



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

HardwareSecurity

5. Smartcards (tutorial)

Ján Jančár

Summer 2026

Today

2 parallel activities

- 👤 Breaking and reverse-engineering MIFARE Classic (group activity)
 - [PN532](#) NFC reader + [CP2102](#) USB-to-UART converter
 - Identify card types
 - Attack and get the keys
 - Reverse-engineer the format
- 👤 Analyzing smartcards you brought (me & you)
 - [Proxmark 3](#)
 - Identify card types & properties
- **Demo:** NFC passport/eID sniffing
 - Proxmark 3

Hardware

■ PN532

- Low-cost NFC reader (5£)
- Supports ISO/IEC 14443 Type A & B, FeliCa, RFID
- Can emulate cards (but UID emulation has limits)
- Communication interfaces: UART, SPI, I2C
- 5€

■ CP2102

- USB-to-UART converter
- Converts USB to TTL serial (TX, RX, GND, VCC)
- 3.3V logic level (safe for PN532)
- 3€

■ Proxmark 3

- Much more capable (HF & LF)
- Many, many protocols & tools
- Can emulate cards & sniff
- From 40€ to 400€



<https://anagolife.com/gioi-thieu-ve-module-nfc-pn532-voi-arduino/>



<https://dangerousthings.com/product/proxmark3-easy/>

Group activity

- I have 11 readers
- Work in groups of 3-4
 - Identify
 - Exploit
 - Reverse-engineer

Group activity

Tasks

3 phases

- Identification
 - Get a card
 - Use the `id.py` script to obtain ATQA and SAK
 - Identify the card, using the [docs](#)
- Exploitation (Mifare Classic only)
 - Run `mfoc`
 - Run `mfoc-staticnested` (only if necessary)
 - Save dump
 - Pass the card to the next group
- Reverse-engineering
 - Explore the dumps
 - There will be clues
 - The cards are related
 - Public-transport-like (fictional)

After each phase we will discuss what you learned.

MIFARE



A family of smartcards by NXP

- All kinds of broken, some more than others
 - Reader vs EM-tracing setup
- Classic, Plus, DESFire, Ultralight
- Multiple sizes and variants

- Type A and Type B
- MIFARE is Type A
- Only some MIFARE products support ISO 14443-4 (T=CL)
 - DESFire and Plus
 - See the [docs](#) or this [guide](#)
- Lingo
 - PCD - Reader
 - PICC - Card

- ➔ REQA
 - Request Type A
 - Wakes up cards, reply with ATQA
- ➔ ATQA
 - Answer To Request Type A
 - 2 bytes: UID size, anti-collision support, some proprietary info
- ↔ Anti-collision loop
- ➔ SAK
 - Select Acknowledge
 - After the reader selected the card
 - ISO 14443-4 compatibility, some proprietary info

Specification

- Proprietary cryptography
 - **Crypto-1** (Crapto-1) stream cipher
 - **LFSR** (Linear Feedback Shift Register)
 - 48-bit keys and state
 - 16-bit LFSR for nonces (PRNG)
- 1k, 2k, 4k memory
- Data layout
 - Block: 16 bytes
 - Sector: 3 data blocks + sector trailer
 - Sector trailer
 - ▶ Key A (6 bytes)
 - ▶ Access bits: Which key can do what?
 - ▶ Key B (6 bytes)
- Operations
 - Read, Write
 - Increment, Decrement (Value blocks)

End of memory

Sector	Block	Byte Number within a Block																Description
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
15	3	Key A						Access Bits				Key B						Sector Trailer 15
	2																	Data
	1																	Data
	0																	Data
14	3	Key A						Access Bits				Key B						Sector Trailer 14
	2																	Data
	1																	Data
	0																	Data
:	:																	
:	:																	
:	:																	
1	3	Key A						Access Bits				Key B						Sector Trailer 1
	2																	Data
	1																	Data
	0																	Data
0	3	Key A						Access Bits				Key B						Sector Trailer 0
	2																	Data
	1																	Data
	0	Manufacturer Data																Manufacturer Block

001aan011

Byte Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Description	Key A						Access Bits				Key B (optional)					

001aan013

⚠ Dump will be inverted!

Overview

- 1 Reader: Picks sector and key (A/B), sends this info
 - 2 Card: Generates N_{card} nonce (32-bit) and sends N_{card}
 - 3 Reader: Generates N_{reader} nonce (32-bit) and sends $E_k(N_{card}, N_{reader})$
 - 4 Card: Decrypts and verifies that N_{card} matches and sends $E_k(N_{reader})$
- Authenticated: Both parties know k
 - Encrypted: Keystream synced and used for encryption (XOR)
 - Nonces prevent replay attacks

Card-only

- **Darkside:** mf_cuk
 - **Exploit:** Keystream leakage through errors
 - Card encrypts known error value -> You get the keystream
- **Nested:** mf_oc
 - Requires 1 known key
 - **Exploit:** Keystream leakage through predictable nonces
- **Hardnested:** mf_oc-hardnested
 - Requires 1 known key
 - **Exploit:** Keystream leakage through parity bits
- **Staticnested:** mf_oc-staticnested
 - Requires 1 known key
 - **Exploit:** Keystream leakage through static nonces
- **Backdoor:** mf_oc-backdoor
 - **Exploit:** Known backdoor keys

*You do not need
to understand the
attacks to mount them!*

Reader-only ... Card-reader sniff ...

Group activity

Setup

- Follow the setup instructions
- Linux (Docker) setup preferred
- Linux (native) also works, but takes longer
- Windows (WSL) is complicated
- Mac is unsupported, sorry

Start the setup now.

Group activity

Cards

- A (4 copies)
- B (4 copies)
- C (4 copies)
- D (4 copies)
- EC1
- VS
- MC

Group activity

Identification

- Use the `id.py` script
- Evaluate the ATQA and SAK
- Also the `GetVersion` response
- Also the PPSE response
 - Hint: emvlab [TLV decode](#)
- Once card is identified
 - Mifare Classic: Continue with exploitation
 - Other: Pass it on
- Make notes for each card
 - ? What information can you get from the card?

Group activity

Identification (solution)

- A (4 copies)
 - MIFARE Classic 1k
- B (4 copies)
 - MIFARE Classic 1k
- C (4 copies)
 - MIFARE Classic 1k
- D (4 copies)
 - MIFARE Classic 1k
- EC1
 - MIFARE Classic EV1 1k
 - Taiwanese EasyCard
- VS
 - Visa debit card
- MC
 - Mastercard debit card

Group activity

Exploitation

- Run `mfoc` (Nested attack)
 - If it works, you are done
 - If it shows you logs with the distance not changing, **abort** and try the attack below
 - ? Which keys were the default?
- Run `mfoc-staticnested` (Staticnested attack)
 - Explore the options (you may need `--force-detect-distance`)
 - If it works, you are done
 - ? Which keys were the default?
- Save the dump!
- Pass on the card

Group activity

Exploitation (solution)

- A (4 copies)
 - Dump
- B (4 copies)
 - Dump
- C (4 copies)
 - Dump
- D (4 copies)
 - Dump
- A, B, C, D
 - Same set of keys
 - Always the default one for sector 0
- EC1
 - mfoc and mfoc-staticnested do not work for this one
 - Yet I got the dump on a Proxmark3
 - ▶ It has some reverse-engineered ways of computing the keys
 - Dump

Group activity

Reverse-engineering

Tips

- Look at the hexdump `hexdump -Cv <dump.bin>`
- Find strings `strings <dump.bin>`
- Find timestamps (e.g., unix epoch)
- Find and examine value blocks
 - value, negated value, value, then `00ff00ff`
- Values may be little-endian
- Diff the dumps
 - Examine the changes and the static values
 - What actions could one do with a public transport card?
- A -> B -> C -> D
 - What actions were performed in between the dumps?
- Consider the **HardwareLand**

HardwareLand

Public transport

2026-05

Ticket prices

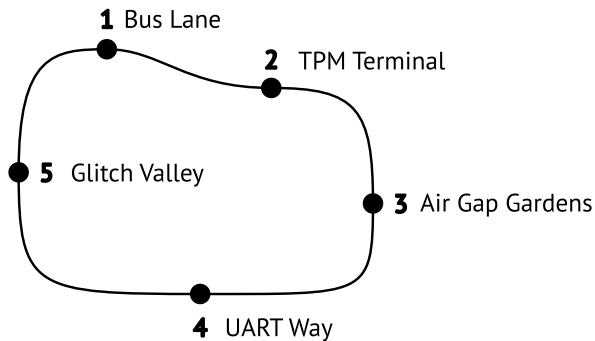
Short-trip 10H

Long-trip 20H

Tariffs

Normal

Half



Reverse-engineering (solution)

Block 0 (040)

0f	42	6f	62	20	74	68	65	20	42	75	69	6c	64	65	72
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Name length Name

Block 2 (060)

1c	0a	14	00	00	00	00	00	00	00	00	00	00	00	00	00	01
YY	MM	DD														Value block pointer
Expiry date																

Group activity

Reverse-engineering (solution)

Sector 2

Block 0 (080)	<u>aa 00 00 00</u> Value	<u>55 ff ff ff</u> $\overline{\text{Value}}$	<u>aa 00 00 00</u> Value	00 ff 00 ff
Block 1 (090)	<u>96 00 00 00</u> Value	<u>69 ff ff ff</u> $\overline{\text{Value}}$	<u>96 00 00 00</u> Value	00 ff 00 ff
Block 2 (0a0)	00 00 00 00	ff ff ff ff	00 00 00 00	00 ff 00 ff

Group activity

Reverse-engineering (solution)

Sectors 3,4,5,6

Ride 11 02 35 92 fc 69 14 00 aa 00 00 05 1e 96 fc 69
 └─┘ └─┘ └──────────┘ └──┘ └──────────┘ └─┘ └──────────┘
 Type From Entry time Cost Balance before To Exit time

Recharge 22 64 00 00 96 00 00 fa 00 00 62 9a fd 69 00 00
 └─┘ └──┘ └──┘ └──┘ └──┘ └──┘ └──┘ └──┘ └──┘ └──┘ └──┘ └──┘ └──┘
 Type Value Balance before Balance after Time

Ride 11 05 af a8 fd 69 14 00 fa 00 00 02 b4 ac fd 69
 └─┘ └─┘ └──┘ └──┘ └──┘ └──┘ └──┘ └──┘ └──┘ └──┘ └──┘
 Type From Entry time Cost Balance To Exit time

Group activity

Reverse-engineering (solution)

(Sector 1)

- Bob the Builder, born 1995-05-21
- Card valid until 2028-10-20
- Normal tariff
- Pointer to current value block (0, 1, 2)

(Sector 2)

- Value blocks track last 3 changes in card value

(Sectors 3, 4, 5, 6)

- Ride (Tag 0x11)
 - Stop entry (1 byte)
 - Entry time (Unix epoch 4 bytes, LE)
 - Cost (2 bytes, LE)
 - Balance before (3 bytes, LE)
 - Stop exit (1 byte)
 - Exit time (Unix epoch 4 bytes, LE)
- Recharge (Tag 0x22)
 - Value (3 bytes, LE)
 - Balance before (3 bytes, LE)
 - Balance after (3 bytes, LE)
 - Timestamp (Unix epoch 4 bytes, LE)

Group activity

Reverse-engineering (solution)

A

- Ride from stop 2 to stop 5, cost 20, before 170, after 150
- Entry 2026-05-07 15:23:01: 1778160181
- Exit 2026-05-07 15:39:42: 1778161182

B

- Recharge 100, before 150, after 250
- Time 2026-05-08 10:10:10: 1778227810

C

- Ride from stop 5 to stop 2, cost 20, before 250, after 230
- Entry 2026-05-08 11:11:11: 1778231471
- Exit 2026-05-08 11:28:20: 1778232500

D

- Ride from stop 2 to stop 1, cost 10, before 230, after 220
- Entry 2026-05-08 15:15:15: 1778332515
- Exit 2026-05-08 15:17:10: 1778332630

NFC passport/eID sniffing demo

- Can sniff a BAC exchange between card and phone (reader)
- MRZ entropy is low, could brute-force the key and get the data

Tools

- [nfc-tools](#) GitHub org
 - mfoc
 - mfcuk
- Proxmark3 & [Iceman fork](#)
- [Farebot](#)

Takeaways

- ❓ What are **you** taking away from this tutorial?
- ❓ What surprised you?

References

- CCC'07: MIFARE Security
- A Practical Attack on the MIFARE Classic
- Dismantling MIFARE Classic
- Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards
- Darkside
- Nested and an Overview
- Hardnested
- Staticnested
- Backdoor

Workshops

- Cracking Mifare Classic on the cheap (workshop)
- Hacking Mifare Classic Cards

EasyCard

- CCC'10: Reverse Engineering a real-world RFID payment system
- slides
- easytool