#)
 Number of rounds: 1000
 APDU header: 80010000
 Input regex: 00[0-9A-F]{2}
 Elapsed time: 0 days 00:00:02.814
 Source measurements: [measurements.csv](#)

Show explicit traps

```

1 private void example2(APDU apdu) {
2     byte[] apdubuf = apdu.getBuffer();
3     short dataLen = apdu.setIncomingAndReceive();
4     // SET KEY VALUE
5     m_aesKey.setKey(apdubuf, ISO7816.OFFSET_CDATA);
6     // INIT CIPHERS WITH NEW KEY
7     m_encryptCipher.init(m_aesKey, Cipher.MODE_ENCRYPT);
8     m_decryptCipher.init(m_aesKey, Cipher.MODE_DECRYPT);
9     m_encryptCipher.doFinal(apdubuf, ISO7816.OFFSET_CDATA, ((short) (0x10)), m_ramArray, ((short) (0)));
10    m_decryptCipher.doFinal(apdubuf, ISO7816.OFFSET_CDATA, ((short) (0x10)), m_ramArray, ((short) (0)));
11    // Hash input
12    m_hash.doFinal(apdubuf, ISO7816.OFFSET_CDATA, dataLen, m_ramArray, ((short) (0)));
13    // GENERATE DATA
14    m_secureRandom.generateData(apdubuf, ISO7816.OFFSET_CDATA, ((short) (0x10)));
15    // Sign data
16    short signLen = m_sign.sign(apdubuf, ISO7816.OFFSET_CDATA, ((byte) (dataLen)), m_ramArray, ((byte) (0)));
17    // Generate fresh key pair on-card
18    m_keyPair.genKeyPair();
19    m_publicKey = m_keyPair.getPublic();
20    m_privateKey = m_keyPair.getPrivate();
21    // INIT WITH PRIVATE KEY
22    m_sign.init(m_privateKey, Signature.MODE_SIGN);
23 }
    
```

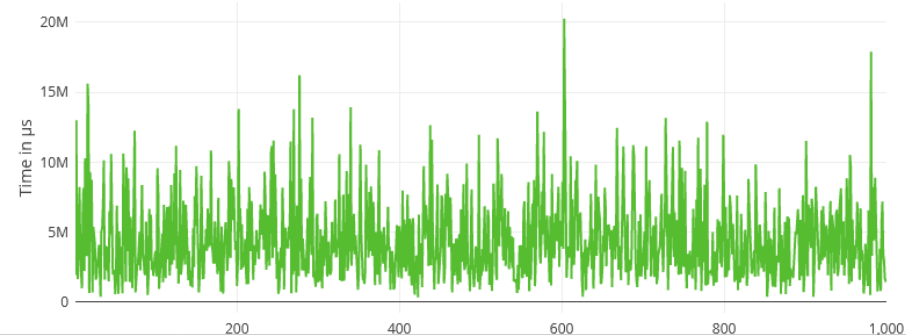
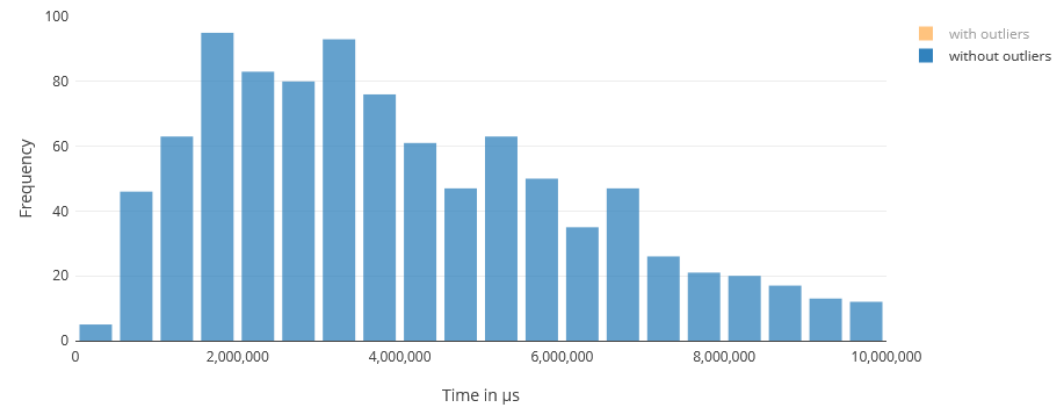
Colour explanation

Currently selected trap

Trap was never reached

Trap was reached only sometimes

Click on a bin to get a list of corresponding inputs.



JCProfilerNext – memory consumption

opencrypto.jcmathlib.OCUnitTests()

TRAP_opencrypto_jcmathlib_OCUnitTests_argb_arge_6

Mode: memory

Card ATR: [3B80800101](#)

APDU header: measured during installation

Input: measured during installation

Elapsed time: 0 days 00:00:00.294

Source measurements: [measurements.csv](#)

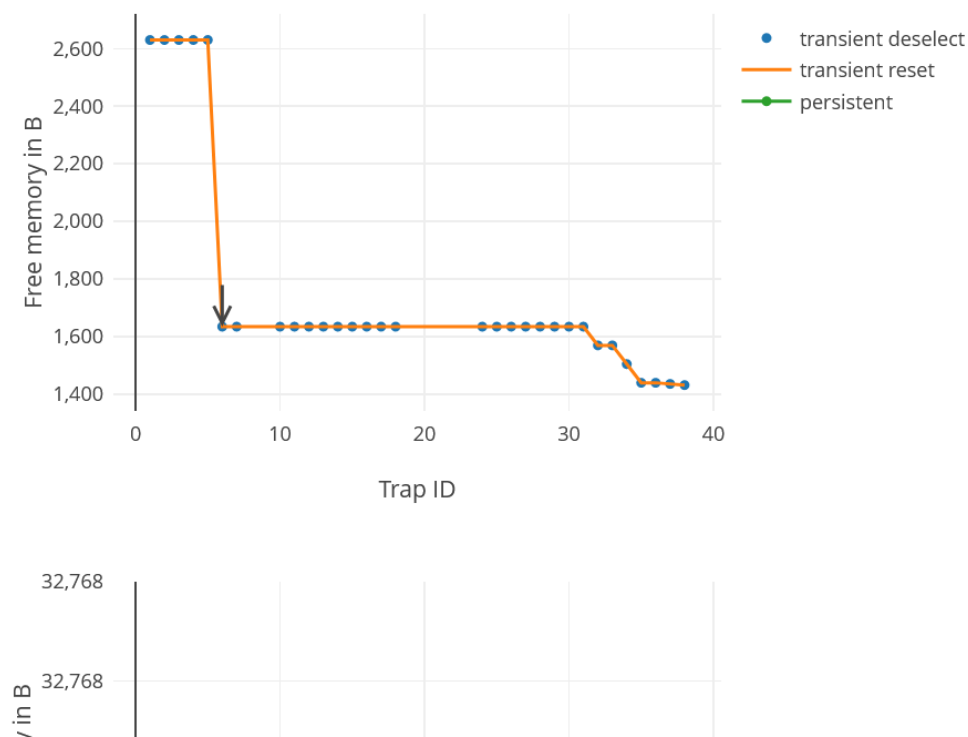
Show explicit traps

Diff in B

```

1 public OCUnitTests() {
2     OperationSupport.getInstance().setCard(OperationSupport.SIMULATOR); // Tr
3     m_memoryInfo = new short[ ((short) (7 * 3)) ]; // Contains RAM and EEPROM i
4     m_memoryInfoOffset = snapshotAvailableMemory( ((short) (1)), m_memoryInfo
5     if (bTEST_256b_CURVE) {
6         m_ecc = new ECCConfig( ((short) (256)) );
7     }
8     if (bTEST_512b_CURVE) {
9         m_ecc = new ECCConfig( ((short) (512)) );
10    }
11    m_memoryInfoOffset = snapshotAvailableMemory( ((short) (2)), m_memoryInfo
12    // Pre-allocate test objects (no new allocation for every tested operat
13    if (bTEST_256b_CURVE) {
14        m_testCurve = new ECCurve(false, SecP256r1.p, SecP256r1.a, SecP256r
15        m_memoryInfoOffset = snapshotAvailableMemory( ((short) (3)), m_memory
16        // m_testCurveCustom and m_testPointCustom will have G occasionally
17        m_customG = new byte[ ((short) (SecP256r1.G.length)) ];
18        Util.arrayCopyNonAtomic(SecP256r1.G, ((short) (0)), m_customG, ((sh
19        m_testCurveCustom = new ECCurve(false, SecP256r1.p, SecP256r1.a, Sec
20    }
21    if (bTEST_512b_CURVE) {

```



JCProfilerNext – checking for non-constant behavior

opencrypto.jcmlib.OCUnitTests#test_BN_MOD(javacard.framework.APDU,short)

TRAP_opencrypto_jcmlib_OCUnitTests_hash_test_BN_MOD_argb_javacard_framework_APDU__short_arge_10

Mode: time

Card ATR: [3B80800101](#)

Number of rounds: 1000

APDU header: B0252100

Input regex: 00[0-9A-F]{64}[4-7][0-9A-F]{63}

Elapsed time: 0 days 00:58:39.803

Source measurements: [measurements.csv](#)

Show explicit traps

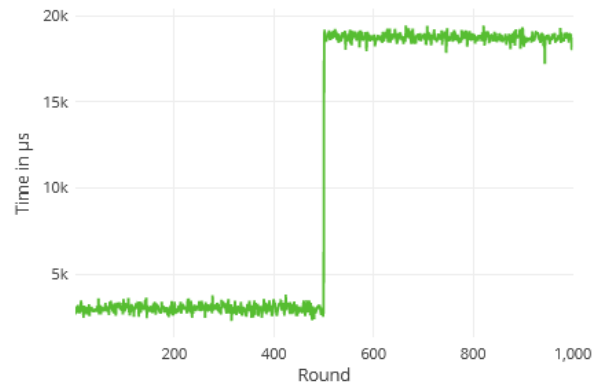
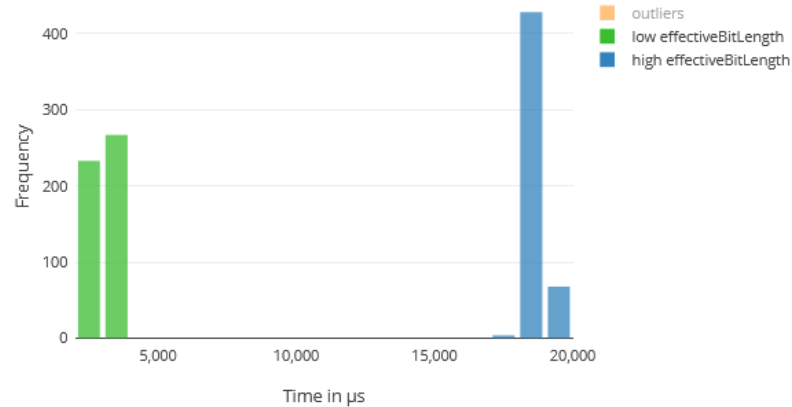
```

Avg µs
1 void test_BN_MOD(APDU apdu, short datalen) {
2   byte[] apdubuf = apdu.getBuffer();
3   short p1 = ((short) (apdubuf[ISO7816.OFFSET_P1] & 0xff));
4   Bignat num = m_testBN1;
5   num.set_size(p1);
6   Bignat mod = m_testBN2;
7   mod.set_size(((short) (datalen - p1)));
8   num.from_byte_array(p1, ((short) (0)), apdubuf, ISO7816.OFFSET_CDATA);
9   mod.from_byte_array(((short) (datalen - p1)), ((short) (0)), apdubuf,
10  num.mod(mod);
11  short len = num.copy_to_buffer(apdubuf, ((short) (0)));
12  apdu.setOutgoingAndSend(((short) (0)), len);
13 }
    
```

Colour explanation

- Currently selected trap
- Trap was never reached
- Trap was reached only sometimes

Click on a graph item to get a list of corresponding inputs.



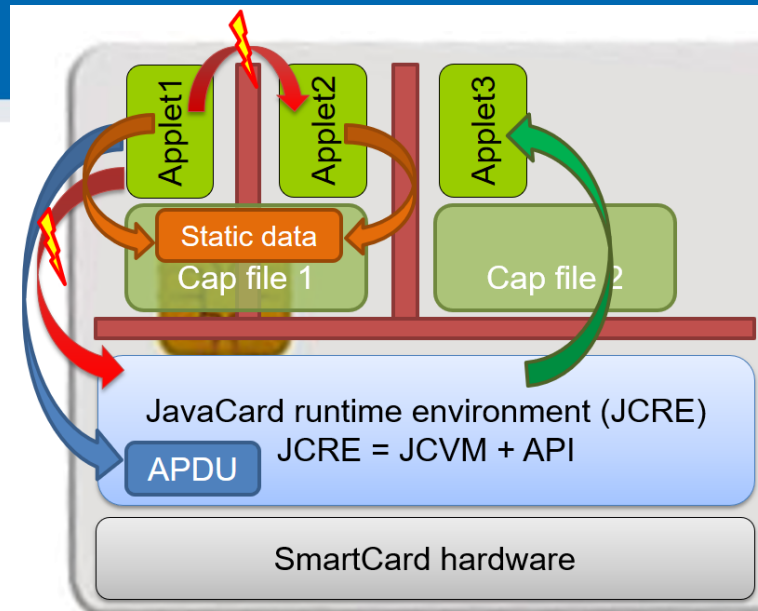
JCProfilerNext – profiling via power measurement

- The default measurement option is host-based timer => imprecise
 - Exception thrown after every line of code, measured with whole roundtrip
- Idea: insert distinct operation visible in powertrace after every line
 - Original code is instrumented with `3xRNG.generateData()` instead of exception
 - Powertrace of whole method is captured
 - RNG operations are detected and used as separators
 - Precise timing of operation is obtained
 - Visualization is performed using standard JCProfilerNext pipeline
- More elaborate setup (oscilloscope), but very precise measurement
 - better detection of non-constant-time operations

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/JCAIlgTest>
 - <https://github.com/OpenCryptoProject/JCProfiler>
- Curated list of JavaCard applets
 - <https://github.com/crocs-muni/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



P PetrS

0

Is my password brute-force-able if consists of 9 printable characters?

- **Place/upvote questions in slido while listening to lecture video**
- **We will together discuss these during every week lecture Q&A (every Monday, 17-18:00)**

Join at
slido.com
#pv204_2021

