# PV079: Cryptographic smartcards and their applications

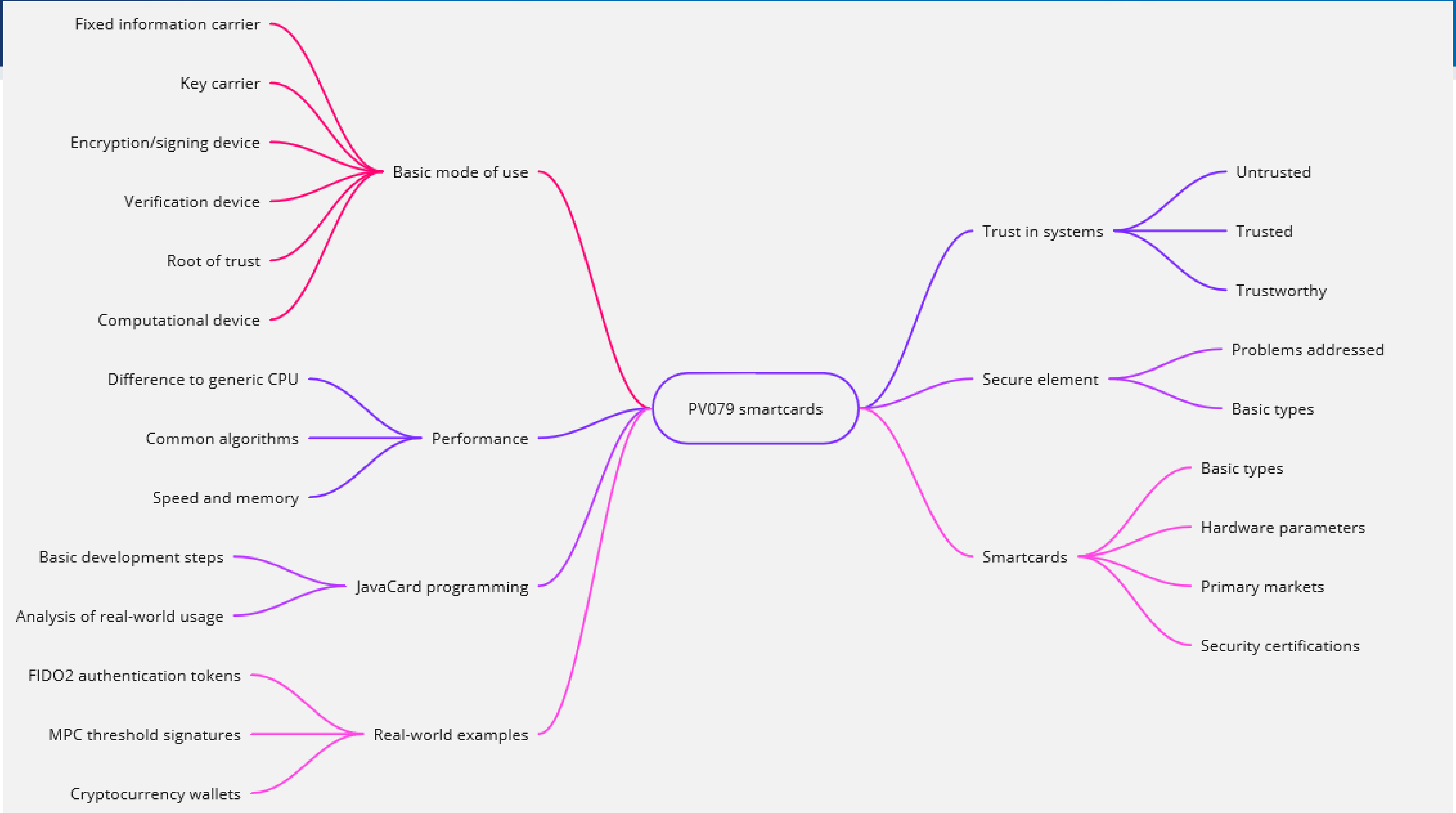
## Cryptographic secure hardware

[Petr Švenda](#) svenda@fi.muni.cz @rngsec
Centre for Research on Cryptography and Security, Masaryk University

CROCS
Centre for Research on Cryptography and Security

# Plan for today

1. Secure elements – Why we need them?
2. Applications – Where and how to use?
3. Smartcard programming – How to develop own application?
4. Interesting real-world examples


PV079 smartcards
Basic mode of use
Fixed information carrier
Key carrier
Encryption/signing device
Verification device
Root of trust
Computational device
Performance
Difference to generic CPU
Common algorithms
Speed and memory
JavaCard programming
Basic development steps
Analysis of real-world usage
Real-world examples
FIDO2 authentication tokens
MPC threshold signatures
Cryptocurrency wallets
Trust in systems
Untrusted
Trusted
Trustworthy
Secure element
Problems addressed
Basic types
Smartcards
Basic types
Hardware parameters
Primary markets
Security certifications

# UNTRUSTED
# VS.
# TRUSTED
# VS.
# TRUSTWORTHY

- **Untrusted system**
  - System itself explicitly unable to fulfill some security policy
  - Additional layer of protection must be employed (encrypt before store, sign before send…)
  - Not itself a bad property – system cannot fail us as we do not expect security guarantees
- **Trusted system**
  - *"…system that is relied upon to a specified extent to enforce a specified security policy. As such, a trusted system is one whose failure may break a specified security policy."* (TCSEC, Orange Book)
  - Component which harms our security if misfunction
- **Trustworthy system**
  - *"Computer system where software, hardware, and procedures are secure, available and functional and adhere to security practices"* (Black's Law Dict.)
  - User have reasons to trust (e.g., was heavily tested and scrutinized)

# TRUSTED SECURE ELEMENT

# What exactly can be secure element (SE)?

- Anything user is willing to trust for provision of security ☺
  - Depends on definition of “trust” and definition of “element” and “secure”
  - We will use narrower definition
- Trusted element is element (hardware, software or both) in the system intended to increase security *level* w.r.t. situation without the presence of such element
  1. Paper cheque vs. payment card with magnetic stripe vs. card with chip (smartcard)
  2. User authenticating with password vs. One-Time-Password generator
  3. Feature phone vs. phone with secure enclave for keys
  4. (Bank vs. bank with metal safe)

# What problems are secure elements addressing?

- What problems are secure elements addressing?
  - Secure storage (keys and sensitive data)
  - Protected secrets even if physically attacked (tamper resistant)
  - Secure (cryptographic) computational device (signature, authentication)
  - Hardware root of trust (initial check of boot sequence)
  - Unspoofable logging
  - Enforcement of specific policy (PIN before sign, four eyes policy…)
  - Easy to carry, easy to embed into another device, low battery usage

Which of these can’t be solved with laptop or cell phone?

# INTRO TO SMART CARDS

# Basic types of (smart) cards

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

# Basic types of (smart) cards (2)

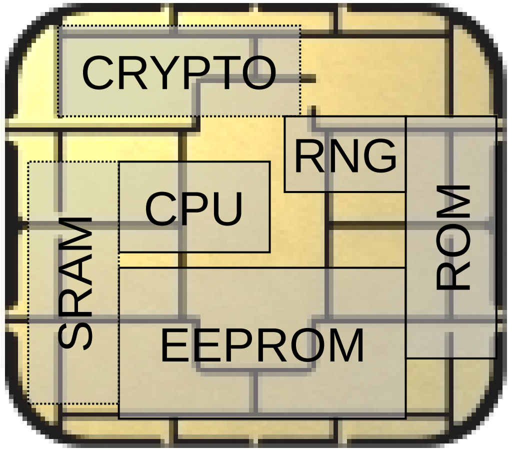

4. Cryptographic smart cards
   - Support for (real) cryptographic algorithms
   - Mifare Classic ($1), Mifare DESFire ($3)

We will mainly focus on categories 4 and 5

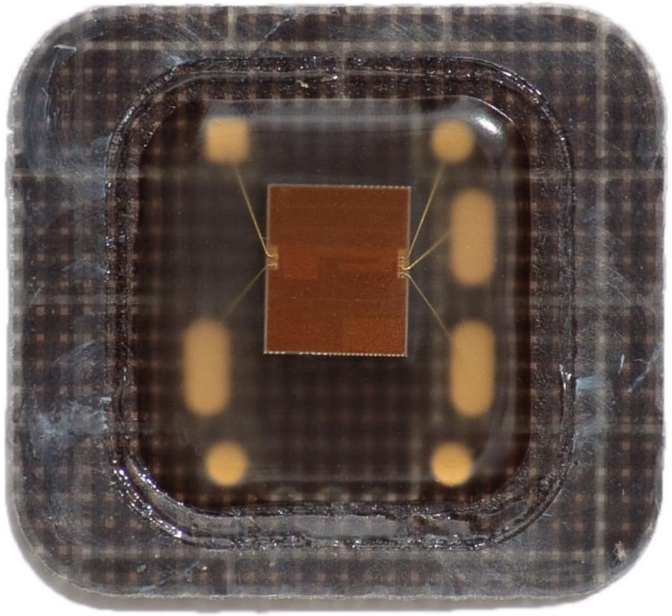
5. User-programmable cryptographic smart cards
   - JavaCard, .NET card, MULTOS cards ($2-$30)
   - Chip manufacturers: NXP, Infineon, Gemalto, G&D, Oberthur, STM, Atmel, Samsung...
6. Secure environment (enclave) inside more complex CPUs
   - ARM TrustZone, Intel SGX…

*Credit Wikimedia Commons*

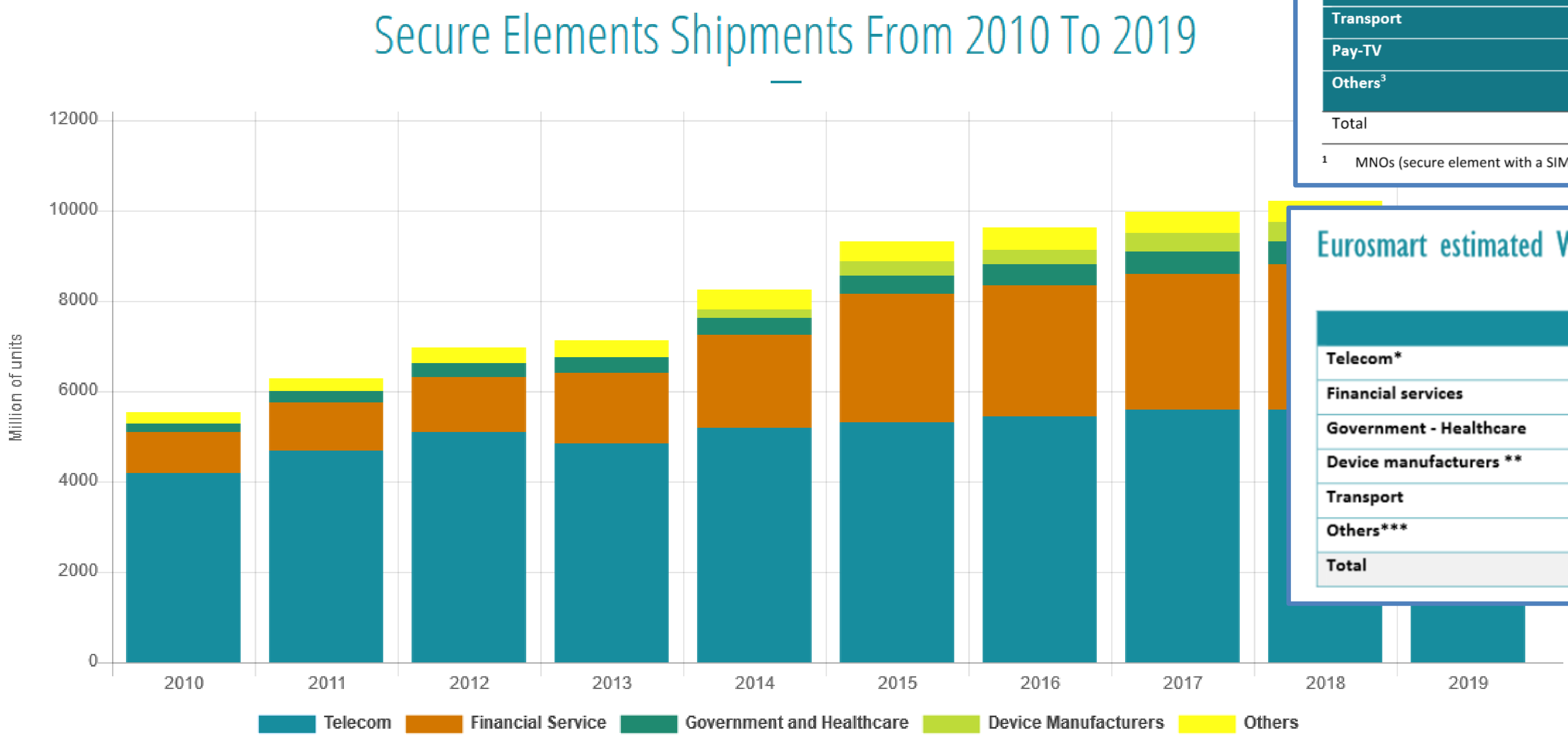
# Cryptographic smart cards




- SC is quite powerful device
  - 8-32 bit processor @ 5-50MHz
  - persistent memory 32-200+kB (EEPROM)
  - volatile fast RAM, usually <<10kB
  - truly random generator
  - cryptographic coprocessor (3DES,AES,RSA-2048,ECC...)
- ~9.3 billion units shipped in 2021 (EUROSMART)
  - mostly smart cards, telco, payment and loyalty...
  - ~3 billion contactless (EUROSMART)
- For environments where attacker has physical access
  - NIST FIPS140-2 standard, security Level 4
  - Common Criteria EAL4-6+



# Primary markets for smartcards




January 2022

Eurosmart estimated WW µP 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*** |

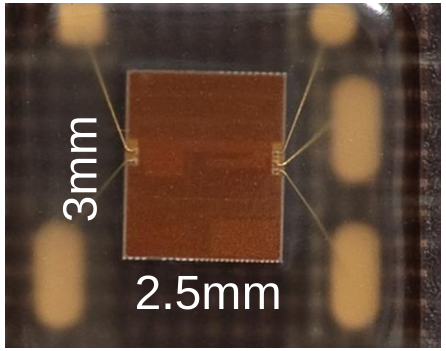
[1] MNOs (secure element with a SIM application)

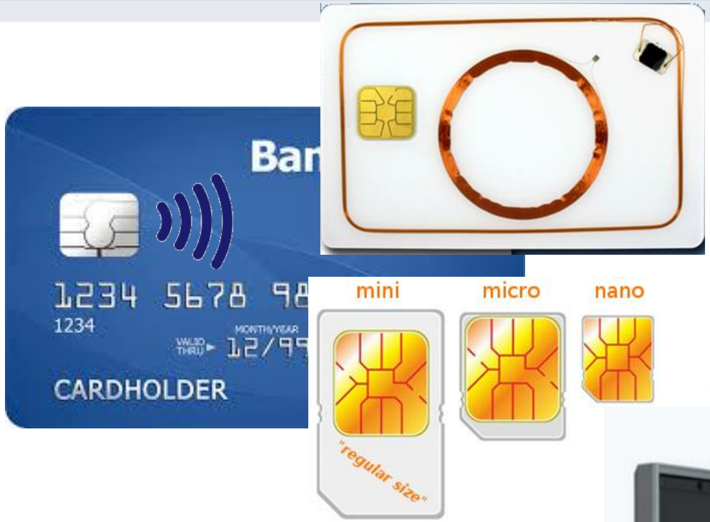
Eurosmart estimated WW µP TAM - (Mu)

| | 2021 | 2022 forecasts |
|---|---|---|
| **Telecom*** | 4700 | 4600 |
| **Financial services** | 3250 | 3200 - 3300 |
| **Government - Healthcare** | 510 | 550 |
| **Device manufacturers **** | 490 | 520 |
| **Transport** | 220 | 220-245 |
| **Others***** | 155 | 150 |
| **Total** | **9325** | **9.240 - 9. 360** |

*https://www.eurosmart.com/eurosmarts-secure-elements-market-analysis-and-forecasts/*
*https://www.eurosmart.com/2021-secure-elements-global-market-and-2022-estimates/*

# Smart cards forms






- Many possible forms
  - ISO 7816 standard
  - SIM size, USB dongles, Java rings, implants…
- Contact(-less), hybrid/dual interface
  - contact physical interface
  - contact-less interface (NFC phone can communicate!)
  - hybrid card – separate logics on single card
  - dual interface – same chip accessible contact & c-less
- Card emulation (contactless)
  1. Card emulation mode (physical in-phone secure element)
  2. Host-based card emulation (without physical element)
     - Apple/Google/Samsung… Pay



http://simcardsize.com/sim-card-sizes/ https://shop.cobo.com/products/cobo-vault-essential https://www.intel.com/ https://yubico.com

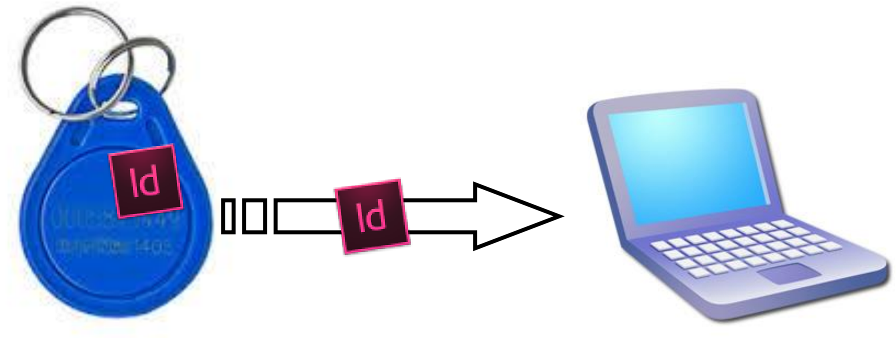
# Smart card is highly protected device

- Intended for physically unprotected environment
  - NIST FIPS140-2 standard, security Level 4
  - Common Criteria EAL5+/6+…
- Tamper protection
  - Tamper-evidence (visible if physically manipulated)
  - Tamper-resistance (can withstand physical attack)
  - Tamper-response (erase keys…)
- Protection against side-channel attacks (timing, power, EM)
- Periodic tests of TRNG functionality
- Approved crypto algorithms and key management
- Limited interface, smaller trusted computing base (than usual)
  - http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm
- Designed for security and certified != secure

# BASIC MODES OF USAGE

# Secure element carries fixed information

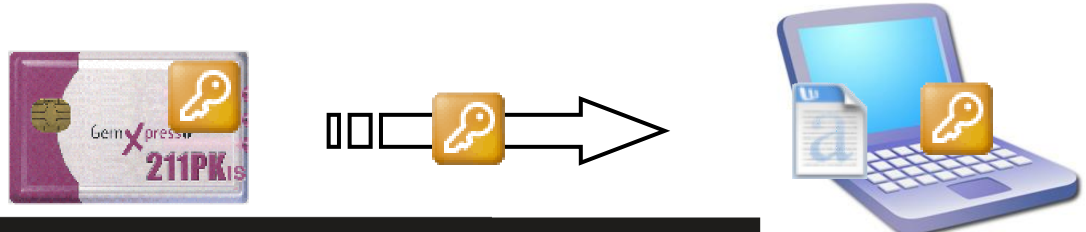
- Fixed information ID transmitted, no secure channel
- Low-cost solution (nothing “smart” needed)
- Problem: Attacker can eavesdrop and clone chip




# Secure element as a secure carrier

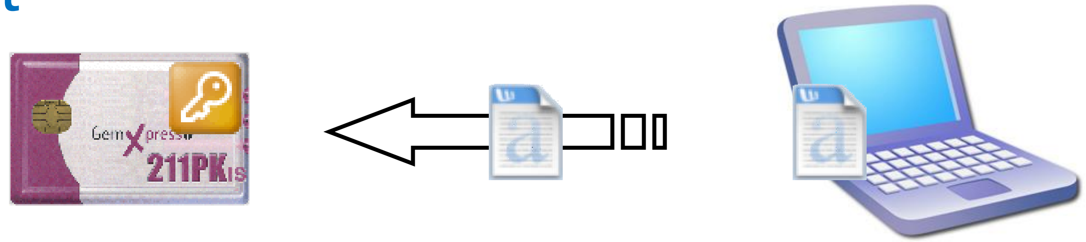
- Key(s) stored on a card, loaded to a PC before encryption/signing/authentication, then erased
- High speed usage of key possible (>>MB/sec)

- Attacker with an access to PC during operation will obtain the key
  – key protected for transport, but not during the usage
- Secure element can be embedded into another device
  – Into hardware wallet – stored seed loaded before use
  – Card with keys plugged into larger Hardware Security Module (HSM)



# Secure element as encryption/signing device

- PC just sends data for encryption/signing…
- Key never leaves the secure element
  - – personalized in secure environment
  - – protected during transport and usage



- Attacker must attack the secure element
  - – or wait until is inserted and PIN entered!
- Performance depends on the parameters of secure element
  - – Low speed encryption (~kB/sec) for smartcards
    - low communication speed / limited card performance
  - – High speed for cryptographic accelerators (communication + fast HW)

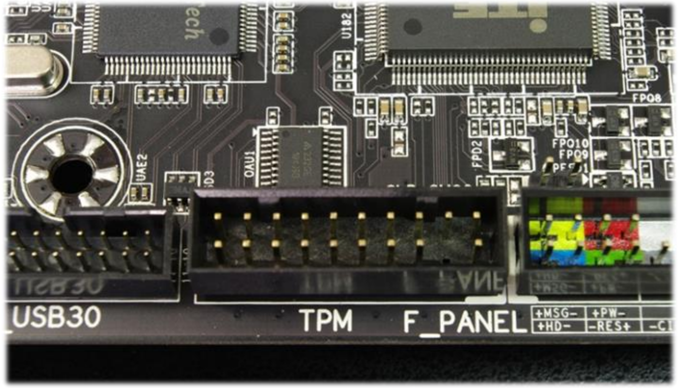
# Secure element as verification device

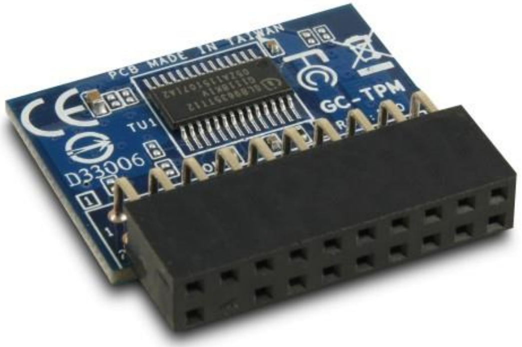
- Device with lower overall security embeds secure element for sensitive tasks, invokes it via dedicated API
  - – E.g., secure element in mobile phones
- Sensitive data (keys, fingerprint, password) never leaves SE
  - – Limits exposure of sensitive data

- Attacker must attack secure element to extract secrets
  - – or redirect calling application to itself!
  - – How se fingerprint to check and response transmitted?
  - – Requires secure channel between components

# Secure element as root of trust (TPM)

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

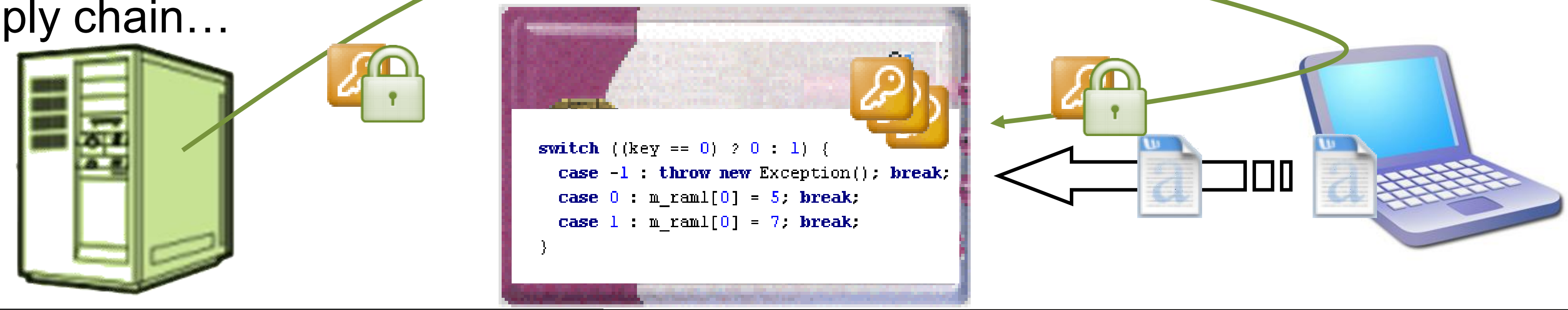


# Secure element as computational device

- PC just sends input for application running on secure element
- Application code & keys never leave the secure element
  - card can perform complicated programmable actions
  - new code can be uploaded remotely
  - 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 secure element or initial input
  - Or developer, supply chain…



# For whom is SE trusted? Who is an attacker?

- Payment smart card
  - for issuing bank
- SIM card
  - for phone carriers
- Trusted Platform Module (TPM)
  - for user as storage of Bitlocker keys, TE for remote entity during attestation
- Trusted Execution Environment in mobile/set-top box
  - for issuer for confidentiality and integrity of code handling stream decryption keys
- Hardware Security Module for TLS keys
  - for web admin to protect server's private key
- Energy meter
  - for utility company to measure real consumption
- Tachograph
  - for compliance control (limit driving time)
- AWS KMS, Azure KeyVault
  - for user to protect keys against cloud operator (to same extend)

# Application domains changes in time

- Cheap yet relatively hard to attack despite physical access
  - Sensitive data can be stored and used yet carried in pocket
  - Protection against the end-user (SIM, satellite decoders…)
- But we now have smartphones!
  - Payments via Apple Pay, Google Pay without physical smartcard
    - Still uses VISA/Mastercard payment infrastructure
  - Smartphones can make smartcards obsolete in large portion of previous usage domains!
- But smartphones are also quite too complex (=> bugs)
  - Sensitive data / keys etc. on smartphone are more vulnerable
- New use-cases
  - Trusted Platform Module (smartcard on the motherboard)
  - FIDO2 U2F/WebAuthn tokens (improved authentication tokens, mostly solves URL phishing attack!)
  - Cryptocurrency hardware wallets (smartcard with trusted display)

# SMARTCARD ALGORITHMS AND PERFORMANCE

# Performance

- Performance is dependent on multiple factors
  - Base clock speed, instruction set, caches, available RAM, parallelism, algorithm implementation, communication speed…
- Difference between standard CPU and smartcard
  - Low clock frequency (<50MHz), no parallelism
  - Small RAM (need too offload data to slower memory)
- How is one supposed to run asymmetric cryptography fast enough?
  - If base CPU is slow (50MHz) and memory small (<10kB)
- Answer: dedicated co-processors for particular operations (AES, RSA…)
  - Faster and also more protected against side-channels

CRoCS

# Common algorithms

- Basic - cryptographic co-processor
  - Truly random data generator
  - 3DES, AES128/256, (national algorithms)
  - MD5, SHA1, SHA-2 256/512
  - RSA (up to 2048b common, 4096 possible)
  - ECC (up to 256b common, 521b possible)
  - Diffie-Hellman key exchange (DH/ECDSA)
- Custom code running in secure environment
  - E.g., HMAC, OTP code, re-encryption
  - Might be significantly slower (e.g., SW AES 50x slower)

| Feature | First in version | JC ≤ 2.2.1 (21 cards) | JC 2.2.2 (26 cards) | JC 3.0.1/2 (12 cards) | JC 3.0.4 (29 cards) | JC 3.0.5 (11 cards) |
|---|---|---|---|---|---|---|
| *Truly random number generator* | | | | | | |
| **TRNG** (ALG_SECURE_RANDOM) | ≤ 2.1 | 100% | 100% | 100% | 100% | 100% |
| *Block ciphers used for encryption or MAC* | | | | | | |
| **DES** (ALG_DES_CBC_NOPAD) | ≤ 2.1 | 100% | 100% | 100% | 100% | 100% |
| **AES** (ALG_AES_BLOCK_128_CBC_NOPAD) | 2.2.0 | 52% | 96% | 100% | 100% | 100% |
| **KOREAN SEED** (ALG_KOREAN_SEED_CBC_NOPAD) | 2.2.2 | 5% | 62% | 75% | 34% | 0% |
| *Public-key algorithms based on modular arithmetic* | | | | | | |
| **1024-bit RSA** (ALG_RSA(_CRT) LENGTH_RSA_1024) | ≤ 2.1 | 76% | 96% | 100% | 93% | 82% |
| **2048-bit RSA** (ALG_RSA(_CRT) LENGTH_RSA_2048) | ≤ 2.1 | 67% | 96% | 100% | 93% | 82% |
| **4096-bit RSA** (ALG_RSA(_CRT) LENGTH_RSA_4096) | 3.0.1 | 0% | 0% | 0% | 3% | 0% |
| **1024-bit DSA** (ALG_DSA LENGTH_DSA_1024) | ≤ 2.1 | 5% | 8% | 8% | 10% | 0% |
| *Public-key algorithms based on elliptic curves* | | | | | | |
| **192-bit ECC** (ALG_EC_FP LENGTH_EC_FP_192) | 2.2.1 | 5% | 62% | 83% | 66% | 82% |
| **256-bit ECC** (ALG_EC_FP LENGTH_EC_FP_256) | 3.0.1 | 0% | 50% | 75% | 66% | 82% |
| **384-bit ECC** (ALG_EC_FP LENGTH_EC_FP_384) | 3.0.1 | 0% | 12% | 17% | 62% | 82% |
| **521-bit ECC** (ALG_EC_FP LENGTH_EC_FP_521) | 3.0.4 | 0% | 4% | 8% | 45% | 82% |
| **ECDSA SHA-1** (ALG_ECDSA_SHA) | 2.2.0 | 24% | 84% | 100% | 69% | 82% |
| **ECDSA SHA-2** (ALG_ECDSA_SHA_256) | 3.0.1 | 5% | 12% | 100% | 69% | 82% |
| **ECDH IEEE P1363** (ALG_EC_SVDP_DH) | 2.2.1 | 29% | 81% | 100% | 69% | 82% |
| **IEEE P1363 plain coord. X** (ALG_EC_SVDP_DH_PLAIN) | 3.0.1 | 5% | 4% | 67% | 48% | 82% |
| **IEEE P1363 plain c. X,Y** (ALG_EC_SVDP_DH_PLAIN_XY) | 3.0.5 | 0% | 0% | 0% | 17% | 82% |
| *Modes of operation and padding modes* | | | | | | |
| **ECB, CBC modes** | ≤ 2.1 | 100% | 100% | 100% | 100% | 100% |
| **CCM, GCM modes** (CIPHER_AES_CCM, CIPHER_AES_GCM) | 3.0.5 | 0% | 0% | 0% | 0% | 0% |
| **PKCS1, NOPAD padding** | ≤ 2.1 | 95% | 100% | 100% | 100% | 100% |
| **PKCS1 OAEP scheme** (ALG_RSA_PKCS1_OAEP) | ≤ 2.1 | 14% | 31% | 8% | 41% | 82% |
| **PKCS1 PSS sheme** (ALG_RSA_SHA_PKCS1_PSS) | 3.0.1 | 14% | 19% | 83% | 41% | 100% |
| **ISO14888 padding** (ALG_RSA_ISO14888) | ≤ 2.1 | 14% | 12% | 8% | 0% | 0% |
| **ISO9796 padding** (ALG_RSA_SHA_ISO9796) | ≤ 2.1 | 81% | 100% | 100% | 86% | 100% |
| **ISO9797 padding** (ALG_DES_MAC8_ISO9797_M1/M2) | ≤ 2.1 | 90% | 100% | 100% | 100% | 100% |
| *Hash functions* | | | | | | |
| **MD5** (ALG_MD5) | ≤ 2.1 | 90% | 77% | 92% | 62% | 0% |
| **SHA-1** (ALG_SHA) | ≤ 2.1 | 95% | 100% | 100% | 100% | 100% |
| **SHA-256** (ALG_SHA_256) | 2.2.2 | 14% | 88% | 100% | 97% | 100% |
| **SHA-512** (ALG_SHA_512) | 2.2.2 | 5% | 23% | 25% | 90% | 100% |
| **SHA-3** (ALG_SHA3_256) | 3.0.5 | 0% | 0% | 0% | 0% | 0% |

Table 1: The level of support for algorithms specified in JavaCard API. For a given feature, the *version* column specifies the JavaCard specification that defined it first, while the subsequent columns show its availability in cards reporting particular supported version via the *JCSystem.getVersion*() method and maximally supported version of the *javacard.framework* package. Results for smartcards with an unknown version were not included.

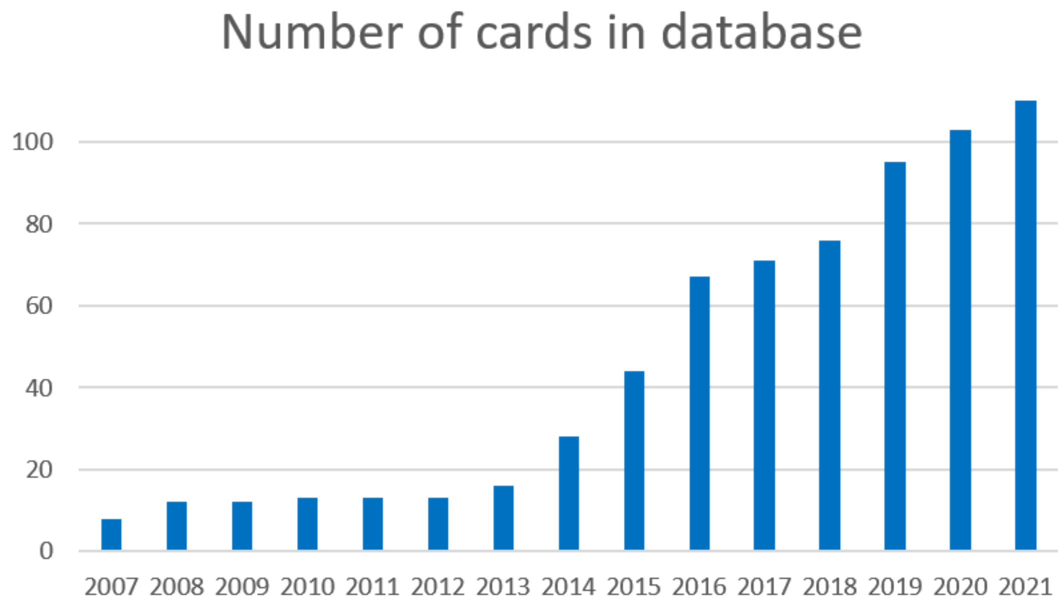
# What is the typical performance?

- Hardware differ significantly
  - Clock multiplier, memory speed, crypto coprocessor…
- Typical speed of operation is:
  - Milliseconds (RNG, symmetric crypto, hash)
  - Tens of milliseconds (transfer data in/out)
  - Hundreds of millisecond (asymmetric crypto)
  - Seconds (RSA keypair generation)
- Operation may consists from multiple steps
  - Transmit data, prepare key, prepare engine, encrypt
    - → additional performance penalty
  - Usability rule of thumb: operation shall finish in 1-1.5sec

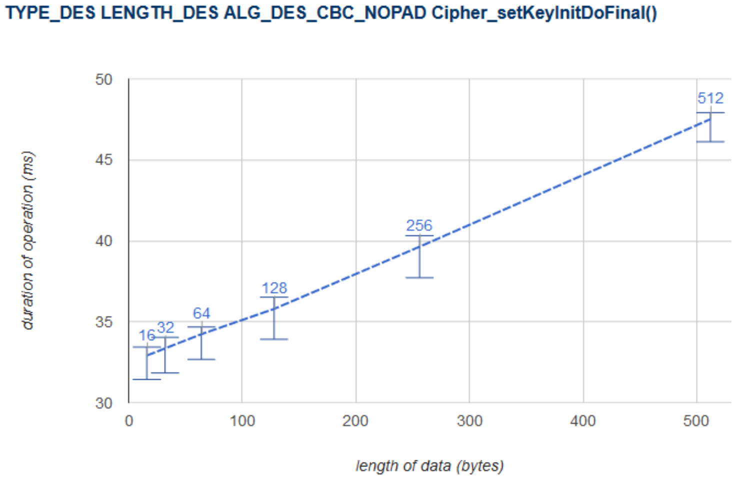
How we know?

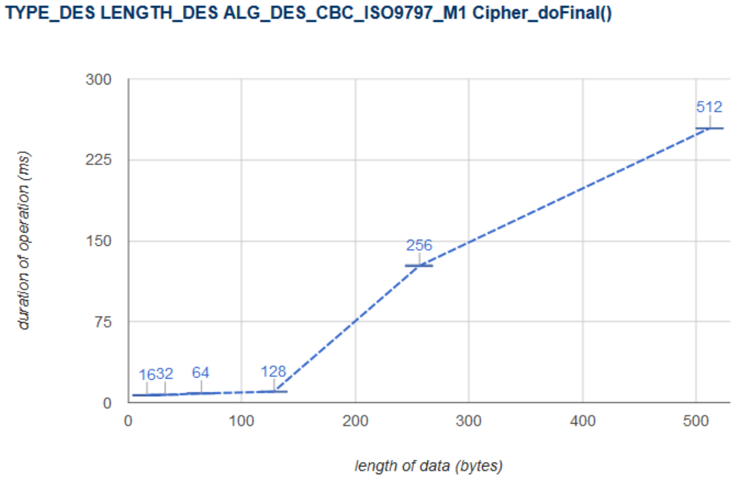
Read from specs, from certification reports, or probe directly!

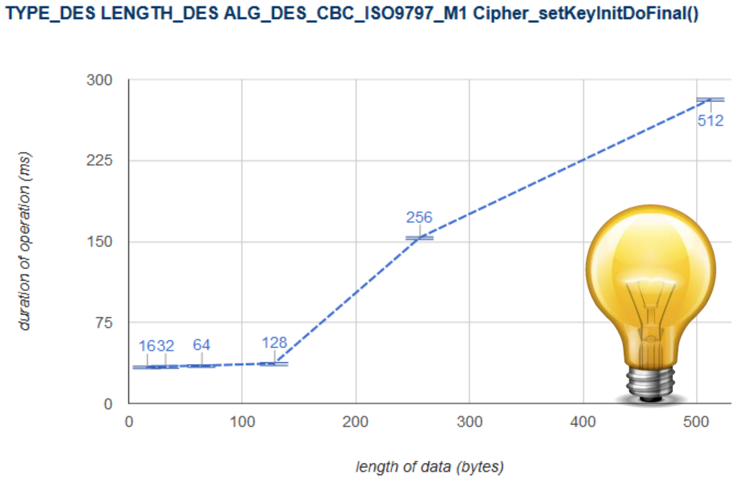
- **JCAlgTest: Robust identification metadata for certified smartcards**, Petr Svenda, Rudolf Kvasnovsky, Imrich Nagy, Antonin Dufka, 19th International Conference on Security and Cryptography (SECRYPT'22), pp.597-604, INSTICC, 2022.
  – https://crocs.fi.muni.cz/papers/jcalgtest_secrypt22

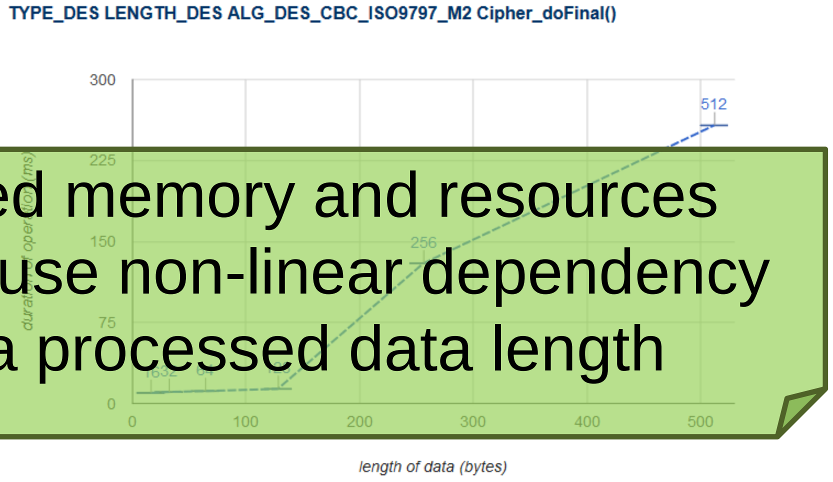


# Performance with variable data lengths



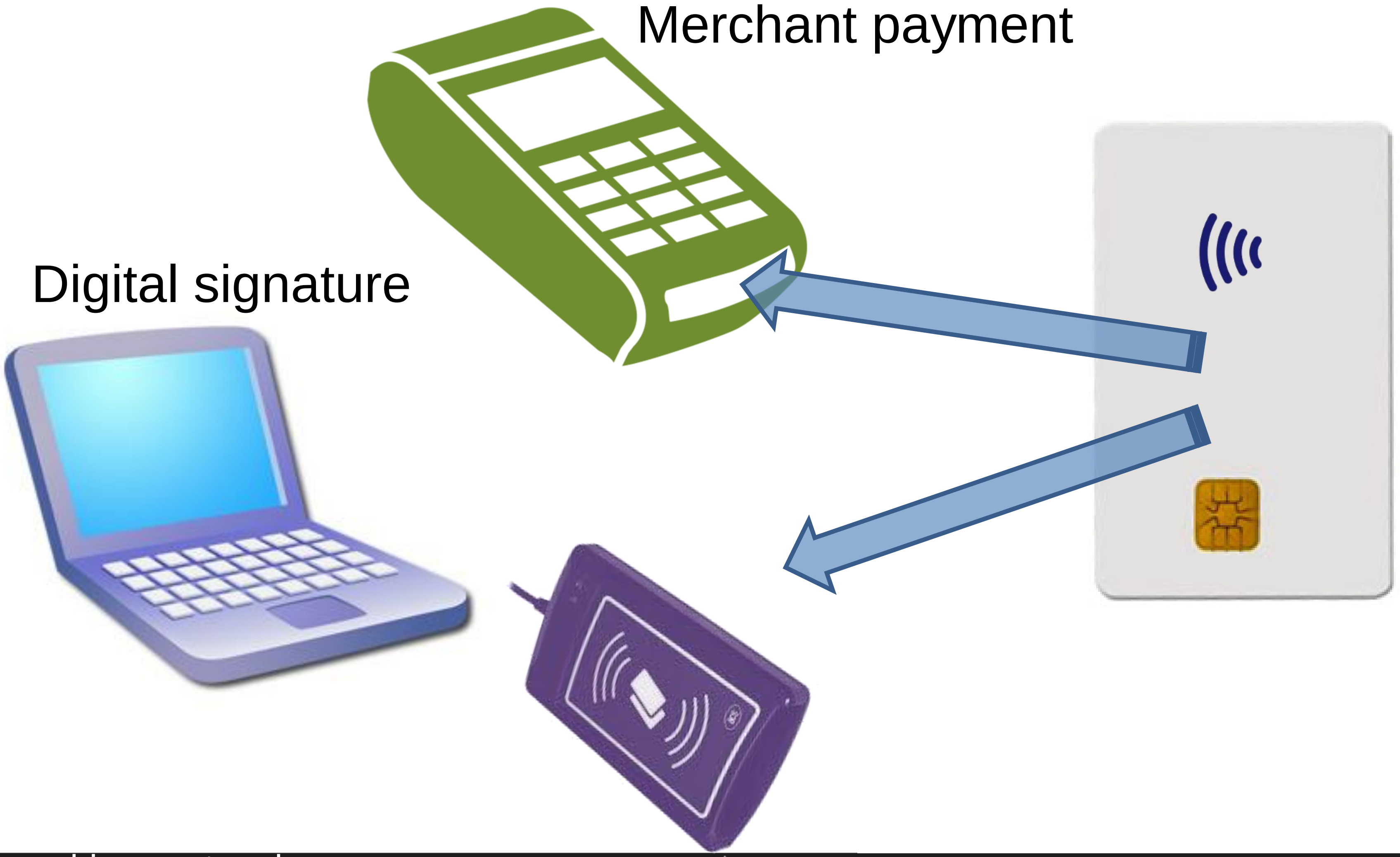




Limited memory and resources may cause non-linear dependency on a processed data length

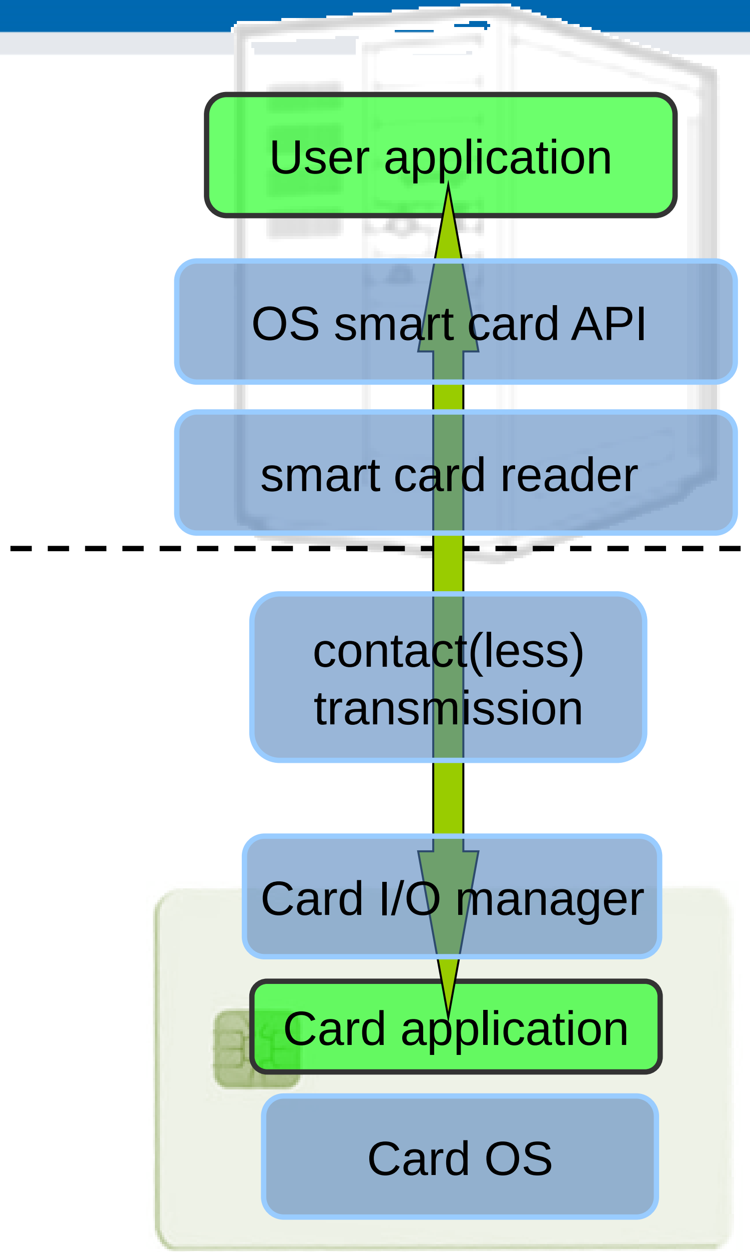
# Smartcard programming, use from external programs

# Big picture – terminal/reader and card

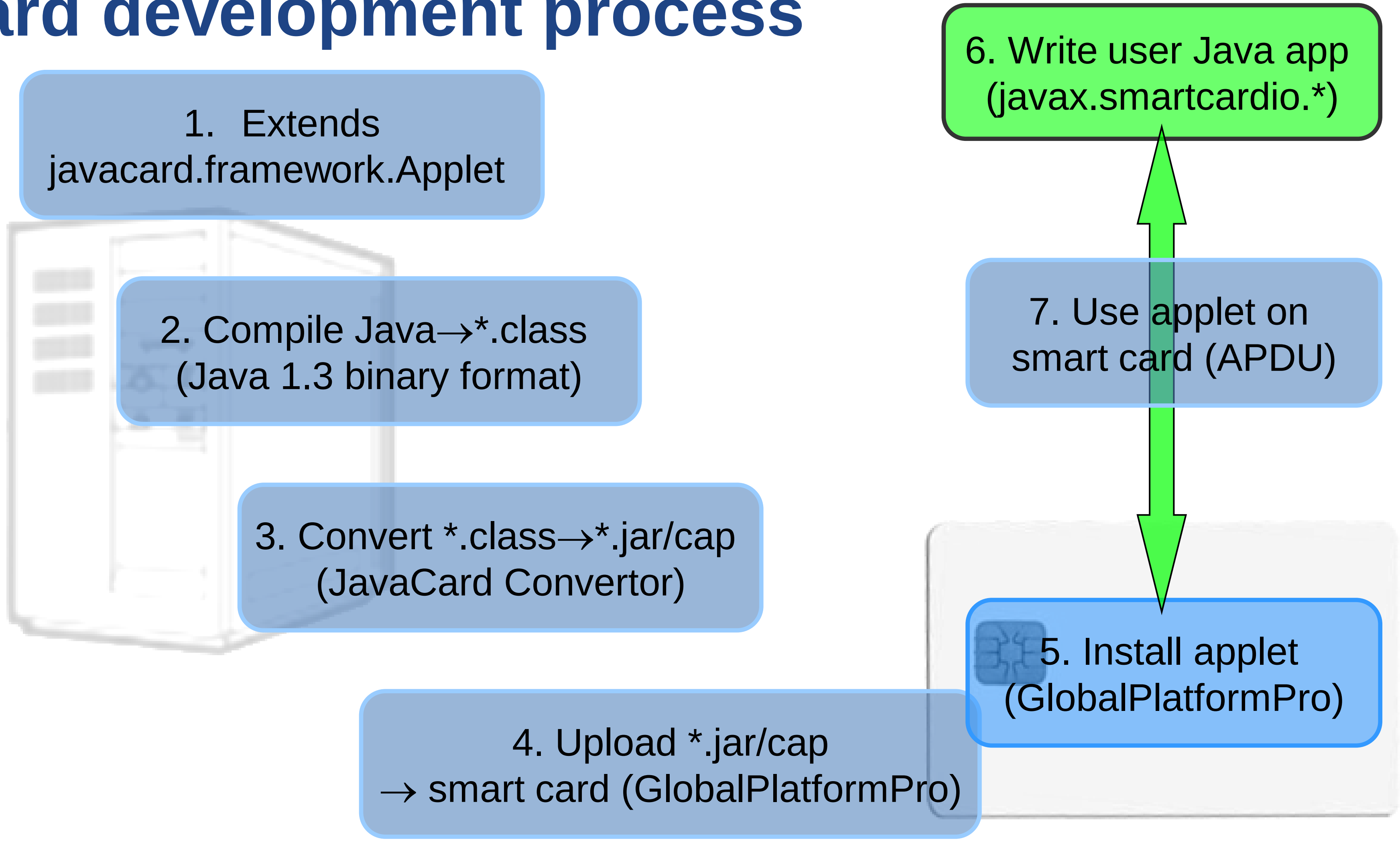


# Big picture - components

- User application
  - Merchant terminal GUI
  - Banking transfer GUI
  - Browser TLS
  - …
- Card application
  - EMV applet for payments
  - SIM applet for GSM
  - OpenPGP applet for PGP
  - U2F applet for FIDO authentication
  - …



# How to develop on-card application?
# JavaCard development process



# Pains for users/developers

- Closed-source, IP-heavy, NDA-based industry
- Primary users for manufactures/vendors are large customers
  - Little interest in small / niche users (< 100k units)
  - Important API proprietary and/or not accessible (ARM TrustZone, proprietary JC packages, detailed specs…)
  - Supply chain issues (resellers, difficult to securely obtain card)
- What is open and available
  - Open API for applets (JavaCard API)
  - Open-source development toolchain for JavaCard
  - Common Criteria and FIPS140-2 certificates (but details omitted)
  - Results of reverse engineering

# How to analyze real-world usage of technology X?

1. Collect representative sample of users / projects (ideally “all”)
   – E.g., all open-source JavaCard projects on GitHub
2. Establish significance of projects
   – E.g., Number of developers/forks/stars, search trends on Google, sales stats…
3. Analyze projects for the level and style of use of technology X
   – E.g., static code analysis of JavaCard keywords and constants
   – Ideally trends in time if possible (e.g., code state in time via git commits)

- *“The adoption rate of JavaCard features by certified products and open-source projects”, L. Zaoral, A. Dufka, P.Svenda, CARDIS’23*

# Certified smartcards and JavaCard-related projects

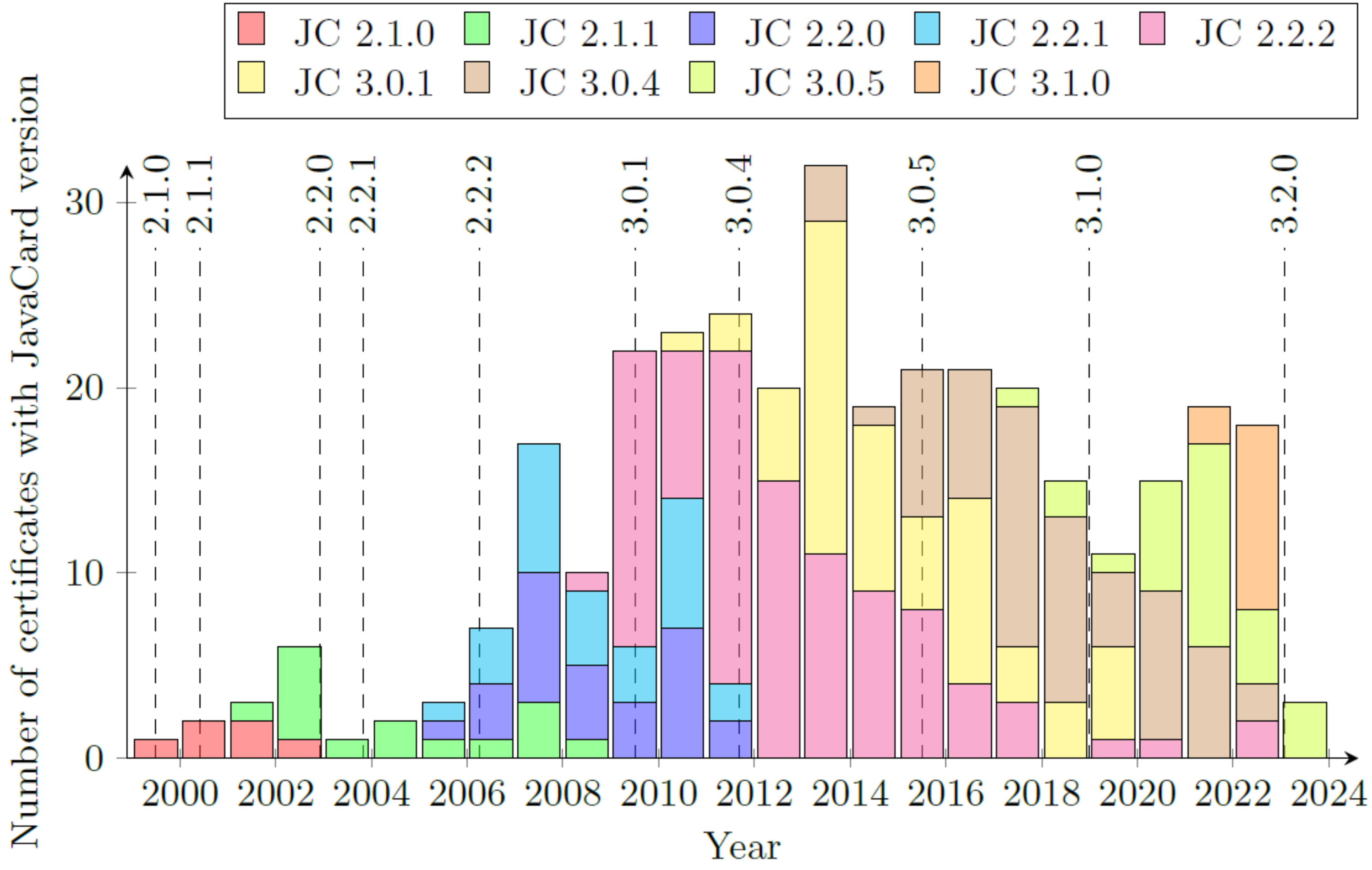



**Fig. 1.** The number of certification documents mentioning specific JavaCard API version per year (the year 2023 only till June). In case multiple versions were detected in a document, only the latest one was included in the plot.

# Activity of open-source JavaCard applets in time

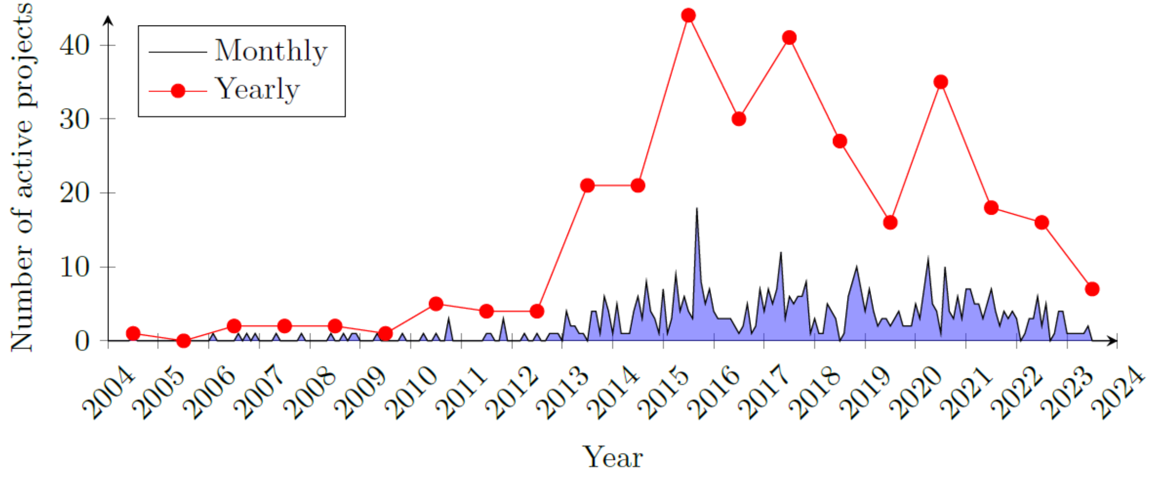
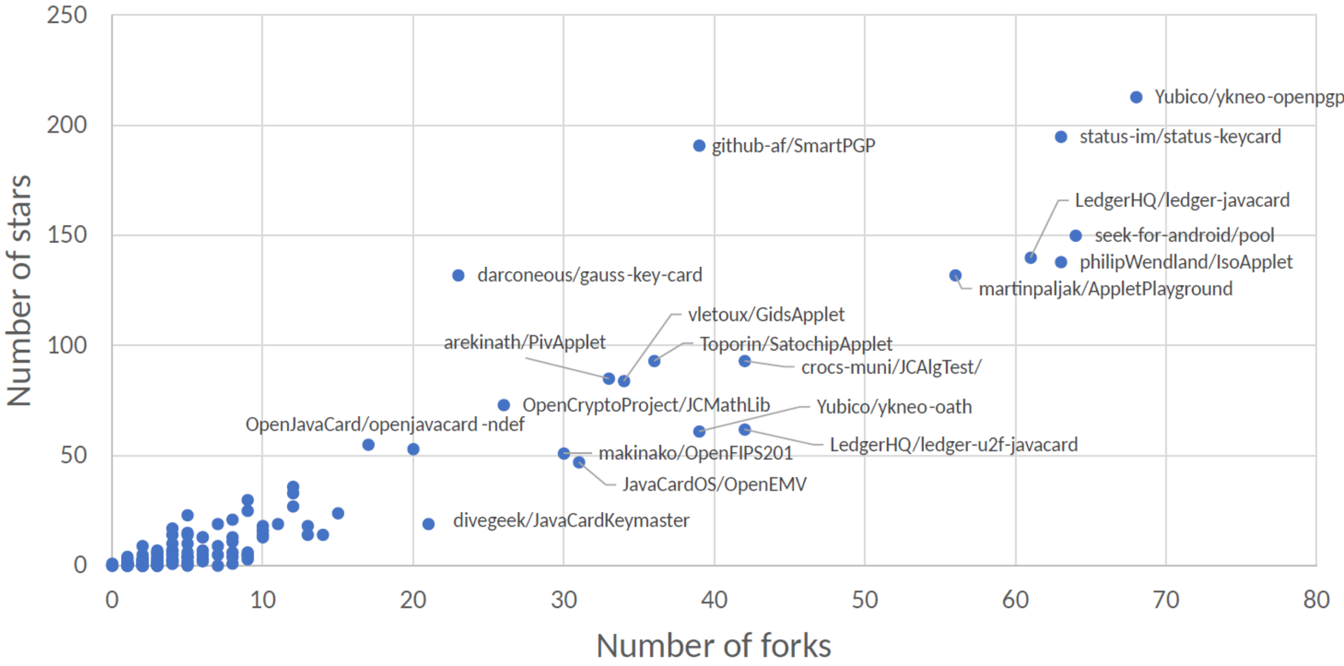



**Fig. 3.** Number of open-source projects with at least one commit per month (black line) or per year (red line) respectively. The year 2023 is only till end of June.




- Is open-source ecosystem representative of the whole domain?
  - Likely two orders of magnitude more developers in **non**-open source domain
  - Proprietary applets with access to proprietary API may be different

# INTERESTING REAL-WORLD EXAMPLES

# FIDO2 tokens – current state

- FIDO alliance of major companies
- Original U2F protocol extended and moved und
  - – U2F → FIDO2 → WebAuthn
  - – https://www.w3.org/TR/webauthn/
- Large selection of tokens now available
  - – including open-hardware like SoloKey
- Android and iOS added systematic support for FIDO U2F since 2019
  - – Mobile phone acts as FIDO2 token, secure enclave used for storage and exec

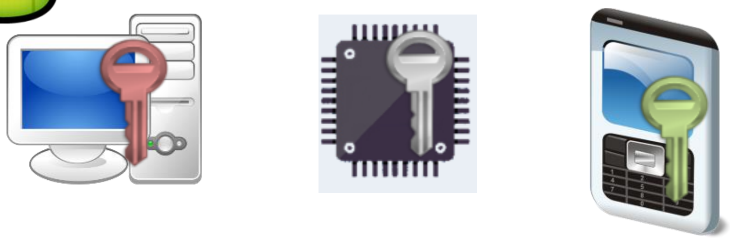
Usable also for authentization and decryption
(more people, threshold k-of-n)

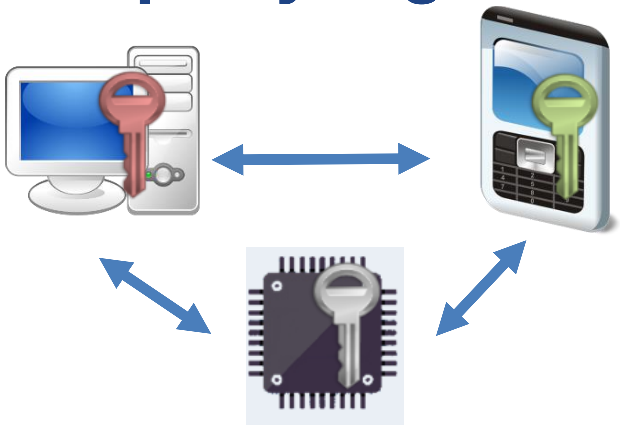


## Multiparty signature

## Single signature

## Multiple signatures



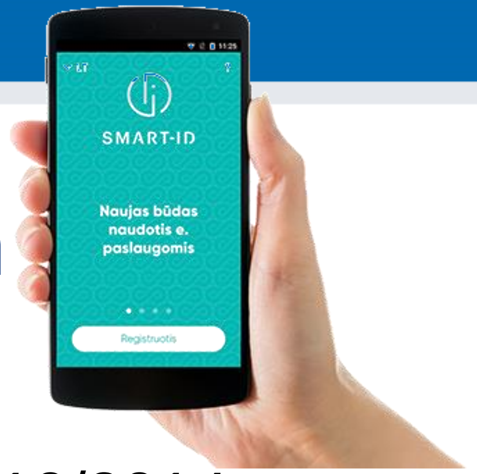


Signature

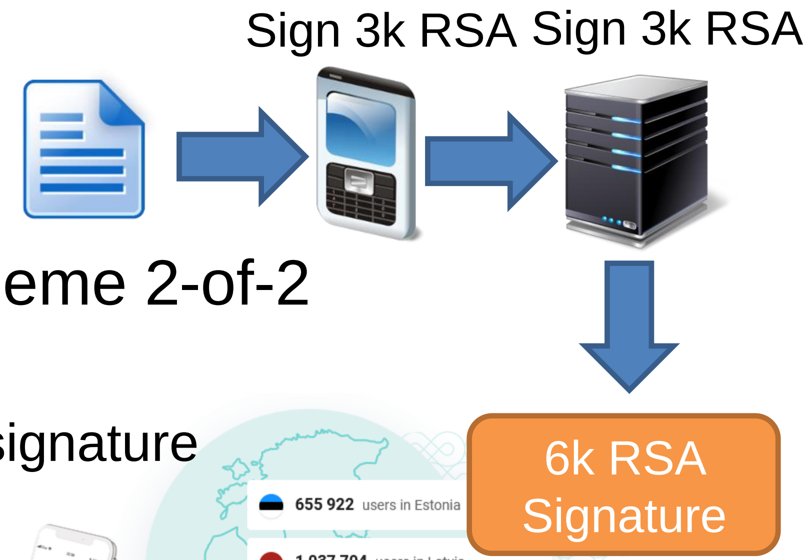
Signature | Signature | Signature

Signature

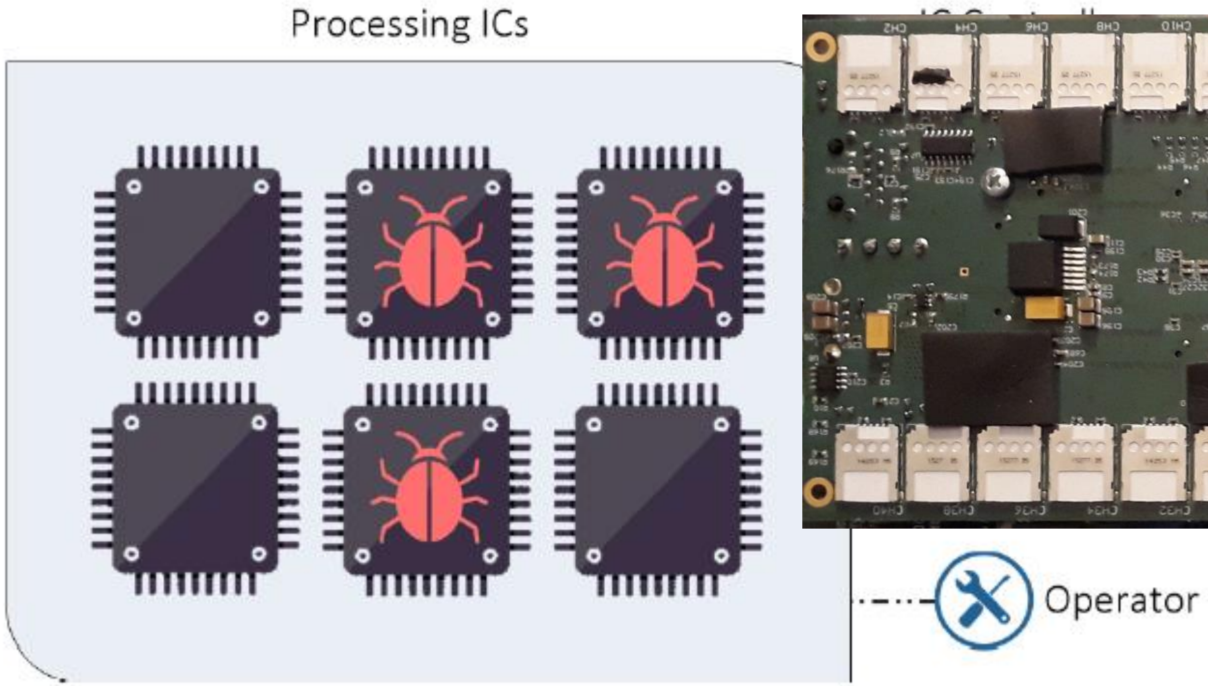
# Real-world example: Smart-ID signature system

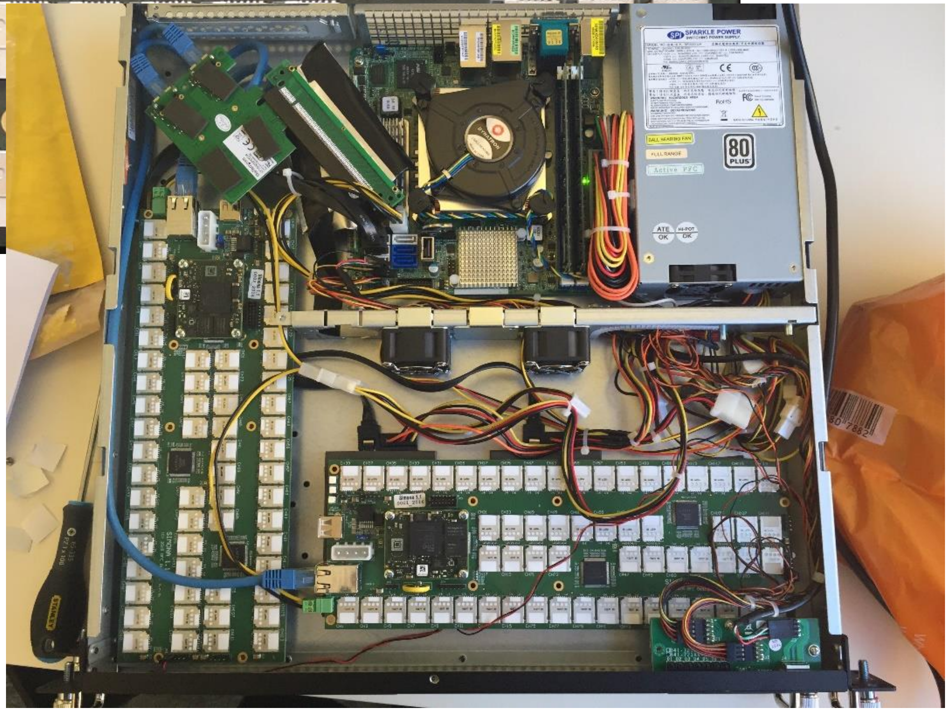



- Banks in Baltic states, >3M active users
- Qualified Signature Creation Device (QSCD) per Regulation No 910/2014
- Collaborative computation of signature using:
  1. User's mobile device (3072b RSA)
  2. Smart-ID service provider (3072b RSA)
- Two-party RSA signatures, multiparty signature scheme 2-of-2
  – Whole signature key never present at a single place
  – Smart-ID service provider cannot alone compute the valid signature
- Resulting signature is 6144b RSA signature
  – => compatible with existing systems



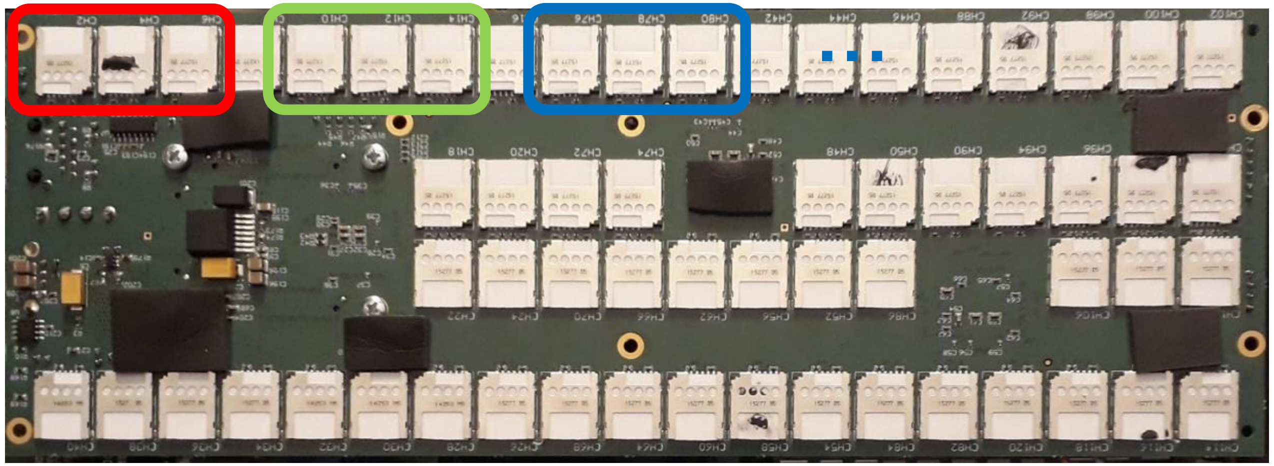

# Myst: secure multiparty signatures



- High-speed, multi-tenant (120 cards)
- Robust against bugs, backdoors

# SmartHSM for multiparty (120 smartcards, 3 cards/quorum)



**120 cards => 40 quorums**
**=> 300+ decryptions / second**
**=> 80+ signatures / second**

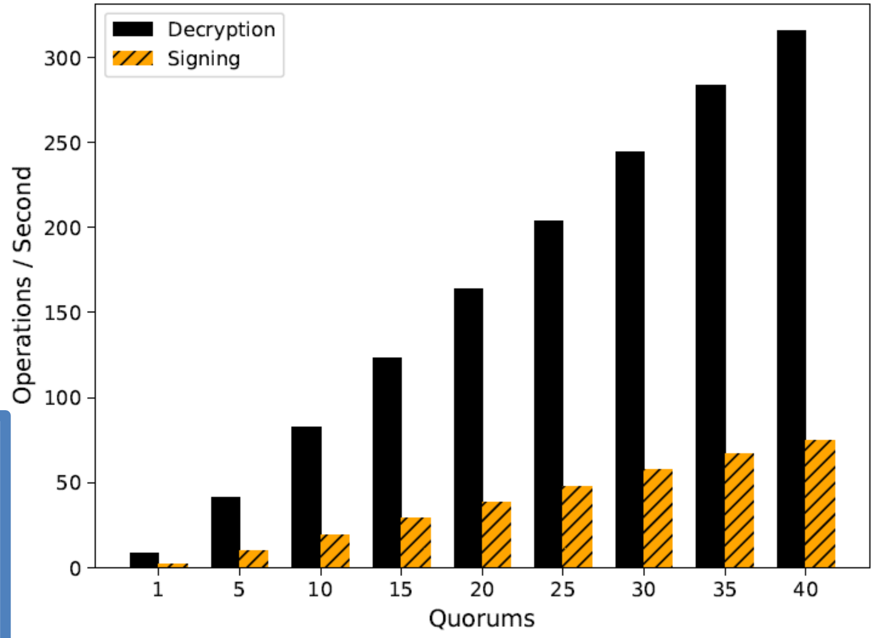



Figure 10: The average system throughput in relation to the number of quorums ($k = 3$) that serve requests simultaneously. The higher is better.

# Cryptocurrency hardware wallets


- Trezor One - first hardware wallet, Czech Republic (2013)
- Seed generated and stored inside, PIN to unlock wallet and sign
- Trezor One cryptographic operations executed on STM32 MCU
  - Side-channel attacks on private key during the use (not really relevant attack)
  - Fault-induction attack during PIN verification (~$200 device to bypass PIN)
- Ledger Nano S wallet – cryptographic smartcard + MCU + display
  - seed stored and cryptographic operations executed inside secure element
  - Side-channel and fault induction attacks very difficult to perform
- But secure element is proprietary – need for trust in its implementation
  - Seed can be stolen / exfiltrated by bug or backdoor


*Images by Trezor and Ledger*

# Open-source wallet with two different secure elements

- Idea: Split trust between multiple proprietary vendors
  - Two secure elements manufactured by different vendors
  - Seed split into three parts (shares): MCU, SE1, SE2
- Decreases required trust into a single SE vendor and its supply chain
- Is the issue completely solved?

# Conclusions

- SC massively deployed ($1x10^{10}$/year), mainly w.r.t. security
  - wide range of usage (banking, SIM, access control)
  - secure storage (encryption/signature keys)
    - on-card asymmetric key generation!
  - secure code execution
  - interesting protocols involving smart cards (multiparty signing…)
- Limited memory ($10^2$ kB) and CPU power (8-32b,5-50MHz)
  - Low-cost small computer designed specifically for security
  - crypto operation accelerated by co-processors
- Can still be attacked (lecture of Lukasz Chmielewski)
  - typically need for special knowledge and/or equipment
  - still far more secure than standard PC