This work is distributed under the CC BY-SA license, unless stated otherwise.

Hardware Security

2. Smartcards

Ján Jančár

Summer 2026

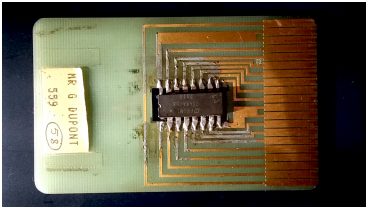
Recap: Secure Elements

- ← Last week: **Secure Elements (SE)**: isolated, tamper-resistant chips with a small Trusted Computing Base (TCB)
- → Today: **evolution**, **hardware**, and **programmability**
  - History, physical properties, communication
  - Global Platform: Managing applets
  - JavaCard: Programming applets
  - Applications: EMV, eID, eMRTD

History

Roland Moreno

- **1974:** Roland Moreno patents the “IC Card”
- **Vision:** A portable chip with memory and “logic to compare a secret code”
- **1980s:** Deployment in French phone cards (Télécarte) and banking
- **Goal:** Replace magnetic stripes which were trivial to clone



https://commons.wikimedia.org/wiki/File:Prototype_moreno.jpg

History

RFID, Magnetic stripes

- **Contactless “barcode”**
  - Fixed identification string (RFID, < 5 cents)
  - Problem: Attacker can eavesdrop and clone the chip
- **Simple memory cards**
  - Magnetic stripe, simple RFID
  - Small write memory (< 1 KB), ~ $0.10
- **Memory cards with PIN protection**
  - Simple protection logic, < 5 KB, < $1
- **Programmable smartcards**
  - Complex cryptographic applets, ~ 100 KB, < $20

The Secret Spy Tech Inside Every Credit Card

? What is the DeliCard using? Try to find out.

NFC Tools, Credit Card Reader
Mifare Classic Tool

MIFARE: Most common “*dumb*” card

⚠ All *kind of* broken

- Developed by NXP
- Over 10 billion chips sold
- **MIFARE Classic**
  - Proprietary CRYPTO1 stream cipher (broken by reverse-engineering)
  - Fixed memory sectors with keys A/B
- **MIFARE Plus**
  - Drop-in replacement for Classic with AES-128
- **MIFARE DESFire** (DES/Fast/Innovative/Reliable/sEcure)
  - Supports 3DES and AES
  - Full file system support
- **MIFARE Ultralight**
  - Cheap, disposable, no encryption (paper tickets)

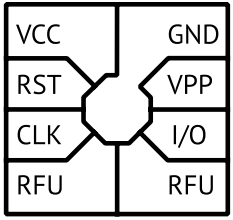
Smartcard hardware

- **Quite powerful for its size**
  - 8–32 bit processor @ 5–50 MHz
  - **EEPROM/Flash:** 32–200+ KB (persistent memory)
  - **RAM:** Very small, usually ≪ 10 KB (volatile memory)
  - **ROM:** For OS and fixed libraries
  - **TRNG:** Truly Random Number Generator. ? What is a TRNG?
  - **Crypto coprocessor:** 3DES, AES, RSA-2048, ECC, etc.
- **TCB size is a critical factor:** “Keep It Simple Stupid” (KISS)

Solution

TRNG generates random numbers from a physical process

Physical interfaces



- **Contact interface (ISO 7816-3)**
  - VCC, GND, I/O, CLK, RST lines
- **Contactless interface**
- **Hybrid vs. Dual interface**
  - Hybrid: Separate chips on a single card
  - Dual: Same chip accessible via both contact and contactless
- **No battery**

Contactless: Proximity vs. Vicinity

⚠ Naming is messy ⚠
Frequency, radio protocol, protocol

- **Prox cards**
  - **Frequency:** 125 kHz (LF)
  - **Range:** Mixed
  - **Speed:** Doesn't matter
  - **Modulation:** Depends
  - **Use:** Ski passes, library cards, physical access, inventory tracking
- **Proximity cards (ISO/IEC 14443)**
  - **Frequency:** 13.56 MHz (HF), NFC
  - **Range:** Very short (~10 cm)
  - **Speed:** High data rates
  - **Modulation:** Type A (100% ASK) & Type B (10% ASK)
  - **Use:** Payment (EMV), Biometric Passports (eMRTD), eID
- **Vicinity cards (ISO/IEC 15693)**
  - **Frequency:** 13.56 MHz (HF)
  - **Range:** Longer range (~1 to 1.5 meters)
  - **Speed:** Lower data rates
  - **Modulation:** Depends
  - **Use:** Ski passes, library cards, physical access, inventory tracking

ISO 7816

Most smartcards *talk* ISO 7816.

- **Part 1–2:** Physical characteristics and contact dimensions
- **Part 3:** Electrical interface and transmission protocols
  - **T=0:** Byte-oriented protocol (older, simpler)
  - **T=1:** Block-oriented protocol (half-duplex, better error handling)
- **Part 4:** Organization, security, and commands (**APDU**)
- **Part 6:** Interindustry data elements for interchange
- **Part 8:** Commands for security operations
- **Part 10:** Synchronous cards

ISO standards are paywalled 💶
Luckily they were leaked

ISO 7816

Part 4: APDU

- The basic communication datagram
- **Command APDU (C-APDU):**
  - Header (4 bytes): **CLA** (Class), **INS** (Instruction), **P1**, **P2** (Parameters)
  - Body: **Lc** (Length of data), Data, **Le** (Expected response length)
- **Response APDU (R-APDU):**
  - Body: Data
  - Trailer (2 bytes): **SW1**, **SW2** (Status Word, e.g., `0x9000` for OK)

Example

- **Scenario:** Selecting the “Hello” Applet
- **Command (Host → Card):**
  - `00 A4 04 00 05 01 02 03 04 05`
  - `00` (CLA), A4 (INS – SELECT), `04` (P1 – by AID), `00` (P2), `05` (Lc), `0102030405` (Data – AID)
- **Response (Card → Host):**
  - `90 00`
  - “Success” (No data returned)
- **Command:**
  - `80 10 00 00 05` (Custom instruction to get “Hello”)
- **Response:**
  - `48 65 6C 6C 6F 90 00` (“Hello” + Success)

ISO 7816

Part 3: ATR

- *Answer To Reset*
- Essentially a card identifier
- Also, information about the communication channel
  - Maximum frequency, some timings, protocol selection (T=0, T=1), ...
- Can be parsed

ISO 7816

Part 4: File system

- Many smartcards (especially older/fixed-function) use a hierarchical file system
- **MF (Master File):** The root of the file system
- **DF (Dedicated File):** Like a directory/folder. Can contain other DFs or EFs
- **EF (Elementary File):** The actual data files
  - **Transparent:** Binary data
  - **Linear Fixed:** Sequence of fixed-length records
  - **Cyclic:** Last-in, first-out (used for logs/history)
- Files are accessed via **File IDs** (2-byte hex codes, e.g., `3F 00` for MF)

Programmable smartcards

- **Why?** Need complex, isolated applications on a single card
- **JavaCard:** Most widely deployed (billions of cards)
- **MULTOS:** Niche
- **Historical:** .NET cards (mostly discontinued), Basic cards

GlobalPlatform

- Specification of API for **card administration**, bridging the OS and applets
- **Over-the-air** updates possible
- **Card Manager (CM):** Special component handling card lifecycle
- **Security Domains (SD)**
  - Logically separated areas with their own authentication keys
  - **ISD (Issuer Security Domain):** The root domain (owns the card)
  - **SSD (Supplementary Security Domain):** For 3rd-party providers
- **Card life cycle:** `OP_READY` → `INITIALIZED` → `SECURED` (normal use) → `CARD_LOCKED` → `TERMINATED` (bricked)

Complicated and full of legacy baggage 🧳

GlobalPlatform

Secure Channel Protocol (SCP)

- **Goal:** Protect communication between the terminal (PC) and the Security Domain
- **Many versions**
  - **SCP02:** Older standard, based on 3DES, deprecated, broken
  - **SCP03:** Modern standard, based on AES
  - **SCP10:** Asymmetric cryptography, based on RSA, broken
  - many more ...

GlobalPlatform

Data Authentication Pattern (DAP)

- **Goal:** Ensure only authorized code is installed on the card
- **Mechanism:** Symmetric key (AES-based MAC) or asymmetric key (RSA)
  - Applet provider digitally signs/MACs the applet file
  - A specific Security Domain on the card contains the corresponding key
  - During `INSTALL` and `LOAD`, the card verifies the signature/MAC (the DAP)
- What is the problem with the symmetric version?
  - **Per-card keys**: Key management, huge amount of keys
  - **Master key**: Card has the key, its leak is catastrophic

JavaCard

- **Maintained by Java Card Forum (since 1997)**
- **Architecture**
  - Java-like application (Applet)
  - Compiled to bytecode, converted to CAP file
  - Executed by **Java Card Virtual Machine (JCVM)**
- **Interoperability:** Apps portable between different vendors (NXP, Infineon, Gemalto…)

JavaCard

Applet: Core methods

- **`install()`:** Called only once during applet installation. Used to create objects and allocate memory (persistent)
- **`select()`:** Called when the terminal selects the applet via AID. Used for session initialization
- **`process()`:** The main entry point for all subsequent APDUs. Parses the command and sends a response
- **`deselect()`:** Called when another applet is selected or the card is reset

Full API docs available.

Example

```
public class HelloApplet extends Applet {
  private byte[] hello = {(byte)'H',(byte)'e',(byte)'l',(byte)'l',(byte)'o'};

  public static void install(byte[] bArray, short bOffset, byte bLength) {
    new HelloApplet().register();
  }

  public void process(APDU apdu) {
    byte[] buffer = apdu.getBuffer();
    if (selectingApplet()) return;
    // TODO: Parse instruction in buffer, here we just return Hello
    Util.arrayCopyNonAtomic(hello, (short)0, buffer, (short)0, (short)5);
    apdu.setOutgoingAndSend((short)0, (short)5);
  }
}
```

JavaCard

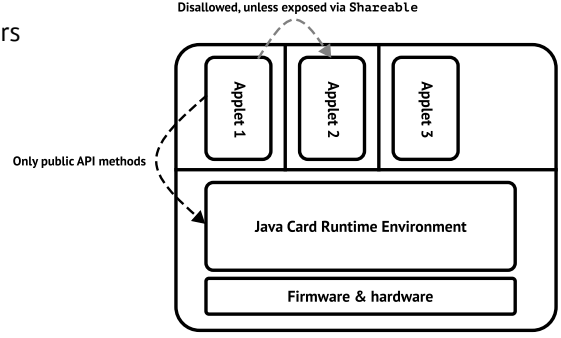
JavaCard vs Regular java

- **No dynamic class loading**
- **No threads and synchronization**
- **No object cloning, finalization**
- **No large primitive data types**
  - No `float`, `double`, `long`, or `char`.
  - Usually **no `int`** (4 bytes) by default; mostly `byte` and `short`.
- **Most standard libraries missing:**
  - No `java.util`, limited `java.lang`
- **Limited garbage collection:** Slow or absent on many cards

https://commons.wikimedia.org/wiki/File:Diet_coke_1.jpg

Benefits

- **Applet isolation:** Firewall prevents applets from accessing each other's data
- **Atomic operations:** Transaction mode (Commit/Rollback) for data integrity if power is lost
- **Rich cryptography API:**
  - Secure access to hardware coprocessors
  - Key objects (protected memory)



JavaCard

Development

1. **Write applet:** Extends `javacard.framework.Applet`
2. **Compile:** Java source to `.class` (restricted Java 1.3+)
3. **Convert:** `.class` to `.cap` (Converted Applet) using the JavaCard Converter
4. **Off-card verification:** Verify good `.cap` for card
5. **Upload & Install:** Using **GlobalPlatform** commands (e.g., `gp --install`)
6. **On-card verification:** Card OS checks CAP file integrity and restrictions
7. **Execution:** Applet runs in isolation on the **JCVM**

? Off-card/On-card verification: What is the threat model?

Solution

Off-card: Developer tests their own applet
On-card: Card verifies applet is well-formed and non-malicious

Context

Big picture

- **User application:** Merchant terminal, Banking GUI, Browser TLS
- **OS smartcard API:** PC/SC (Windows), PC/SC-lite (Linux)
- **Smartcard reader:** Contact (T=0/1) or Contactless (T=CL)
- **Transmission:** APDU packets
- **Card OS:** JavaCard Runtime Environment (JCRE)
- **Card applet:** EMV applet, SIM applet, eMRTD applet

Context

Performance

- Smartcards are **slow** computers (5–50 MHz)
- Humans are **impatient** animals
- **Typical execution times**
  - **AES-128 / SHA-256:** ~1–10 ms (Hardware accelerated)
  - **RSA-2048 Sign:** ~100–500 ms
  - **RSA-2048 Key Generation:** ~2–15 seconds (Heavy primality testing)
  - **ECDSA-256 Sign:** ~50–150 ms
- Benchmarks by the JCAlgTest project

Context

Card Emulation

- **Card Emulation (Contactless)** allows phones to act as smartcards
- **Secure Element mode**
  - A physical SE inside the phone (e.g., Apple Pay, Samsung Pay)
  - High security, hardware-backed
- **Host-based Card Emulation (HCE)**
  - Software-based emulation (Android, e.g. Google Pay)
  - Keys managed in the Android OS or TEE
  - More flexible but potentially less secure than dedicated SE

Applications

A quick overview, each one is worth a lecture of its own

- **EMV:** Payments
- **eID:** Digital signatures and authentication
- **eMRTD:** Biometric passports
- **SIM**: SIM cards
- **Hardware wallets**: Cryptocurrency
- **U2F/FIDO2**: Authenticators

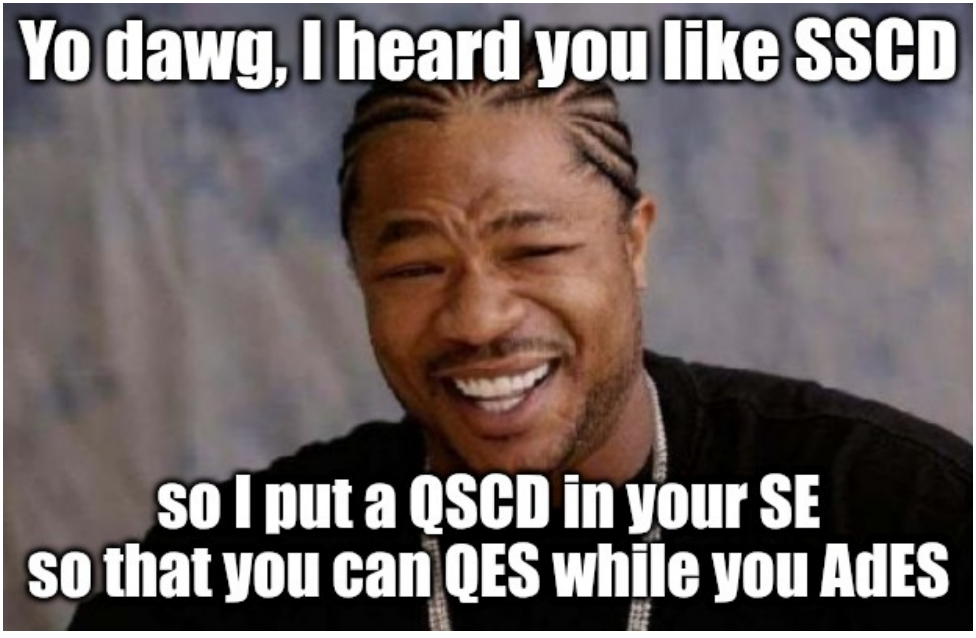
EMV

Europay, Mastercard, Visa

Exposing the flaw in tap to pay

- The global standard for credit/debit smartcards
- **Extremely** complicated, 7 books, > 2000 pages
- **Card Authentication**
  - **SDA (Static Data Authentication):** Card signs static data (cloneable)
  - **DDA (Dynamic Data Authentication):** Card signs a terminal challenge (prevents cloning)
  - **CDA (Combined DDA/Application Cryptogram):** Combines transaction data and DDA to prevent Man-in-the-Middle
- **Cardholder Verification (CVM):** Offline PIN (verified by card), Online PIN (verified by bank), or Signature
- **Transaction Authorization:** Card generates an **ARQC** (Authorization Request Cryptogram) verified online by the issuer bank

eID


Yo dawg, I heard you like SSCD
so I put a QSCD in your SE
so that you can QES while you AdES

eID

- Digital signature and authentication
- **eIDAS Regulation (EU):** Standardizes electronic identification and trust services
- **SSCD (Secure Signature Creation Device):** Hardware that securely manages signature keys
- **QSCD (Qualified Signature Creation Device):** A certified SSCD
- **Qualified Electronic Signatures (QES)**
  - Created by a QSCD
  - Cryptographic proof that the key never left the secure hardware
  - Possesses the **same legal standing as a handwritten signature** in the EU

eID

Only as good as the hardware

- **ROCA Vulnerability (2017)**
  - How do you generate an *n*-bit prime?
  - Infineon's library generated RSA keys with low entropy
  - Allowed practical factorization of the public key to find the private key
  - Impacted **Estonia's eID**, YubiKeys, and millions of TPMs

for small primes x. In this work, we recovered the structure responsible for the properties. All RSA primes (as well as the moduli) generated by the *RSALib* have the following form:

$$p = k * M + (65537^a \bmod M). \tag{1}$$

The integers k, a are unknown, and RSA primes differ only in their values of a and k for keys of the same size. The integer M is known and equal to some primorial $M = P_n\#$ (the product of the first n successive primes $P_n\# = \prod_{i=1}^{n} P_i = 2 * 3 * \cdots * P_n$).

eMRTD

Biometric passports

https://commons.wikimedia.org/wiki/File:Slovak_passport_data_page_sample.jpg

- **ICAO Doc 9303:** Machine Readable Travel Documents (uses ISO 14443, 10–15cm range)
- **Logical Data Structure:** Filesystem
  - **DG1:** Machine Readable Zone (MRZ) data (name, DOB, passport number, etc.)
  - **DG2:** Biometric photo
  - **DG3, DG4:** Fingerprints and iris data
  - **DG15:** Public key for Active Authentication (AA)
  - **EF.SOD:** Document Security Object containing hashes and signatures of all data

eMRTD

Designing a biometric passport

- It contains sensitive data, you will be presenting it at border crossings
- What security properties do you want to ensure?
  - What could go wrong if someone reads my passport from across the room?
  - What if someone copies my passport?
  - What if I can create a fake passport?
  - What if someone wants to track me?

eMRTD

Threat model

All about **physical possession**

- **Without the passport:** Cannot read personal data, eavesdrop, or trace movements
- **With the passport:** Cannot forge data, make digital copies, or read sensitive biometrics (DG3/DG4)
- **Eavesdropping:** Possible at several meters despite short-range RFID

eMRTD

Confidentiality & Access control

- **BAC (Basic Access Control):** Derives symmetric keys from MRZ (DOB, document number, expiry date) using SHA-1
  - What is the problem?
  - **Problem:** MRZ has low entropy and is guessable
  - **Problem:** Once one valid exchange is captured, MRZ can be brute-forced offline
  - **Problem:** Passive listeners knowing MRZ can decrypt all communications
- **PACE (Password Authenticated Connection Establishment):** Modern replacement for BAC
  - Uses nonce encryption and ephemeral elliptic-curve Diffie-Hellman
  - Prevents offline brute-force attacks
  - Only derives session keys if both sides know the password
- **EAC (Extended Access Control):** Protects access to sensitive biometrics
  - **Terminal Authentication (TA):** Terminal must present a certificate chain from Country Verification CA (CVCA)
  - **Chip Authentication (CA):** Next slide

eMRTD

Authenticity & Anti-cloning

- **Passive Authentication (PA):** Document Security Object (EF.SOD) contains
  - Hashes of all Data Groups (DG1, DG2, etc.)
  - Digital signature over hashes, signed by Country Signing CA (CSCA)
  - Proves data was signed by issuing country
  - What is the problem?
  - **Problem:** Does not prevent copying—anyone with access can copy data
- **Active Authentication (AA):** Challenge-response protocol
  - Passport contains private key (EF.AA); public key is in DG15
  - Terminal sends random challenge; passport signs it
  - Proves chip holds the corresponding private key
  - **Problems:** Signature not tied to secure channel (relay attacks). Signs arbitrary data (bad practice)
- **Chip Authentication (CA):** Better alternative to AA
  - Uses static-ephemeral ECDH instead of signing challenges
  - Tied to the secure channel established by PACE/BAC
  - Public key stored in DG14, authenticated via PA

eMRTD

Legacy vs. Modern cryptography

- **Earliest passports (1998 Malaysia):** No cryptography at all
- **First ICAO specs (2003):** Added PA and AA
- **First confidentiality (2005):** BAC introduced
- **Modern improvements:** PACE and EAC (later versions)
- **Backward compatibility issue:** Old passports (5–10 year validity) still in circulation with legacy mechanisms
  - Not all inspection terminals support all protocols
  - Passports may need to support multiple protocols

Gaps in the threat model

- **Tracing with MRZ knowledge:** If border control or a compromised database knows your MRZ, they can track passport movements
  - Specifications prevent tracing without MRZ knowledge
  - Passports always provide random identifiers; publicly accessible files are identical for all citizens of a country
- **Copying with PA-only terminals:** If an inspection system uses only PA (no AA/CA), anyone with a passport scan can make an electronic copy and impersonate
- **Fingerprinting passports:** Same model passports from a country have same unique fingerprint (timing, supported protocols, PACE parameters)

Recommended reading

- For **this** lecture
  - Skim *Smart Card Handbook, Chapter 1*
    - Intro to Smartcards
  - Skim *Smart Card Handbook, Section 13.16.2*
    - JavaCard
- For the **next** lecture
  - None!

Takeaways

- How to communicate with smartcards around you
- What protocols they use
- How they are programmed
- Where they are used
- *Acronyms are everywhere: APDU, SCP, EMV, eMRTD, ...*
- **Next physical tutorial:** Exploring (and programming) smartcards around you

References

- Wolfgang Rankl, Wolfgang Effing: Smartcard Handbook
- Joop van de Pol: The cryptography behind electronic passports
- Tom Chothia and Vitaliy Smirnov: A Traceability Attack Against e-Passports
- Veritasium: The Secret Spy Tech Inside Every Credit Card
- Veritasium: Exposing the flaw in tap to pay
  - Also their references point to interesting papers
- ICAO: ICAO Doc 9303
- Erik Poll: JavaCard
- Erik Poll: Cyber bank robbery
- eIDAS Regulation
- Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec and Vashek Matyas: The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli
- Tom Chotia and Ioana Boureanu have a lot of interesting research on EMV