BUSLab

Brno University Security Laboratory

Cryptographic smart cards and their practical security

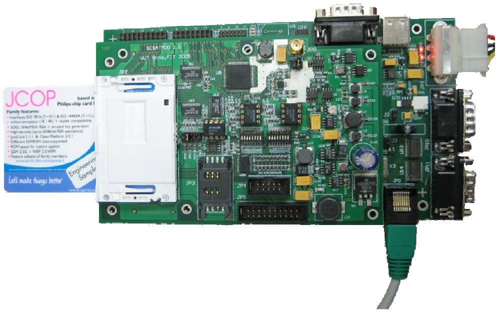
Petr Švenda

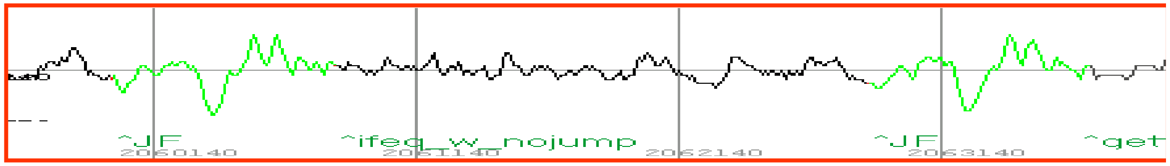
Masaryk University, Czech Republic, Brno

svenda@fi.muni.cz

What’s in pipeline?

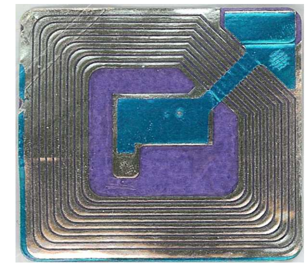
- Cryptographic smart cards
  - Basic details and specifications
- Applications
  - Common applications
  - Custom build systems
- Programming
  - PC and card side
- Attacks
  - Dismantling, side channel attacks…





Cryptographic smart card basics

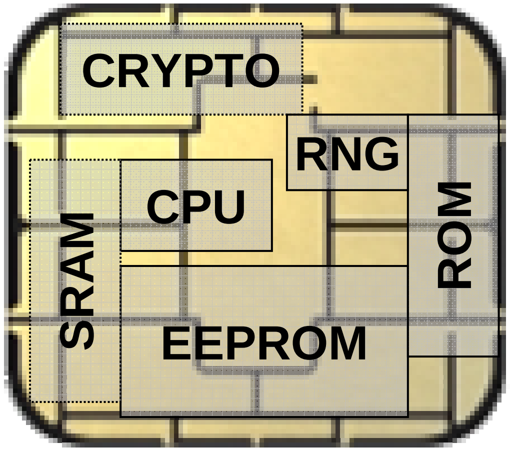
Basic types of (smart) cards



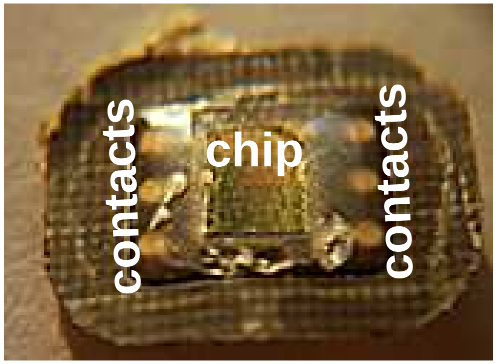
- Contactless "barcode"
  - Fixed identification string (RFID, < 5 cents)
- Simple memory cards (magnetic stripe, RFID)
  - Small write memory (< 1KB) for data, (~10 cents)
- Memory cards with PIN protection
  - Memory (< 5KB), simple protection logic (<$1)
- Cryptographic smart cards
  - Support for (real) cryptographic algorithms
  - Mifare Classic ($1), Mifare DESFire ($3)

- User programmable smart cards
  - Java cards, .NET cards, MULTOS cards ($10-$30)

Cryptographic smart cards



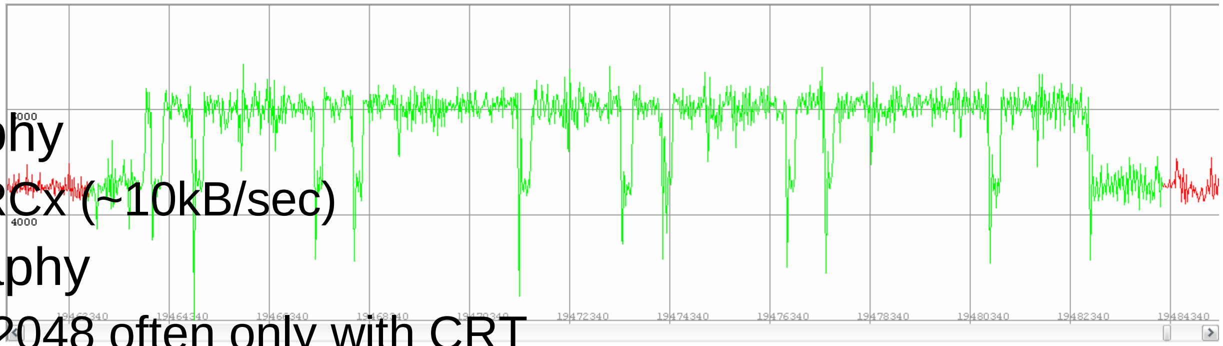
- SC is quite powerful device
  - 8-32 bit procesors @ 5-20MHz
  - persistent memory 32-100kB (EEPROM)
  - volatile fast RAM, usually <<10kB
  - truly random generator
  - cryptographic coprocessor (3DES, RSA-2048,...)
- Programmable (C, *JavaCard*, .NET)
  - (Java) Virtual Machine
  - multiple CPU ticks per bytecode instruction
  - interfaces
    - I/O data line, voltage and GND line (no internal power source)
    - clock line, reset lines
- 5.045 billion units shipped in 2008 (EUROSMART)
  - 4 185 million smartcards, 800 million memory cards
  - 3 580Mu in Telcom, 680Mu payment and loyalty...

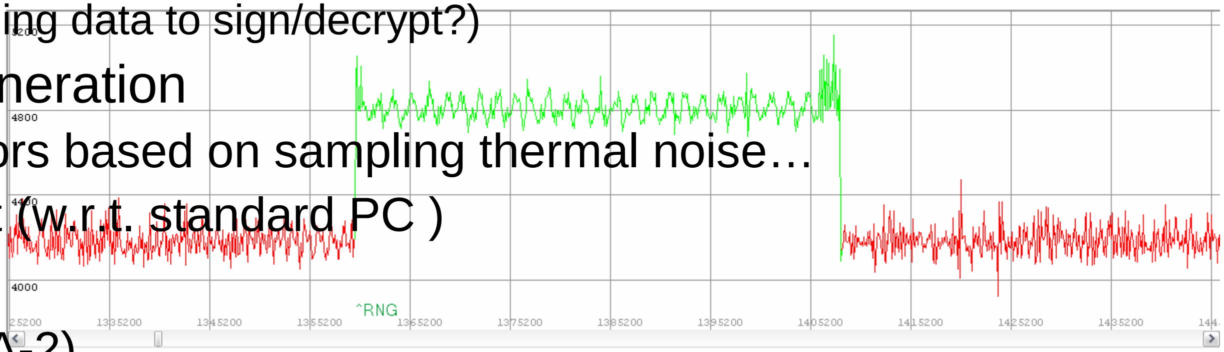


Smart cards forms

- Possible forms
  - ISO 7816 standard
  - SIM size, USB dongles, Java rings…
- Contact(-less), hybrid/dual interface
  - contact physical interface
  - contact-less interface
    - chip powered by current induced on antenna by reader
    - reader->chip communication - relatively easy
    - chip->reader – dedicated circuits are charged, more power consumed, fluctuation detected by reader
  - hybrid card – separate logics on single card
  - dual interface – same chip accessible contact & c-less

Supported algorithms

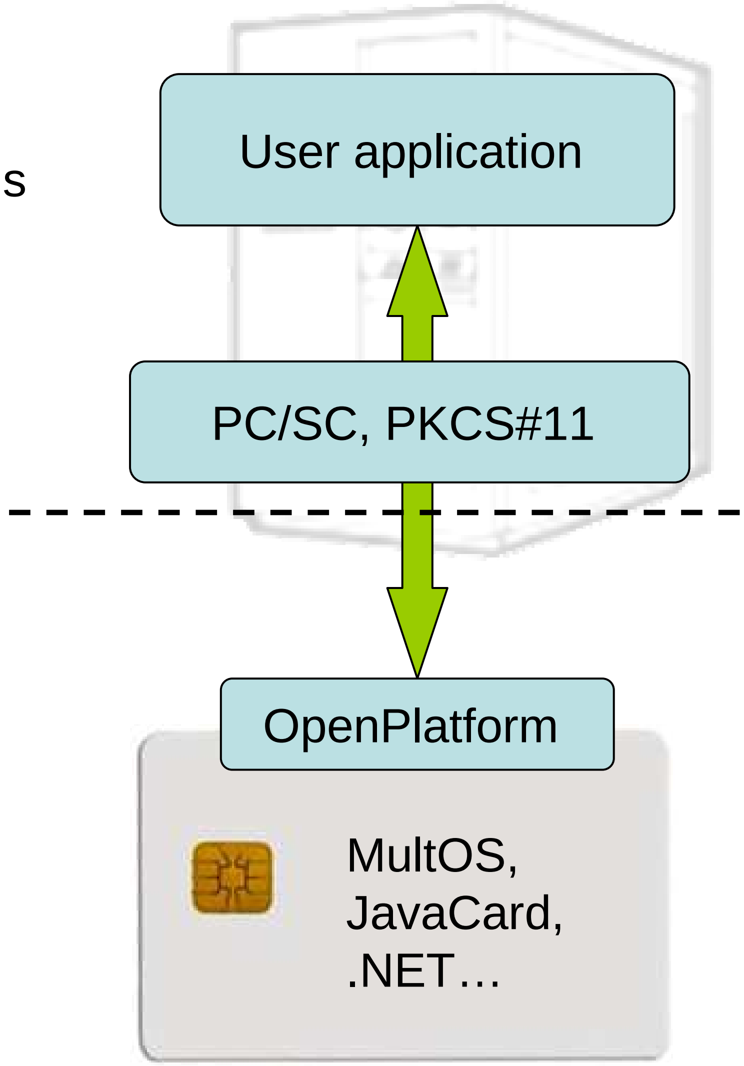




- Symmetric cryptography
  - DES, 3DES, AES, RCx (~10kB/sec)
- Asymmetric cryptography
  - RSA 512-2048bits, 2048 often only with CRT
  - Diffie-Hellman key exchange, Elliptic curves
    - rarely, e.g., NXP JCOP 4.1
  - on-card asymmetric key generation
    - private key never leaves card!
    - (but who is sending data to sign/decrypt?)
- Random number generation
  - hardware generators based on sampling thermal noise…
  - very good and fast (w.r.t. standard PC)
- Message digest
  - MD5, SHA-1, (SHA-2)
- See *http://www.fi.muni.cz/~xsvenda/jcsupport.html* for more

Common environments and interfaces

- MultOS
  - C programming, native code compilation
  - high security certifications, often bank cards
- Java Card
  - open programming platform from Sun
  - applets portable between cards
- Microsoft .NET for smartcards
  - relatively new technology
  - similar to Java Card
  - applications portable between cards
- PC/SC, PKCS#11
  - standardized interface on host side
  - card can be proprietary
- OpenPlatform (GlobalPlatform)
  - remote card management interface
  - secure installation of applications



Applications

Applications

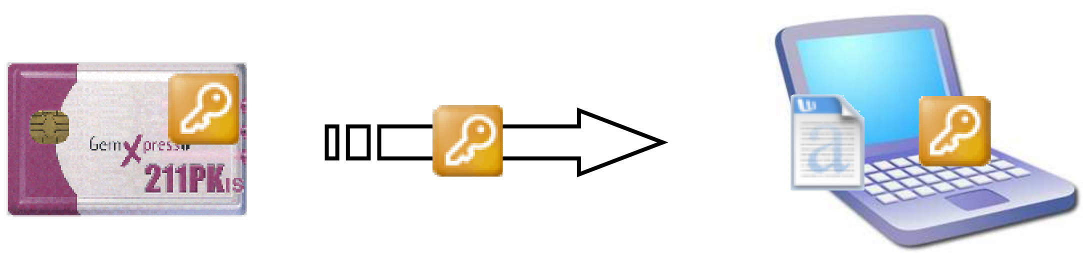
- Bank payment card (EMV standard)
  - cryptographic checksum on payment bill
  - offline PIN verification
- GSM SIM modules
  - GSM banking
  - phone startup PIN protection
- Secure system authentication
  - Windows GINA, Linux PAM modules
  - password storage only, challenge-response protocols
  - door access cards – mostly memory cards only

Application (cont.)

- ePassports
  - contactless cards with Machine Readable Zone (MRZ)
  - secure messaging between reader and passport
    - key derived from MRZ (~35bits entropy!)
  - active authentication
    - challenge-response with on-passport asymmetric key
- Multimedia distribution
  - Digital Rights Management (decryption keys, licenses)
  - pre-paid satellite TV (decryption keys)
- Secure storage and encryption device
  - PGP/GPG, TrueCrypt…

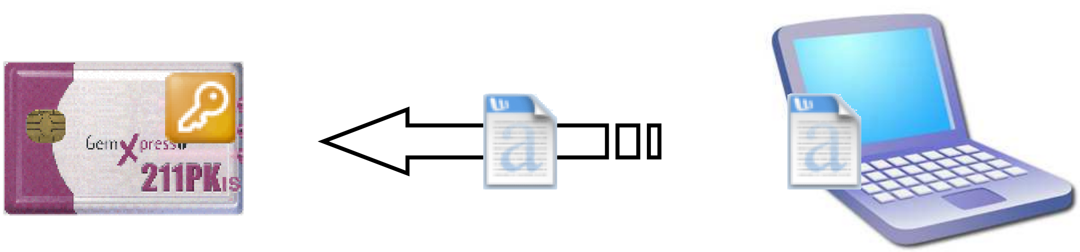
Smart card as a secure carrier

- Key stored on the card, loaded to the PC before encryption, then erased
- High speed encryption (>>MB/sec)
- Attacker with access to the PC during encryption will obtain the key
  - key protected for transport, but **not during usage**



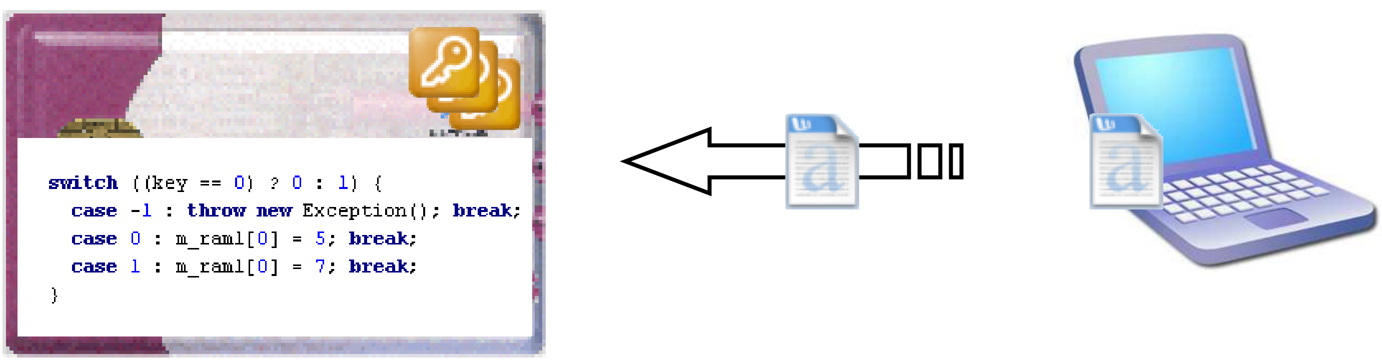
Smart card as an encryption device

- PC just sends data for encryption
- Key never leaves the card
  - protected during transport and usage
- Attacker must attack the smart card
  - or wait until card is inserted and PIN entered!
- Low speed encryption (~kB/sec)
  - mainly due to the communication speed

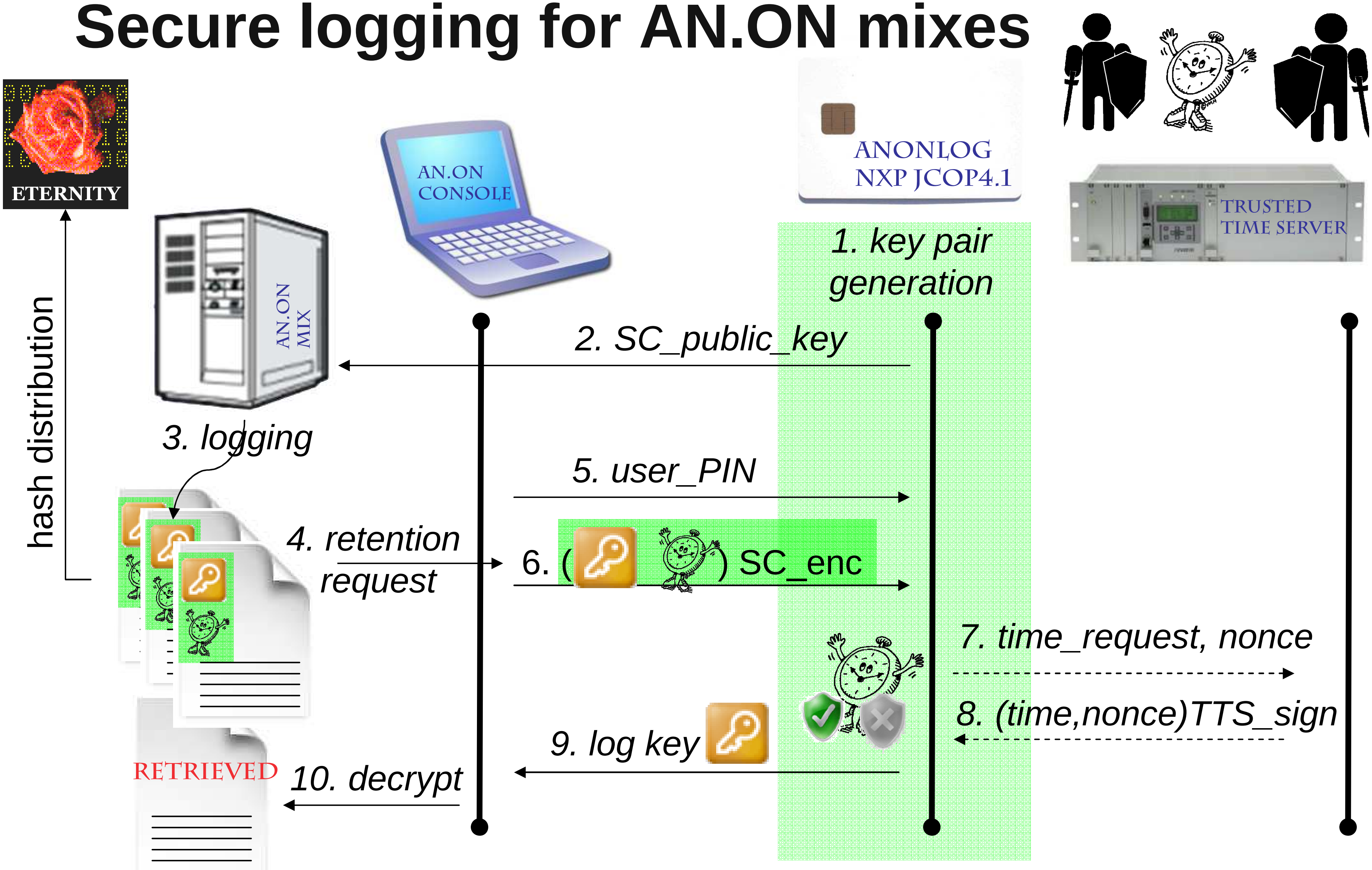


Smartcard as computational device

- 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
    - secure server, trusted time service…
    - PC act as a transparent relay only (no access to data)
- Attacker must attack the smart card



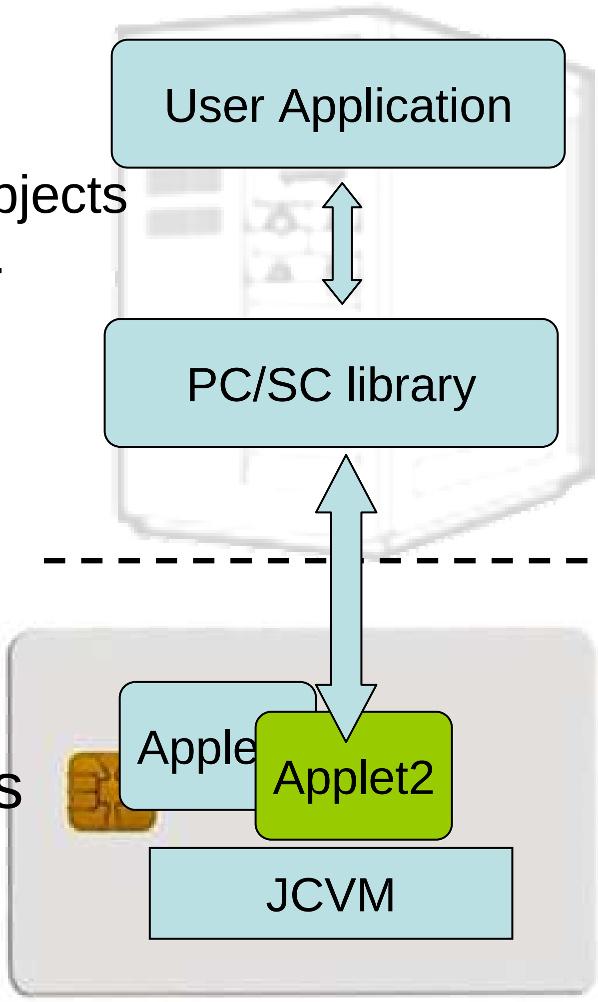
Secure logging for AN.ON mixes



Smart card application development

Java Card 2.x 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



Demo Java Card applet

1. Develop Java Card Applet (NetBeans with Java Card plugin)
   a. subclass *javacard.framework.Applet*
   b. parsing of APDU in *Applet::process() method*
   c. usage of JC API algorithms/objects (e.g., RSAPrivateKey)
2. Upload on smart card (OpenPlatform, GPShell)
   a. applet installed with unique ID (AID)
   b. remote installation possible (secure channel)
3. Develop PC application (PC/SC, SCardxxx fnc)
   a. list available readers (SCardListReaders), connect (SCardConnect)
   b. select command by application AID (00 a4 04 00 xx AID)
   c. send input data, receive processed data from applet (SCardTransmit)

Java Card 3.x

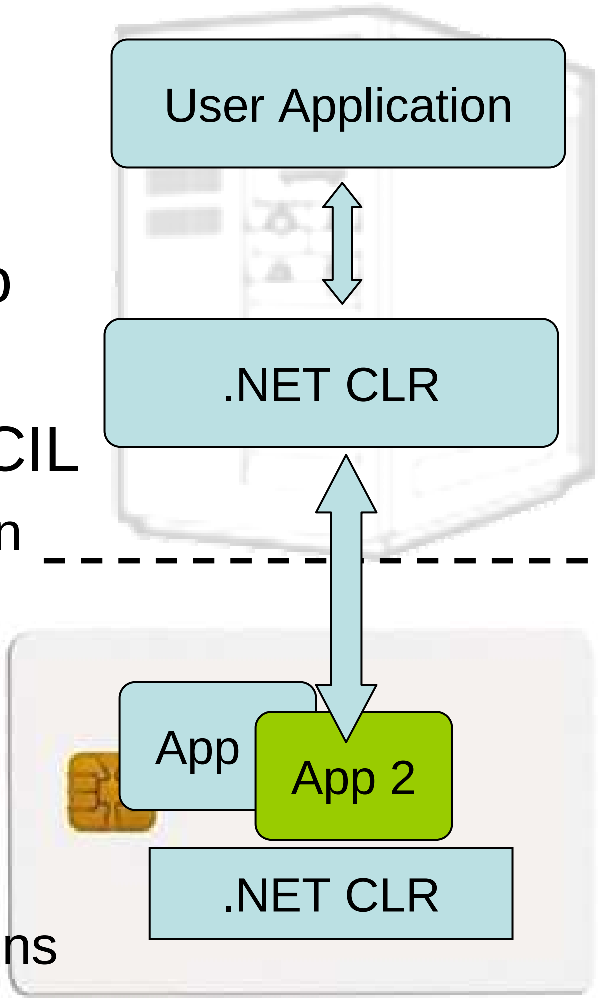
- Recent major release of Java Card specification
  - principal changes in development logic
  - two separate branches – Classic and Connected edition
- Java Card Classic Edition
  - legacy version, extended JC 2.x
  - APDU-oriented communication
- Java Card Connected Edition
  - smart card perceived 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

Java Card security

- Application firewall
  - strict separation of applications
  - communication only over special limited interface (Shareable)
  - formal proof of correctness
  - secure co-existence of independent applications
    - similar to trusted computing, whole "PC" in a protected box
- Offline, on-card, run-time code verification
  - types control, code correctness, …
  - offline – on compilation platform (common, TrustedLogic)
  - on-card – when loading to SC (less common, limited resources)
  - run-time – during execution (uncommon, limited resources)
- Code encryption and signing
  - compiled code is encrypted and signed
  - offline transmitted over insecure channel to SC
    - installed only when correct keys were used
  - online version (more common)
    - mutual authentication and online secure channel

.NET smart card v 2.x applications

- Writing in .NET compatible language
  - C#, VB.NET...
- SDK toolkit integrated into Visual Studio
  - Gemalto SDK - paid
- Compiled into .NET intermediate code CIL
  - both PC (client) and card (server) application
- Uploaded and installed into smartcard
  - executed in .NET Virtual Machine
  - identified by URI (e.g. “CryptoService.uri”)
- Remote method invocation
  - PC app (client) calls SC app (server) functions
  - (transparently realized via APDU commands)

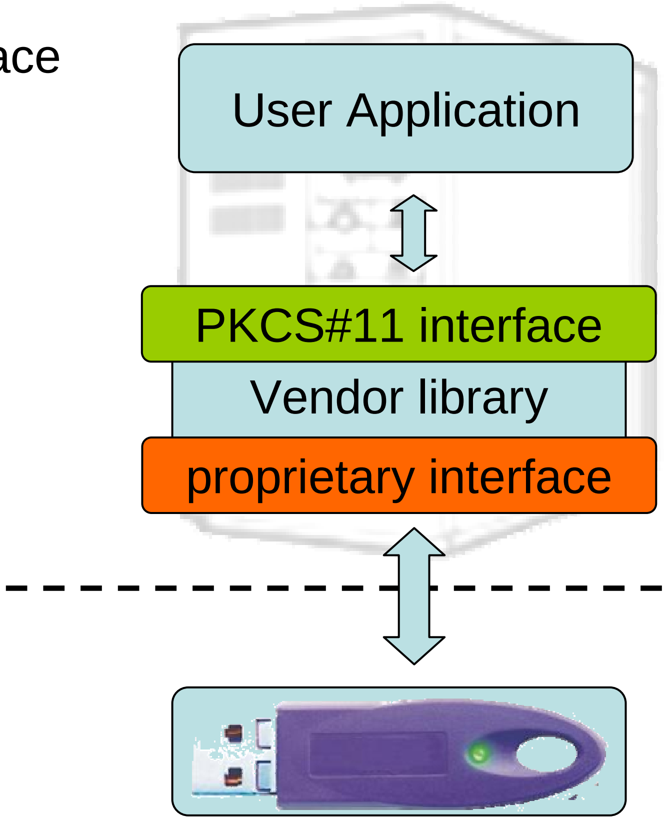


Demo .NET application

1. Develop .NET server (Visual Studio,Gemalto .NET SDK)
   a. implement card functions (will be called by RMI)
   b. identify your application with URI (e.g., ‘*CryptoService.uri*’)
   c. usage of algorithms from *System.Security.Cryptography*
2. Upload on smart card
   a. application installed as server with given URI
3. Develop PC application
   a. stub DLL is automatically generated for server interface
   b. connect to server based on its URI (e.g.,’*apdu://selfdiscover/CryptoService.uri’)*
   c. call methods directly via RMI

PKCS#11

- Well-established standard for smart card access, widely used
- Standardized interface of security-related functions
  - vendor-specific library in OS, often paid
  - communication library->card proprietary interface
- Functionality covers
  - slot and token management
  - management of objects in smartcard memory
  - encryption/decryption functions
  - message digest
  - creation/verification of digital signature
  - random number generation
  - PIN management
  - lots of functions actually in software only ☹
- Secure channel not possible!
  - developer can control only App->PKCS#11 lib



Demo PKCS#11 application

1. Obtain PKCS#11 - enabled card
   a. card side is not programmed by developer
   b. (emulation by Java Cards – OpenSC/Muscle project)
   c. card manufacturer provide dynamic library with PKCS#11 implementation (e.g., opensc-*pkcs11.dll*)
2. Develop PC application
   a. load PKCS#11 library
   b. obtain list of available slots with cards (C_GetSlotList)
   c. connect to card (C_OpenSession)
   d. login by PIN (C_Login), search for object (C_Findxxx)
   e. use it (C_Sign, C_Decrypt)…

PKCS#11 security

- Specification is too broad and sometimes vague
- Lack of policy for function calls
  - functions are too “low-level”
  - sensitive objects can be manipulated directly
- Missing authentication of wrapped key
  - attacker can create its own wrapping key
  - and ask for export of unknown key under his own wrapping key
- Export of longer keys under shorter, …

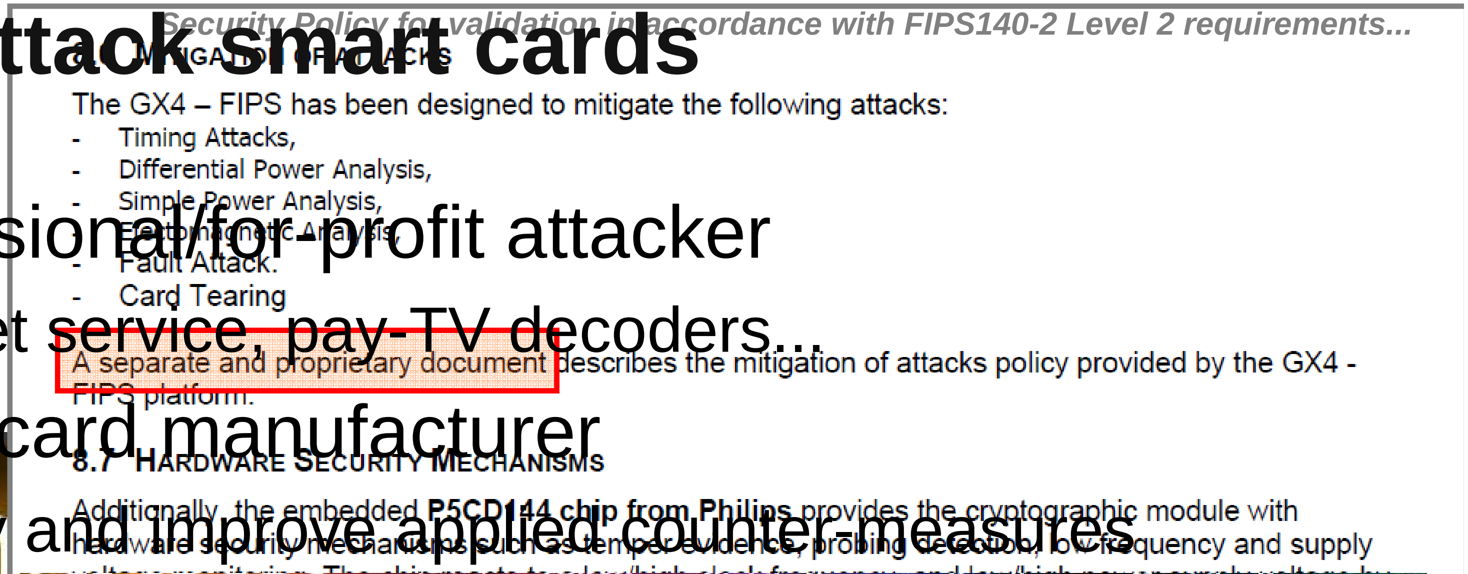
OpenPGP

- Only interface officially supported by GnuPG
  - cards are available
  - can be simulated with Java Card (JOpenPGPGCard)
- Demo GnuPG --card-edit
  - signature, decryption and authentication key
  - private keys generated directly on the card

Smart cards security

Why to attack smart cards

1. Professional/for-profit attacker
   - secret service, pay-TV decoders...
2. Smart card manufacturer
   - verify and improve applied counter-measures
3. Customer
   - test of card before purchase
   - details about actual defensive mechanisms secret
   - same type of card have different hardware over time



Security Policy for validation in accordance with FIPS140-2 Level 2 requirements...

8.6 MITIGATION OF ATTACKS

The GX4 – FIPS has been designed to mitigate the following attacks:

- Timing Attacks,
- Differential Power Analysis,
- Simple Power Analysis,
- Electromagnetic Analysis,
- Fault Attack.
- Card Tearing

A separate and proprietary document describes the mitigation of attacks policy provided by the GX4 - FIPS platform.

8.7 HARDWARE SECURITY MECHANISMS

Additionally, the embedded P5CD144 chip from Philips provides the cryptographic module with hardware security mechanisms such as tamper evidence, probing detection, low frequency and supply

Title:	Securing microprocessors against information leakage and physical tampering
	US20080126766
	May 29, 2008
Abstract:	A processor system comprising: performing a compilation process on a computer program; encoding an instruction with a selected encodir program in the processor using an added mutation instruction, wherein executing comprises executing a mutation instruction to enable dec defense against offline and runtime security attacks including software and hardware reverse engineering, invasive microprobing, fault injec

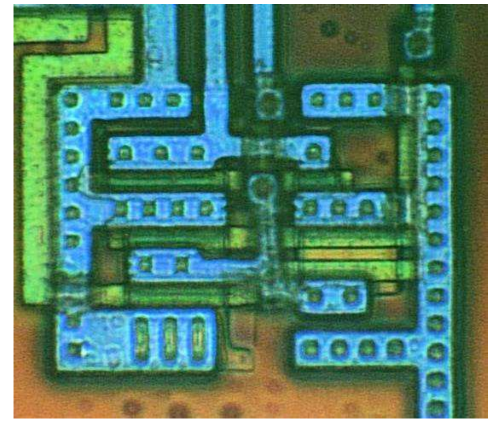
Basic concepts in smart card security

- Physical security
  - physical barrier of chip (e.g., guard, epoxide resin…)
- Tamper resistance
  - property of the (sub-)system
  - difficulty for unauthorized modification higher then rest of the system
- Tamper evidence
  - non-authorized modification will leave detectable traces
  - e.g., nutshell will broke into many pieces
- Tamper detection
  - automatic detection of attempts to tamper (physical) security
  - condition sensor, wire cage…
- Tamper response
  - automatic action after detected attack (e.g., key deletion, card block)

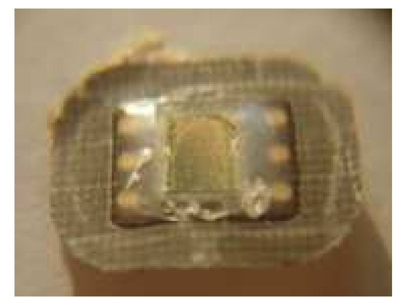
Attacker classification

- Difference based on knowledge, ability, financial support, access to special equipment …
- Classification according to IBM:
  - Class 1 – Clever outsiders
    - intelligent, but missing basic knowledge about system, access only to moderately sophisticated equipment
    - *sometimes class 1.5 – good laboratory equipment (academy)*
  - Class 2 – Insiders (specialized laboratories)
    - specialized technical knowledge and experience
  - Class 3 – Well-funded organizations (govs…)
    - teams of specialists, almost unlimited financial resources, detailed analysis of the system

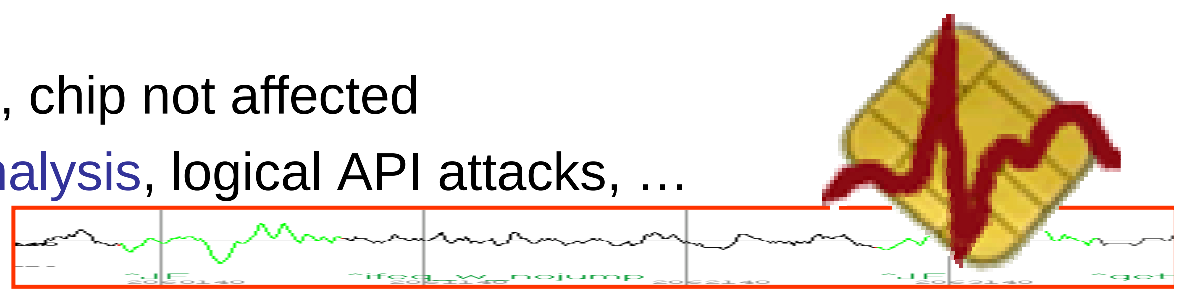
Basic types of attacks



- Invasive
  - physical de-packaging, chip often destroyed
  - reading microprobes, direct memory access
  - usually high cost attack
- Semi-invasive
  - often de-packaging, but chip still usable/working
  - optical fault induction
  - supply voltage and clock peaks, …
  - often low cost

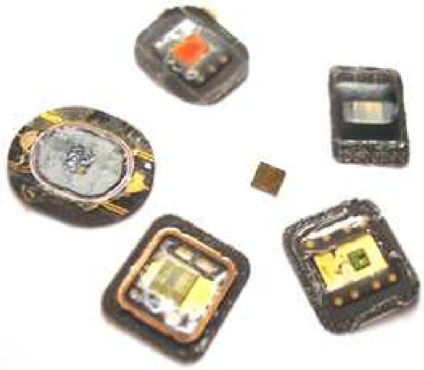


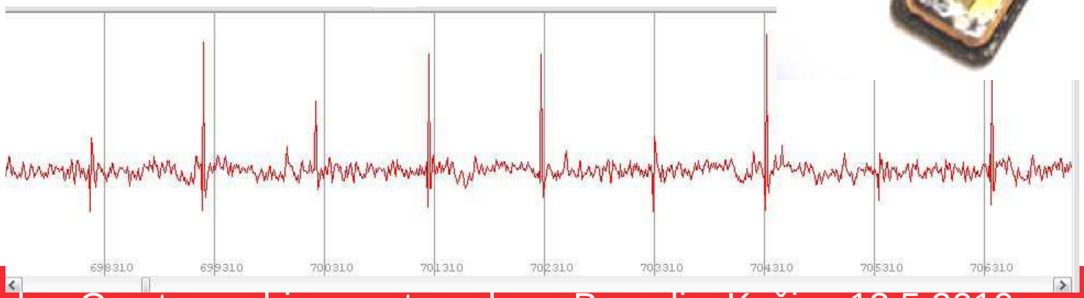
- Non-invasive
  - passive observation, chip not affected
  - timing and power analysis, logical API attacks, …



Fault induction attack

- Sudden changes in operating conditions with aim to induce change in memory, register...
  - harder to induce targeted then random fault
- Target is to:
  - bypass particular instruction (PIN check)
  - change data used by logical flow (current state, cycle counter)
  - obtain result from corrupted operation (RSA CRT)
- Modifiable environmental conditions
  - power/clock/reset signal
  - EM array
  - flash burst, radiation
  - temperature…

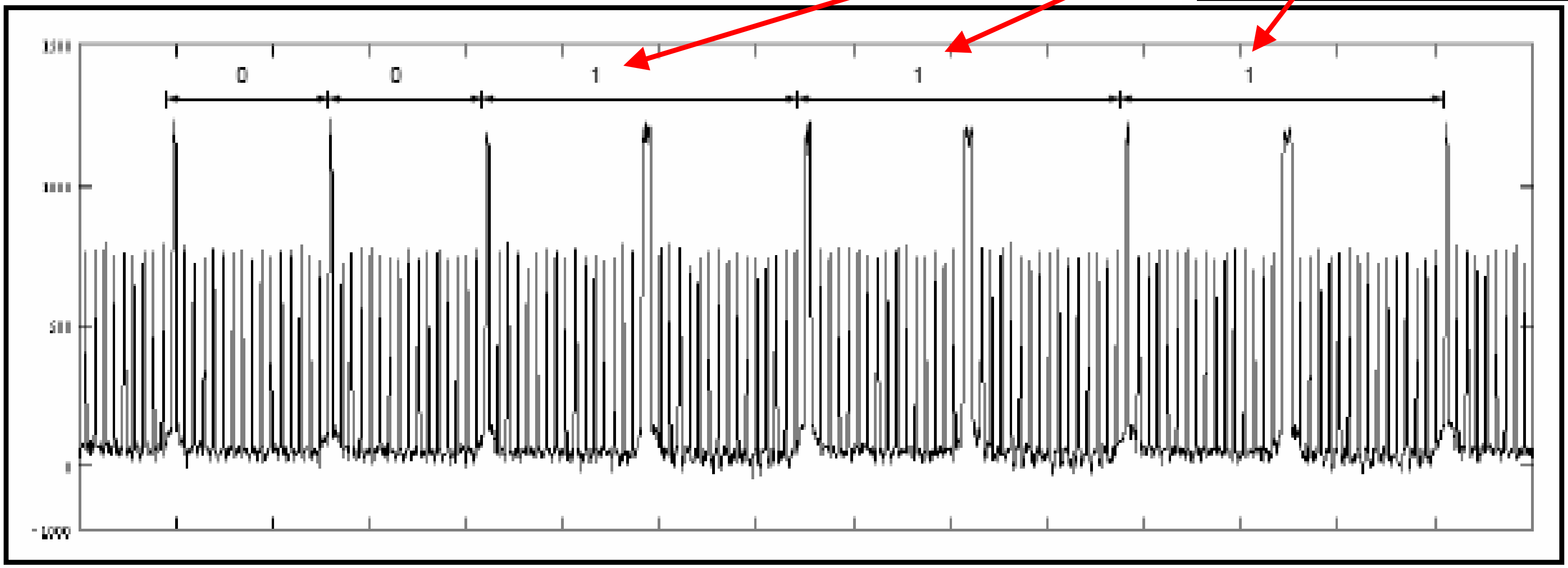




Timing analysis

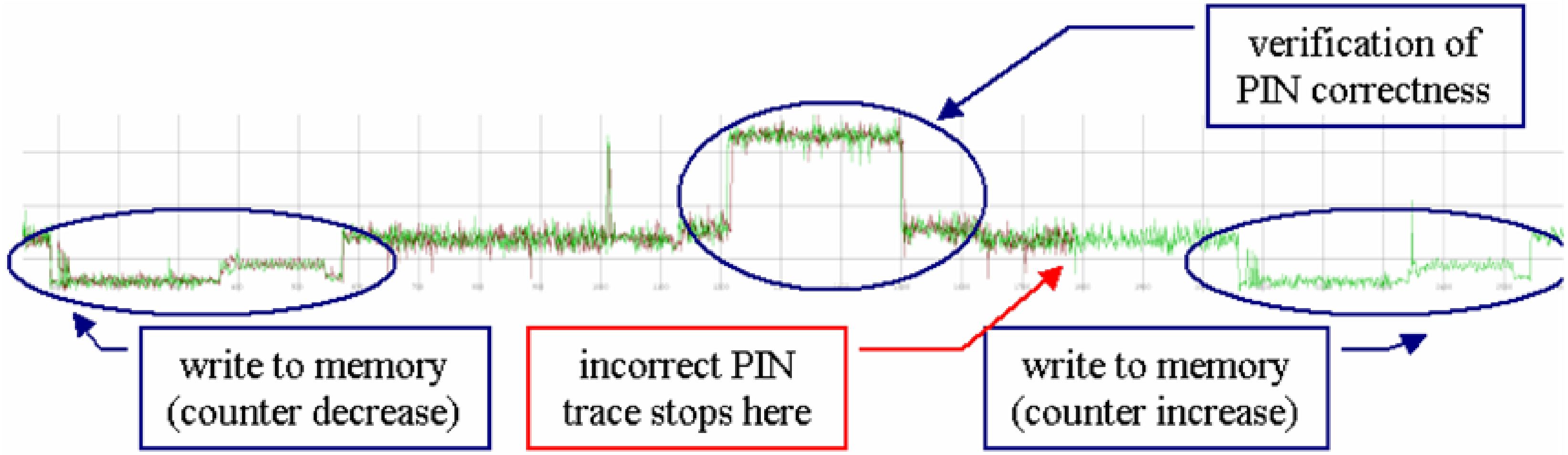
- Length of operation depends on processed data
  - due to speed optimization (limited resources)
  - due to un-aware algorithm design
  - e.g. Montgomery multiplication
- Timings obtained from power trace

$x = m$
FOR $i = n-2$ DOWNTO 0
  $x = x^2$
  IF $(k_j == 1)$ THEN
    $x = x \cdot m$
ENDFOR
RETURN x

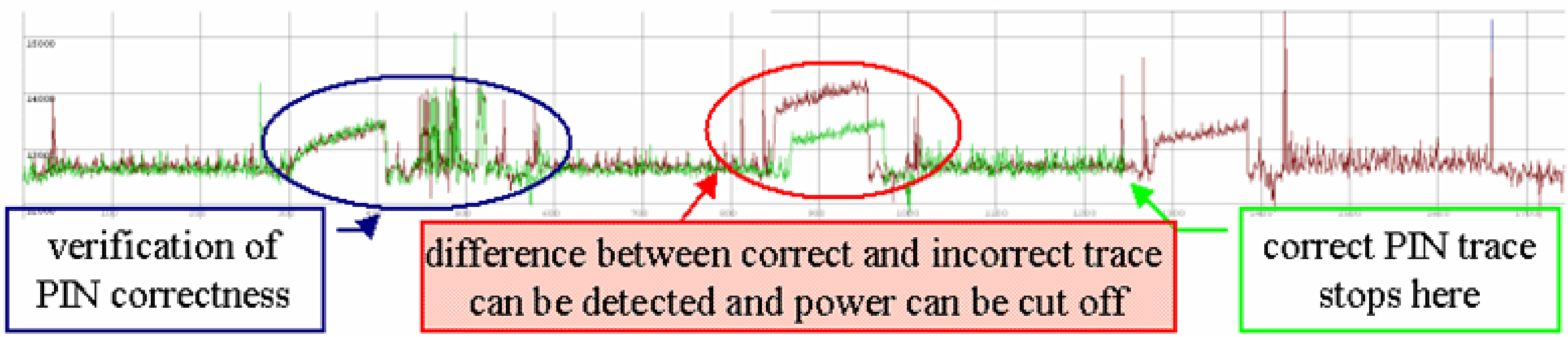


PIN verification procedure

- [Decrease counter, verify, increase] - correct



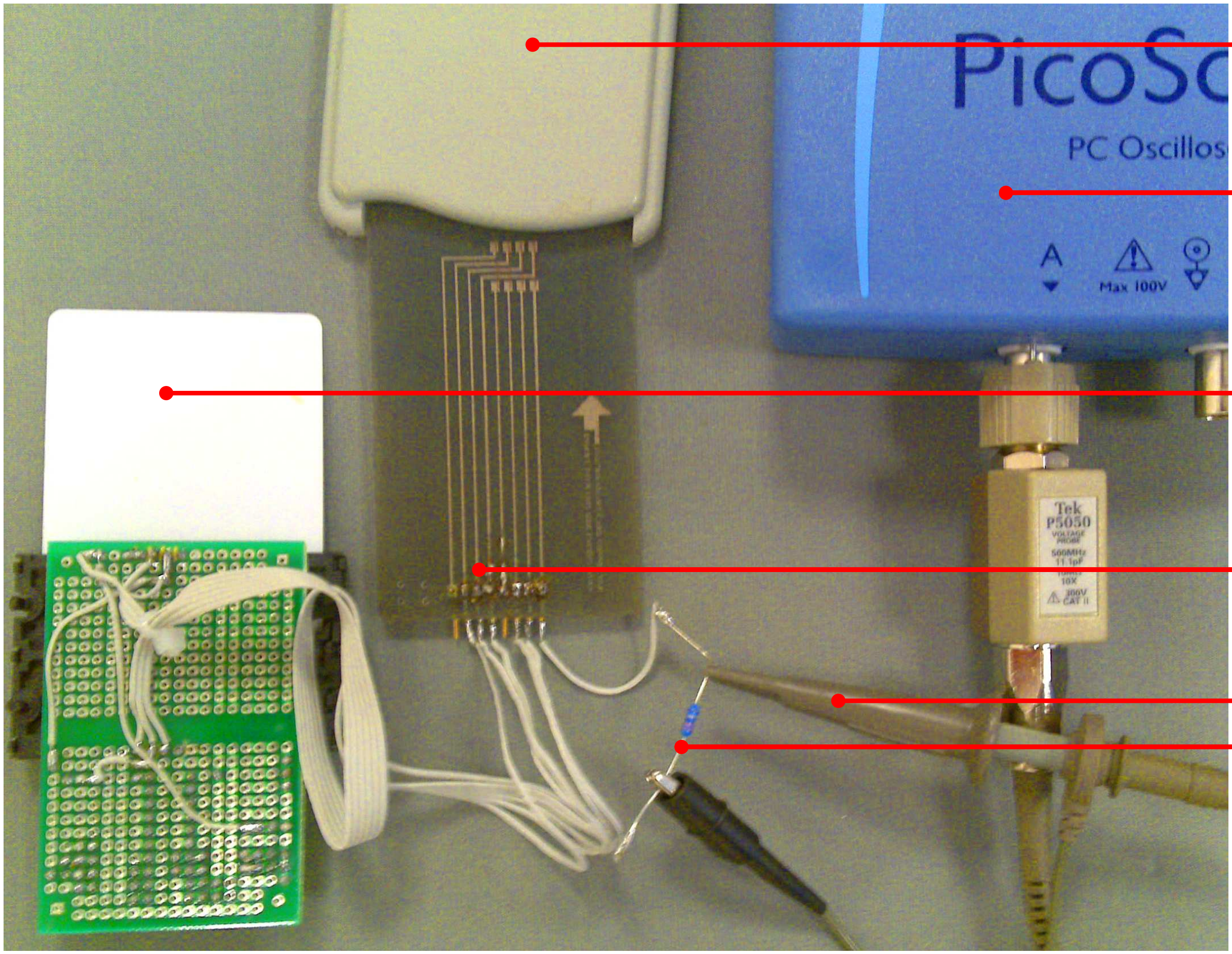
- [Verify, decrease/increase] – possibly wrong



Smart cards power analysis

- Significant vulnerability exposed
  - external power needed
  - current consumption can be measured
  - side-channel leakage, published in 1997 (Kocher et al.)
- Simple Power Analysis
  - what can be read from single/few power consumption trace
- Differential Power Analysis (e.g., OpenSCA project)
  - more advanced statistical processing, multiple traces
- Information about internal execution
  - instructions & data
  - focus on secret key operation (crypto-processor)

Basic setup for power analysis



Smart card reader

Oscilloscope

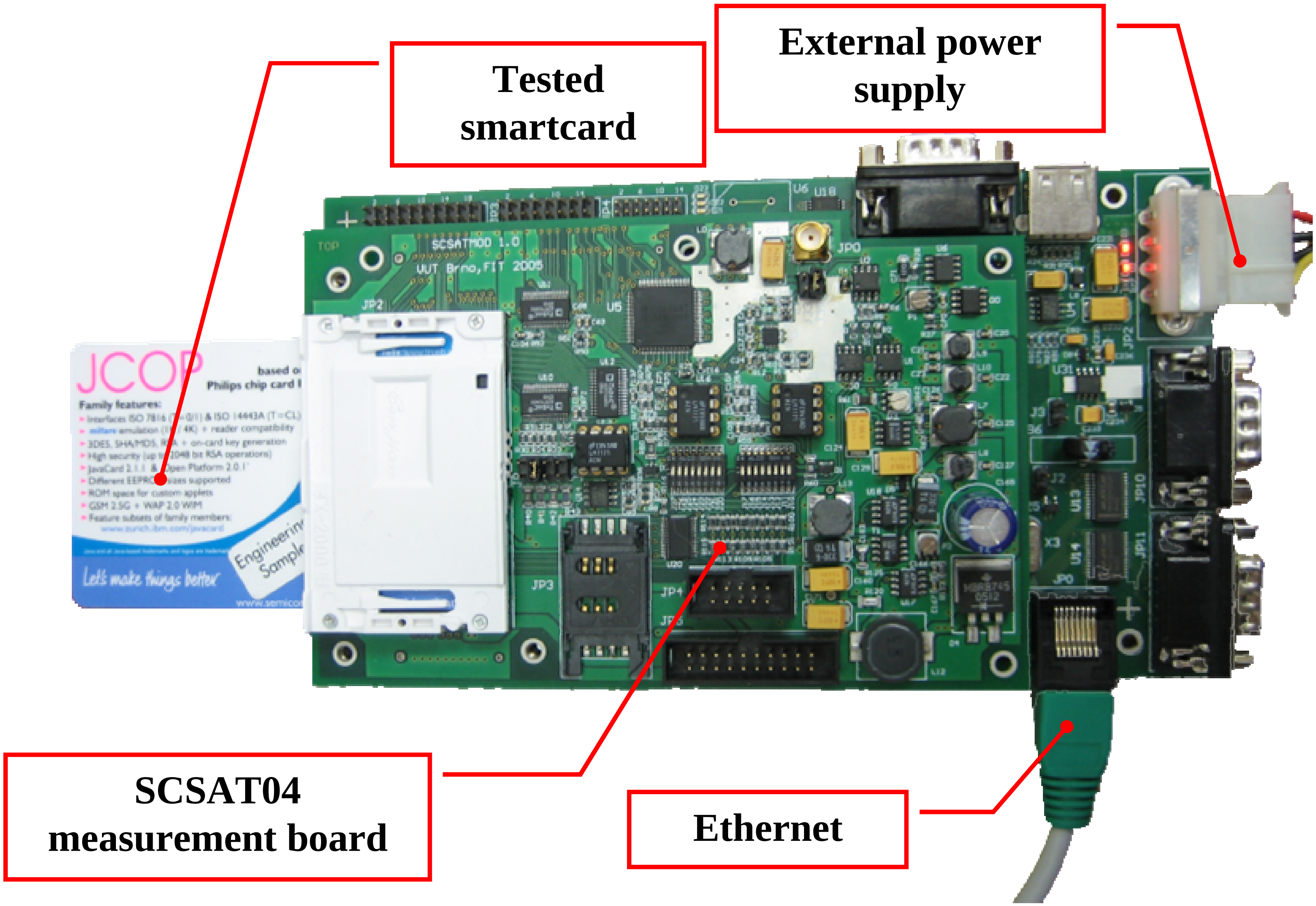
Smart card

Inverse card connector

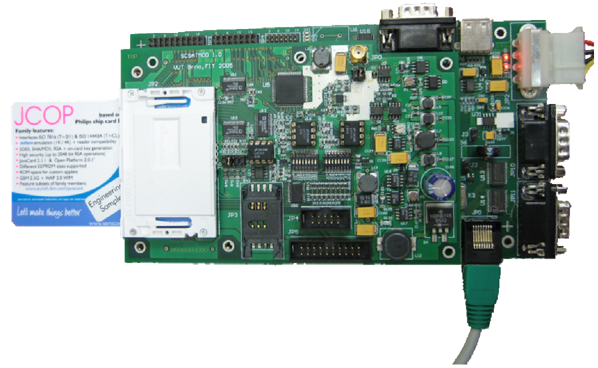
Probe

Resistor 20-80 ohm

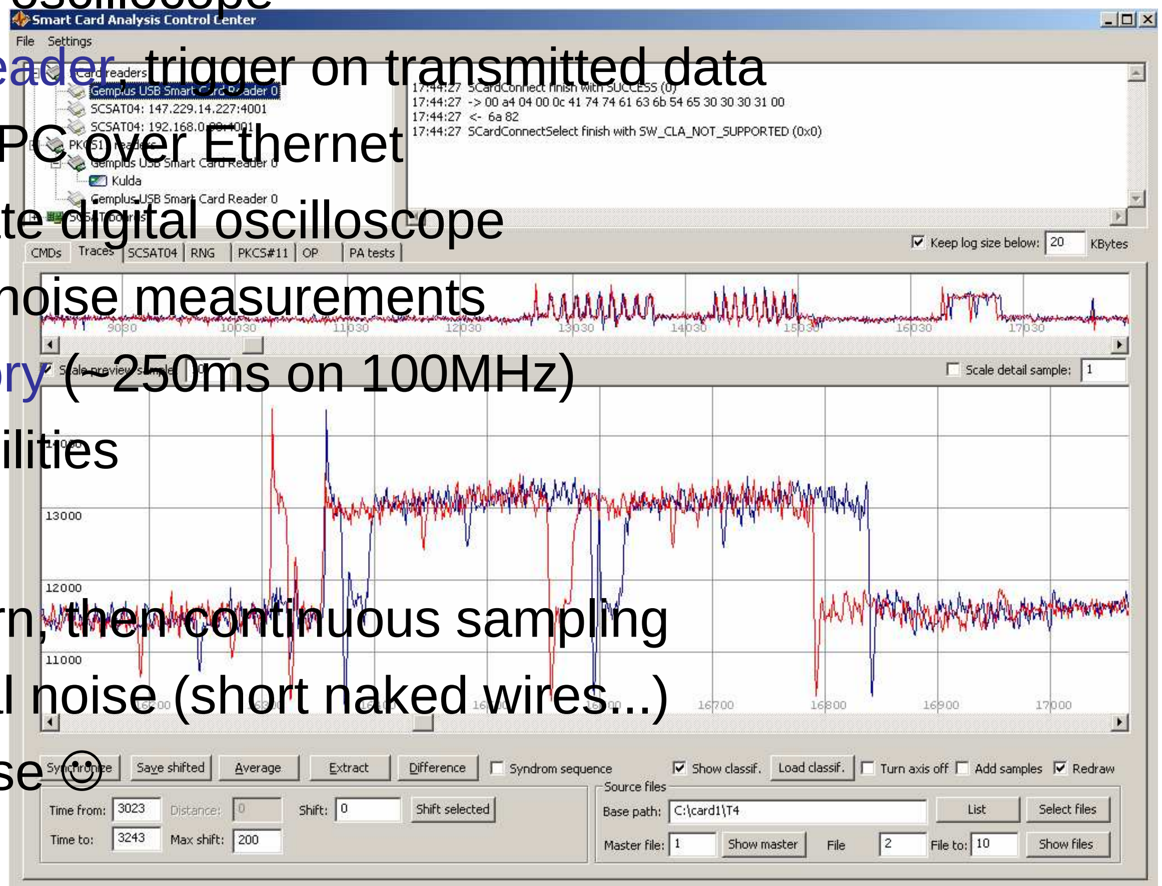
More advanced setup for power analysis



SCSAT04 (build on VUT Brno)

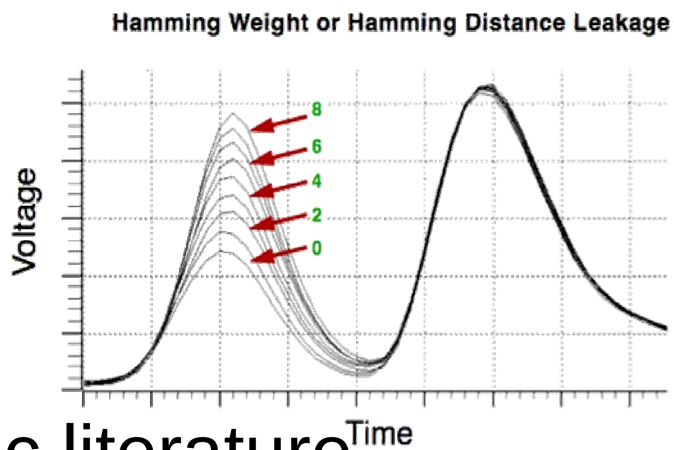


- Linux-based FPGA measurement board
  - advanced PC-based oscillocope
  - acts as smart card reader, trigger on transmitted data
  - communication with PC over Ethernet
  - 100MHz sampling rate digital oscilloscope
  - 12bits samples, low noise measurements
  - 48MB internal memory (~250ms on 100MHz)
  - fault induction capabilities
- Main advantages
  - trigger on data pattern, then continuous sampling
  - optimized for minimal noise (short naked wires...)
  - (relatively) easy to use ☺



Sensitive data leakages - Type I.

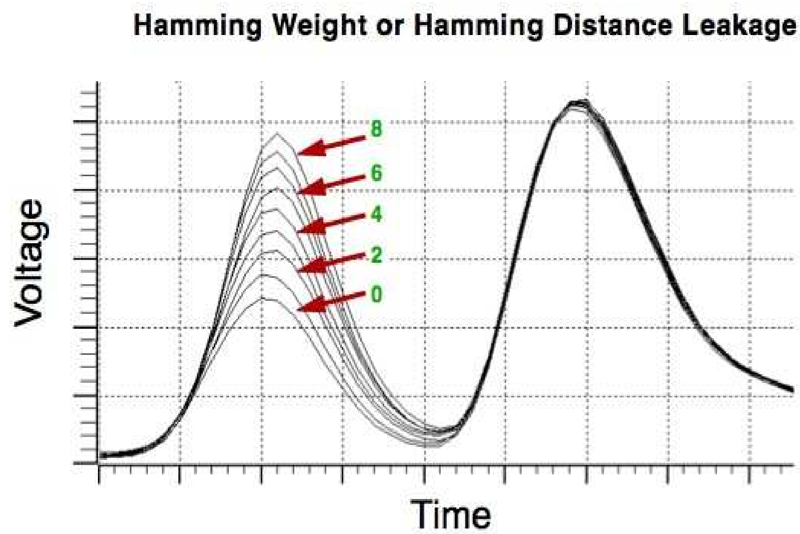
- Data revealed directly when processed
  - e.g., Hamming weight of instruction argument
    - hamming weight of separate bytes of key (2^{56}-> 2^{38})
  - directly observed (SPA)
    - single trace inspection
  - by statistical means (DPA)
    - averaging over multiple traces



- Most common target in scientific literature
  - but usually not on real smart cards!
  - problems with secret mechanisms, noise, delays...

Direct power analysis

```
bool bSimilar = TRUE;
for (short i=0; i<passLength;i++) {
    if (array1[i] != array2[i])
        bSimilar = FALSE;
}
```




```
getfield_a_this 20;
sload_3;
baload;
getfield_a_this 21;
sload_3;
baload;
if_scmpeq L4;
```

	P	A	S	S	W	O	R	D
array1	P	A	S	S	W	O	R	D
array2	A	B	C	D	E	F	G	H

Reverse engineering of Java Card bytecode

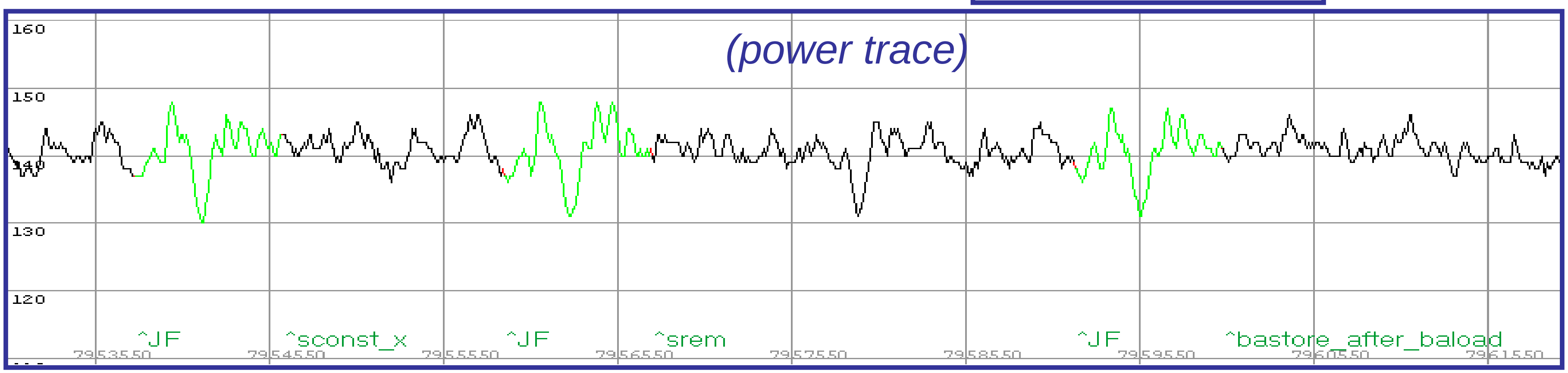
- Goal: obtain code back from smart card
  - JavaCard defines around 140 bytecode instructions
  - JVM fetch instruction and execute it

(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)

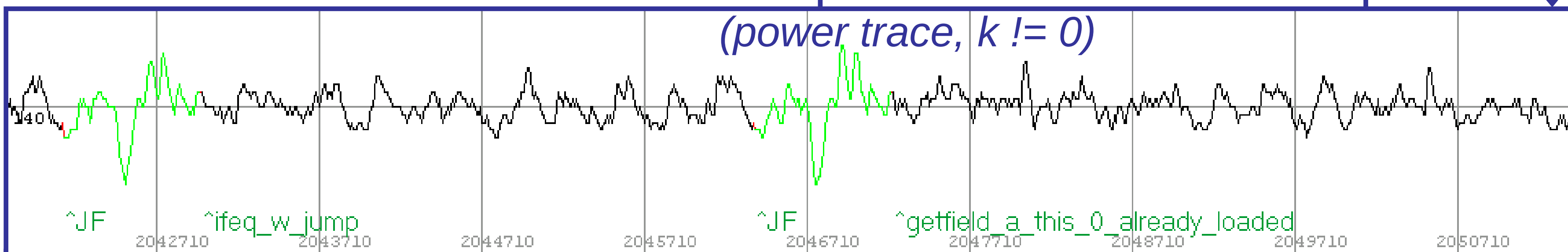
```
if (key == 0) m_ram1[0] = 1;
else m_ram1[0] = 0;
```

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: …
```

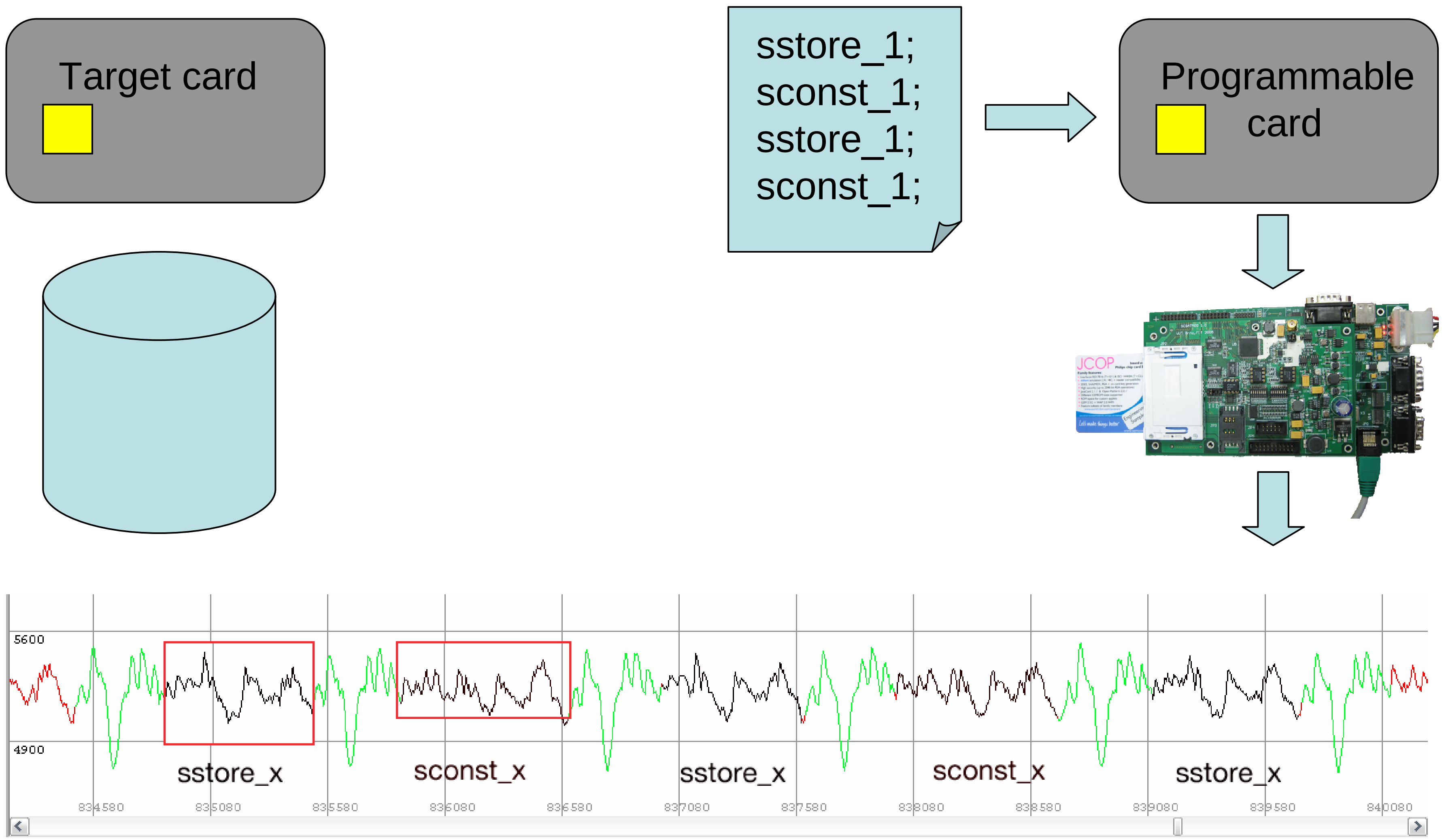
oscilloscope



(power trace, k != 0)

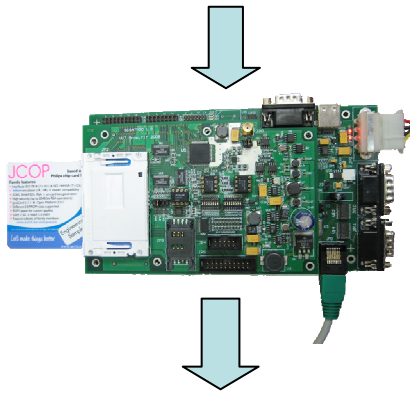
(power trace, k == 0)

Building a database



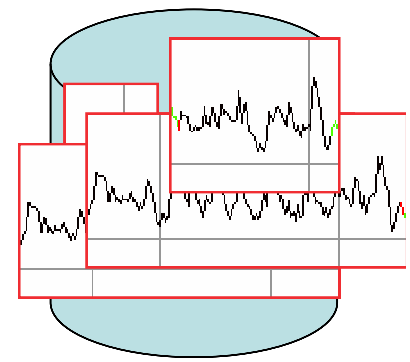
Reverse engineering

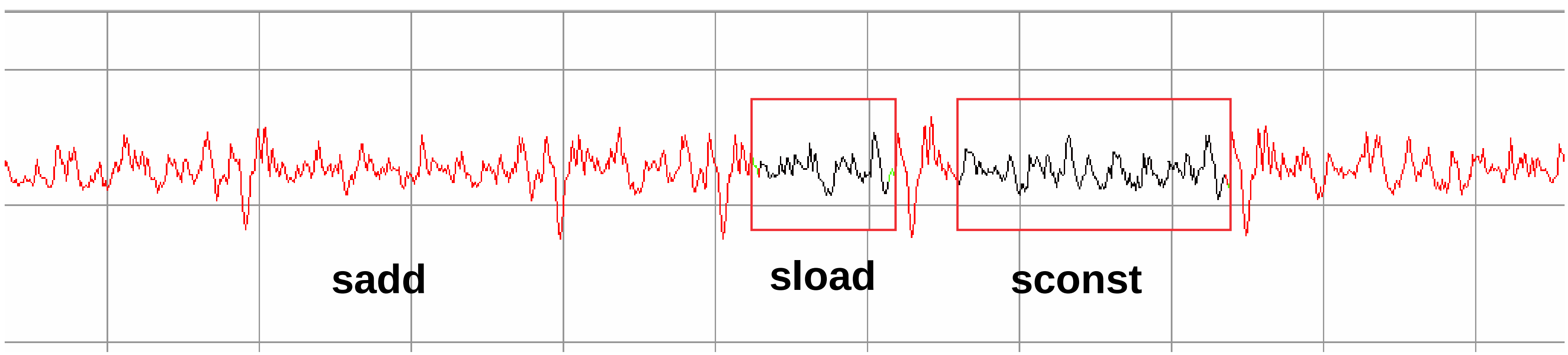
Target card



sadd;
unknown;
sload
sconst

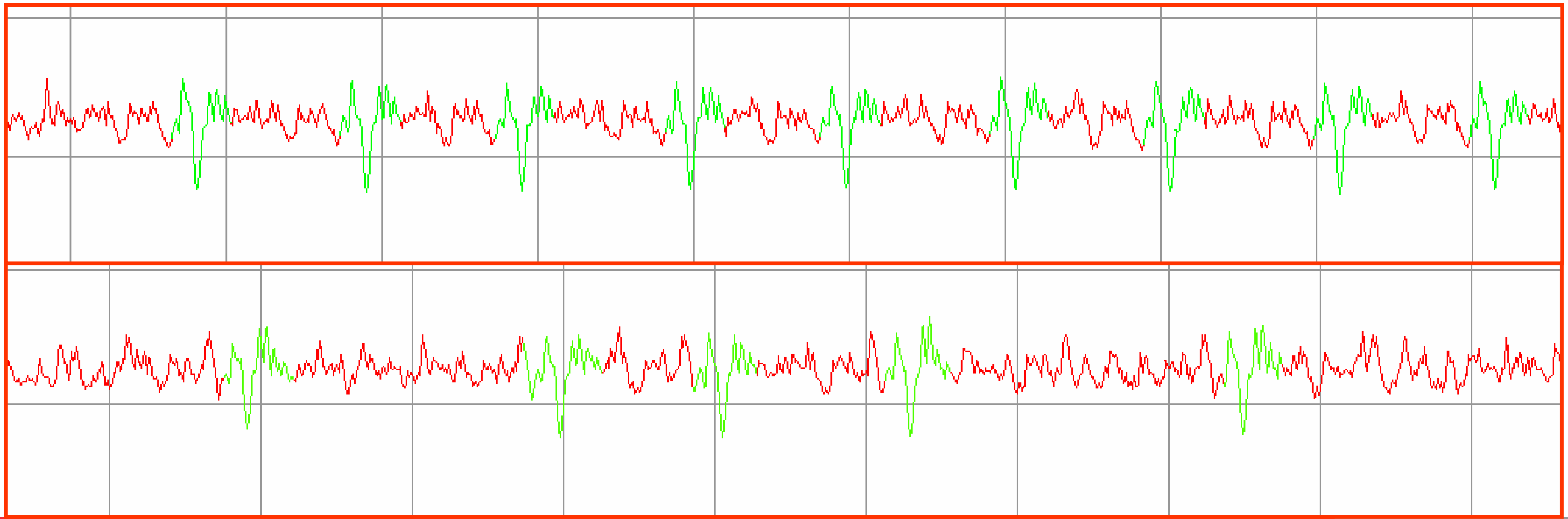
sadd;
sstore 4;
sload 4;
sconst_1;
aload_1;
sload 4;





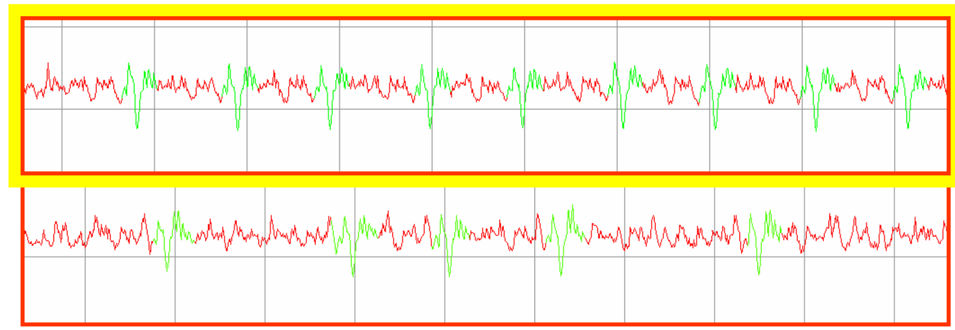
Sensitive data leakages - Type II.

- Different instructions executed
  - depending on the sensitive data
  - executed instructions can be observed
    - branch taken can be inferred
- Sensitive argument leaks



Different sequence of instructions

```
bool bSimilar = TRUE;
for (short i=0; i<passLength;i++) {
    if (array1[i] != array2[i])
        bSimilar = FALSE;
}
```



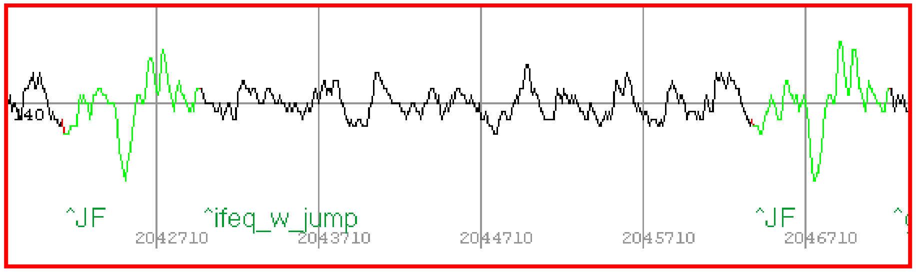
array1	P	A	S	S	W	O	R	D
	=	=					=	
array2	P	A	A	B	C	D	R	F

Sensitive data leakages - Type III.

- Single instruction execution differs
  - depending on the data manipulated
  - e.g., when jump was executed (or not)
- Probably caused by different “microinstructions” for same bytecode instruction (JVM)

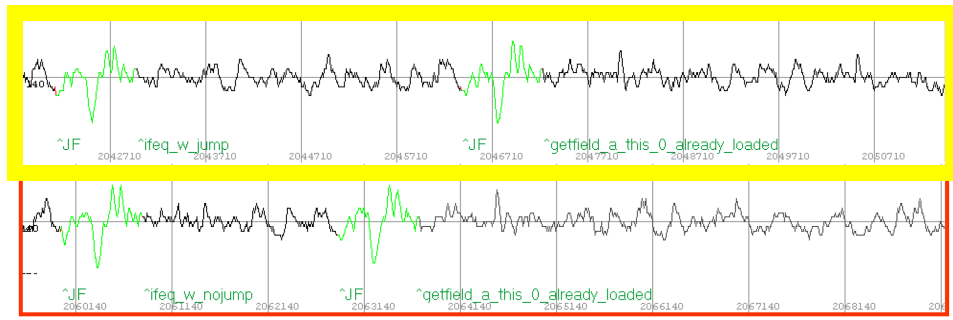
jump not executed (THEN branch)

jump executed (ELSE branch)



Different instruction appearance

```
bool bSimilar = TRUE;
bool bFake = TRUE;
for (short i=0; i<passLength;i++) {
    if (array1[i] != array2[i])
        bSimilar = FALSE;
    else
        bFake = FALSE;
}
```

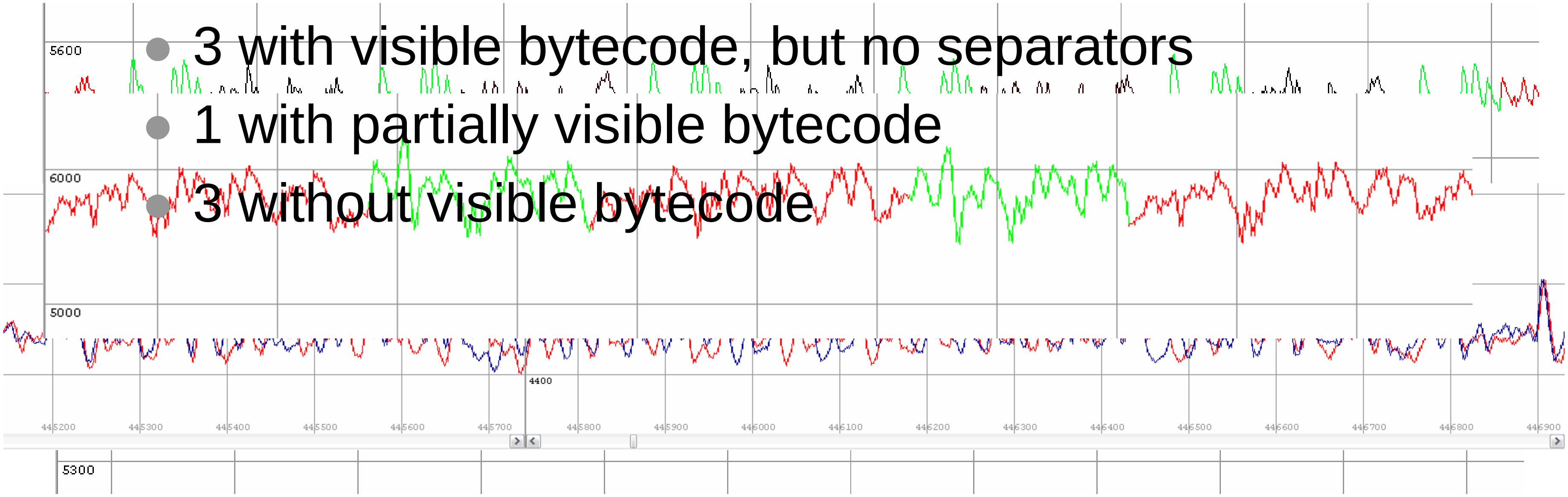



```
getfield_a_this 20;
sload_3;
baload;
getfield_a_this 21;
sload_3;
baload;
if_scmpeq L4;
```

array1	P	A	S	S	W	O	R	D
	=	=						
array2	P	A	A	B	C	D	E	F

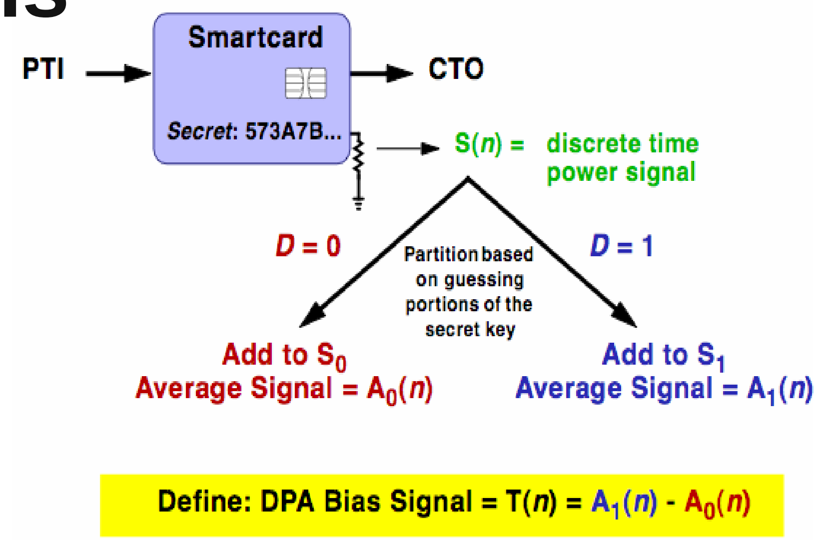
Situation with current smart cards

- Tested 10 different cards from 4 manufactures
  - 3 with clearly visible bytecode and separators
  - 3 with visible bytecode, but no separators
  - 1 with partially visible bytecode
  - 3 without visible bytecode

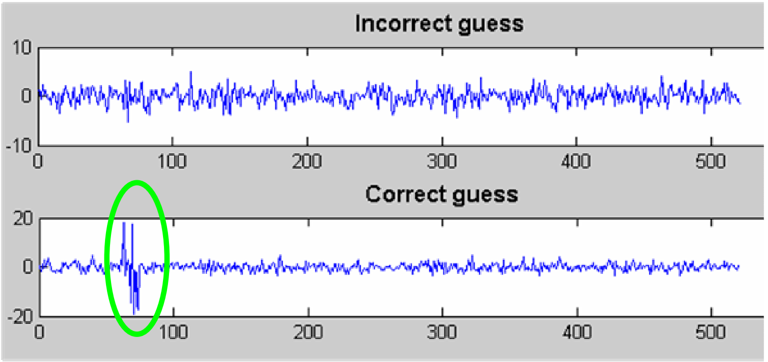


- Caused by used type of the main processor

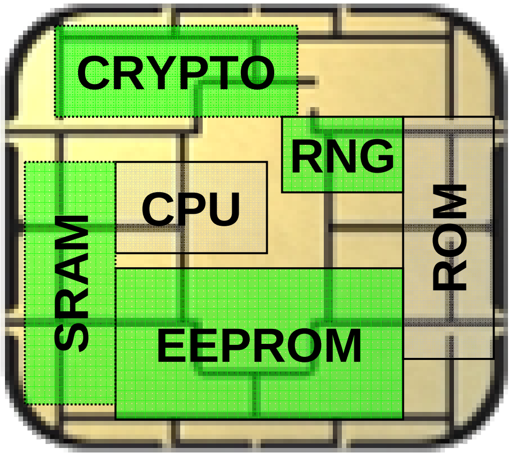
Differential power analysis



- Powerful attack on secret values
  - e.g. encryption keys
- Multiple power traces with key usage
  - 10^3-10^5 traces with known I/O data
  - KEY ⊕ KNOWN_DATA
- Key is guessed byte-per-byte
  - correct guess reveals correlation with traces
  - all possible values of single byte tried (256)
  - traces divided into 2 groups
  - groups are averaged
  - averaged signals are compared
  - significant peaks if guess correct
- No need to know exact implementation
  - big advantage

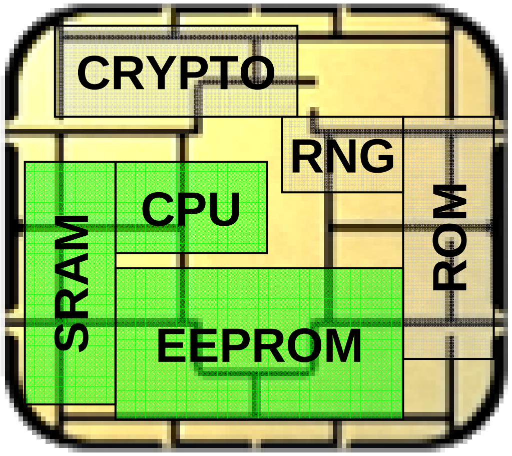


HW protections against power analysis



- Changes on hardware level
  - masking, randomization, dual-rail logics
  - security vs. speed/memory/chip area
- Disadvantages
  - focused mostly on cryptographic coprocessor
  - focused mostly on data protection (algorithm is known)
  - hard to protect general code executed on JavaCard level
- Expensive and non-flexible solution for customer
  - hardware replacement required (price, logistics)

SW protections against power analysis



- Changes on software level
  - best practices & secure coding patterns
  - more flexible, react on actual threats
- Disadvantages
  - limited by underlying hardware
  - may obscure original code functionality
  - additional logical bugs, harder to audit
  - problem with code expandability and maintenance
  - high requirements on developers
- Sometimes the only possibility for a customer

Conclusions

- SC massively deployed (~$5*10^9$), mainly w.r.t. security
  - secure storage, secure code execution
    - on-card asymmetric key generation!
  - wide range of interesting protocols involving smart cards
- Limited memory (10^2 kB) and CPU power (8-32b,5-20MHz)
  - low cost small computer designed specifically for security
  - crypto operation accelerated by co-processors
- Can be programmed
  - free tools are available, single cards can be ordered
  - try it by yourself (reader + card ~30 euro)
- Can be attacked
  - need for special knowledge and equipment
  - still far more secure than standard PC

Thank you for your attention!

Questions

References

- PKCS#11, http://www.rsa.com/rsalabs/node.asp?id=2133
- Java Card SDK http://java.sun.com/javacard/devkit/
- GPShell, http://sourceforge.net/projects/globalplatform/
- OpenSC project, http://www.opensc-project.org
- Power Analysis Attacks, S. Mangard, S. Mangard, E. Oswald, T. Popp, 2007

Remotely Keyed Encryption idea

- Requirements:
  - fast encryption based on host power
  - key should never leaves smartcard
  - encryption/decryption is possible only when smartcard is present
- Idea: use on-card encryption, but move heavy work to PC in secure way
  - Remotely Keyed Encryption (Blaze 1996)
- Basic/strong attacker model
  - attacker with temporary access to the smartcard

Call diagram

1. Initial request V_1, file (in)dependent

Process request, *save State*

2. Response R_1 depends on V_1 and key

Encrypt file with R_1

3. Request V_2 depends on encrypted file

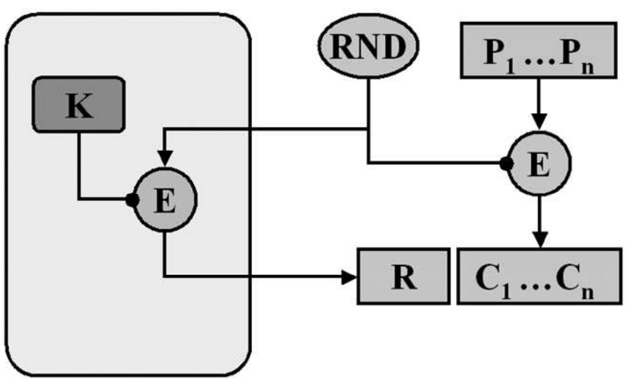
4. Response R_2 depends on key, V_2 and State

Modify file with R_2

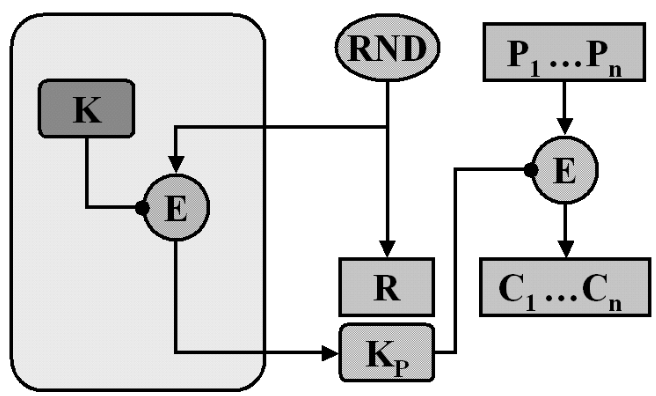
I1 and THCEP schemes

- Fast modes for basic attacker model
  - requires only 1 APDU message
  - THCEP for authentication-only cards (export issues)
- Problems
  - key independent of file
  - attacker can “pre-generate” keys

I1

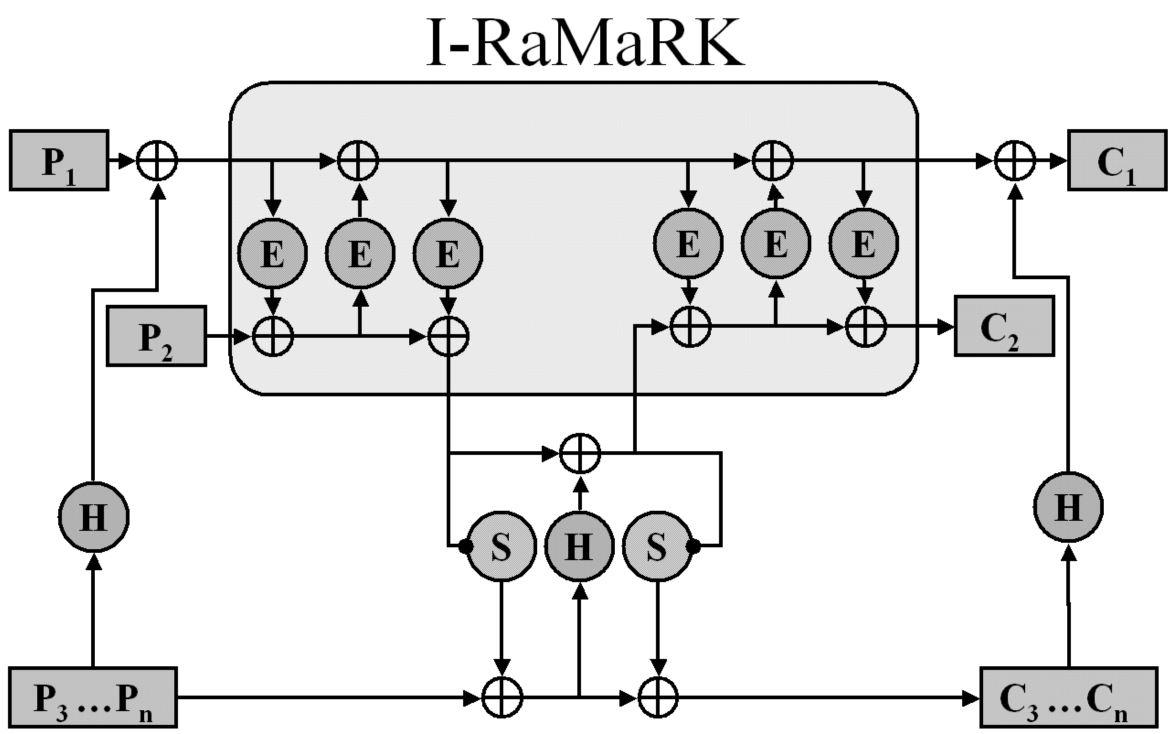


THCEP



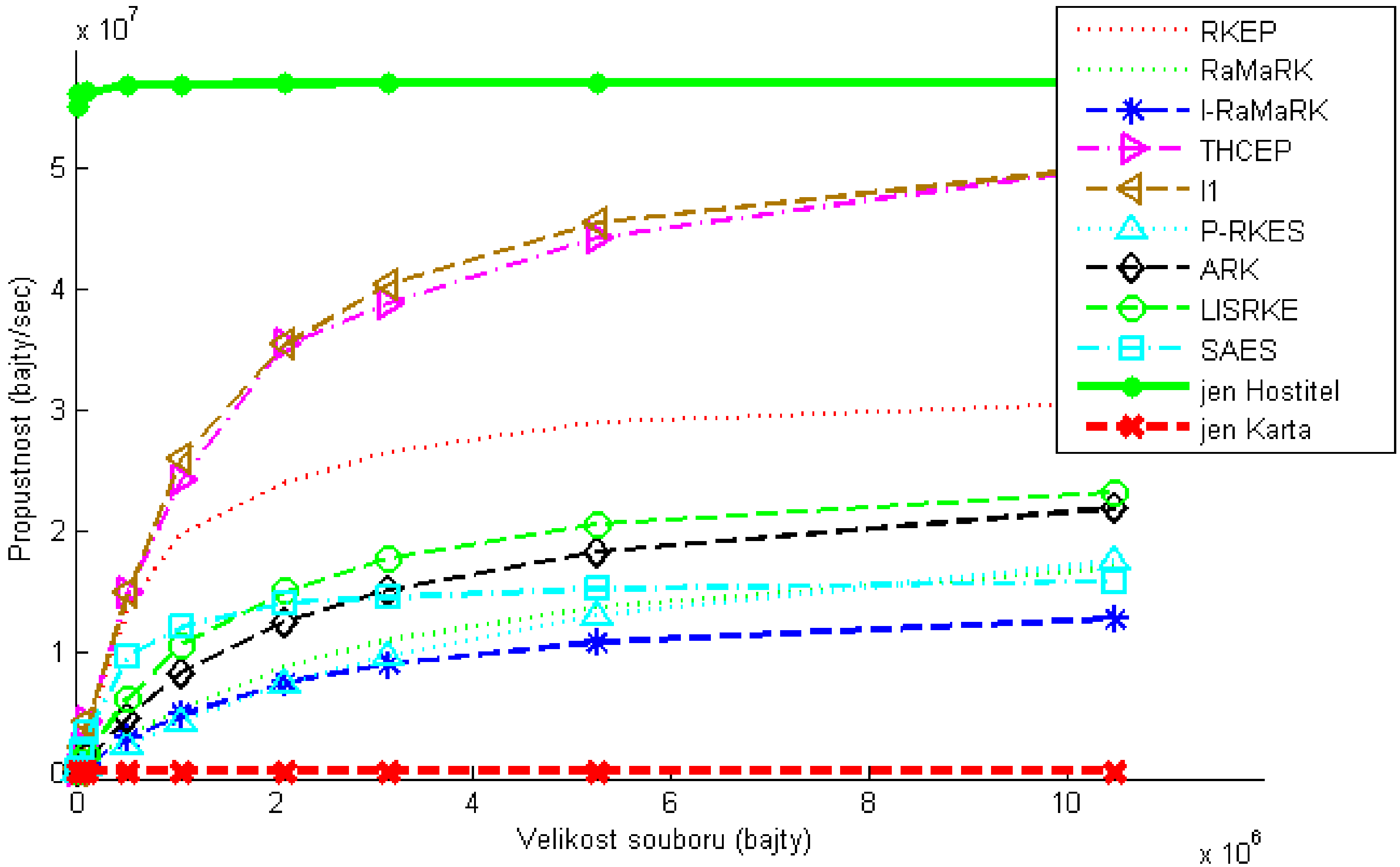
I-RaMaRK

- Secure mode for strong attacker model
- Requires 2 APDU messages, slow



Performance comparison

- AMD Athlon 2500+, GXPLite-Generic (JavaCard)
- all modes, 32B-10MB (B/s)



Attacks against API

- Dedicated hardware device
    - intermediate device simulating both reader and card
    - prototype developed (with VUT Brno)
- Low level data manipulation tool