IERG 4851 – 2026 Fall

香港中文大學
CUHK

Contactless Cards, Inside Out

Yajin ZHOU
https://yajin.org

Today

- By the end you should be able to learn how to hack a card!

A card is a radio

- 13.56 MHz, ISO 14443A — centimetres, not metres
- the card has no battery: the reader's field powers it
- no pairing, no “connect” button
- it answers whoever holds a reader
- so everything before authentication is public



Sony FeliCa reader · Project Kei, CC BY-SA 4.0

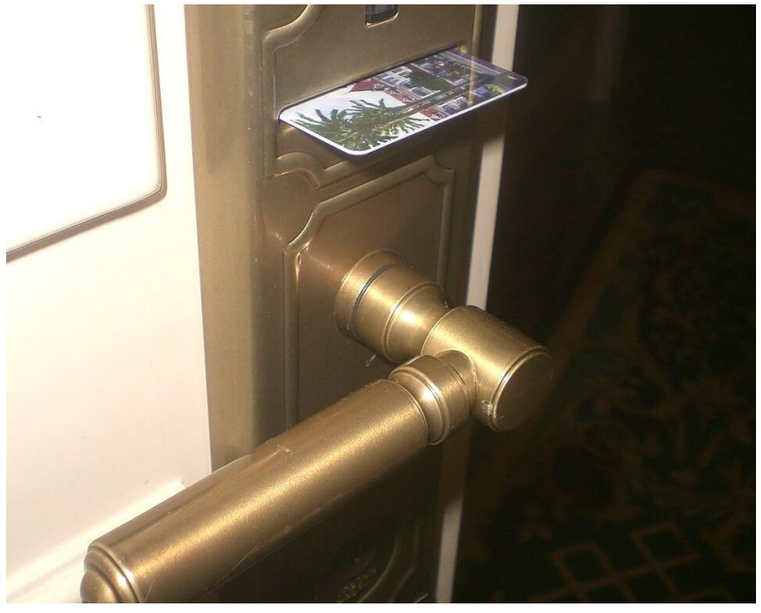
Where you meet this

- transit gates, campus cards, hotel doors, access badges
- one protocol, hundreds of millions of cards



a transit reader

Ubahnverleih, CC0



a hotel room key

Coruberbenda, CC BY-SA 3.0



an access badge

Kymacpherson, CC BY-SA 3.0

What is MIFARE?

- a brand, not a standard: NXP (formerly Philips) started it in 1994
- a family of cards, all speaking ISO 14443A at 13.56 MHz

family	cipher	where you find it
Ultralight	none	event tickets, simple tags
Classic 1K / 4K	Crypto1	transit, campus, hotel locks — the lab's card
Plus	AES	the drop-in replacement for Classic
DESFire	AES	modern transit and ID systems

“1K” is the memory: 1 024 bytes in 16 sectors. “4K” is 40 sectors.

The words, once

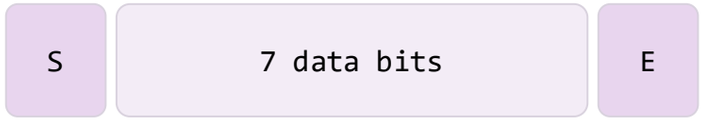
- every abbreviation on the next slides is here — and it stays expanded there too

PCD	the reader	**PICC**	the card (proximity IC card)
REQA	REQuest type A — “any card there?”	**ATQA**	Answer To reQuest type A
ANTICOLL	anticollision — read the UID	**UID**	Unique IDentifier — the serial
BCC	Block Check Character (XOR)	**SAK**	Select AcKnowledge
NVB	Number of Valid Bits	**CRC_A**	CRC type A — 16-bit checksum
HALT	put this card to sleep	**WUPA**	Wake-UP type A — wake a halted card
CL1–CL3	Cascade Level 1–3: the 0x93 / 0x95 / 0x97 rounds	**Crypto1**	the Classic cipher, 48-bit key

Three frame types

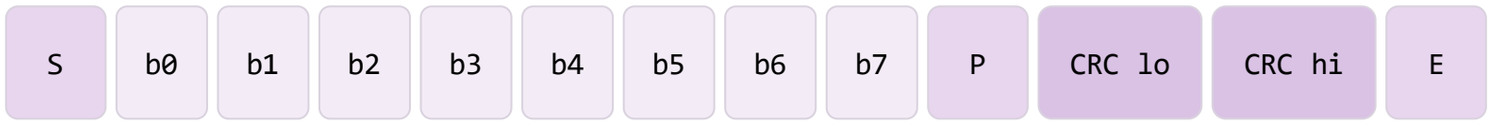
- ISO 14443A defines exactly three; the job decides which one is used

Short frame



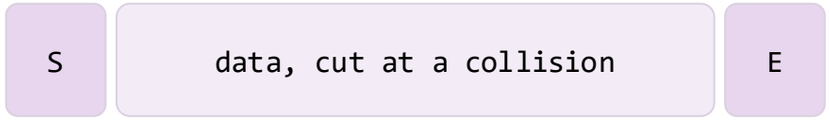
7 data bits, no parity, no CRC — only REQA (0x26) and WUPA (0x52).

Standard frame



start, 8 data bits (one byte), odd parity, then CRC_A, then end — one byte at a time, any length. SELECT, AUTH, READ, WRITE.

Bit-oriented



fixed 4 data bytes + BCC, no CRC; the reader may cut it mid-byte at a collision. ANTICOLL — finding each card's UID.

On the air a byte is 9 bits — 8 data + 1 parity. A frame adds one start bit and one end bit.

What ATQA is for

- ATQA = Answer To reQuest type A — the card's first two bytes
- “yes, a type A card is here” — the reader is not alone in the field
- how long the UID is (single, double, triple), so the reader knows how many cascade levels to run
- whether the card supports the ISO bit-oriented anticollision
- the rest is vendor-specific; some cards use it to pick a proprietary scheme
- our card answers 04 00: single-size UID, standard anticollision
- and it answers before any key — the first thing a card ever reveals

Three messages, byte by byte

```
REQA        26                                  7-bit short frame
  <- ATQA   04 00
ANTICOLL    93 20                               bit-oriented, no CRC
  <-        42 1b 57 05 0b                      UID (4 bytes) + BCC
SELECT      93 70 42 1b 57 05 0b f0 3d           standard frame + CRC_A
  <- SAK    08

BCC = 42 ^ 1b ^ 57 ^ 05 = 0b        CRC_A(93 70 … 0b) = 3df0 -> f0 3d

SELECT is 9 bytes:  93 70  |  42 1b 57 05 0b  |  f0 3d
```

Left: the reader's command. Right: the card's answer. REQA = REQuest type A · ATQA = Answer To reQuest · SAK = Select AcKnowledge

What the UID is

- Unique IDentifier — the card's serial, fixed at the factory
- 4, 7 or 10 bytes: single, double and triple size

```
our card · single size
UID   42 1b 57 05
BCC   0b
→ block 0 holds these five bytes
```

A serial that anyone in range can read is not a secret.

The UID is public

- broadcast before authentication, and printed on the card
- constant for the life of the card
- a fine identifier, a terrible secret
- a deployment that derives its key from these bytes publishes the key:
  - `key = md5(uid)[0:6]`
- a key anyone can compute is not a key

MIFARE Classic 1K: 16 × 4 × 16

sector	block 0	block 1	block 2	block 3
0	UID · BCC · SAK			trailer
1				trailer
2				trailer
3				trailer
4				trailer
5				trailer
6				trailer
7				trailer
8				trailer
9				trailer
10				trailer
11				trailer
12				trailer
13				trailer
14				trailer
15				trailer

1 024 bytes

· 16 sectors, 4 blocks each, 16 bytes a block

· blocks are numbered 0…63 across the card

· block 4·s + j is block j of sector s

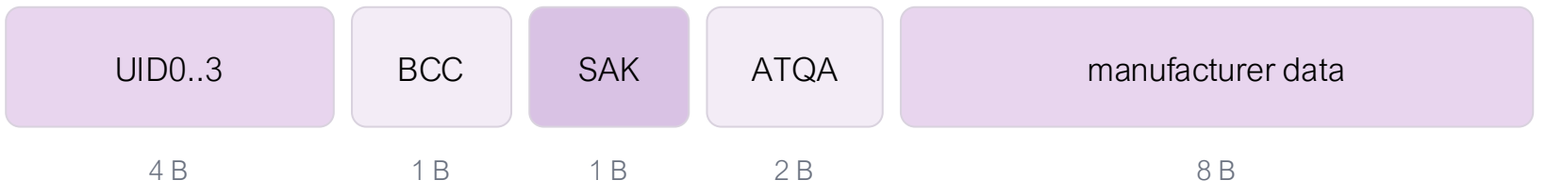
· so block 4 = sector 1, block 0

· block 0: the manufacturer block

· last block of each sector: the trailer

Block 0 is factory-written

- read-only; identical on every card of a model except the serial
- SAK = Select AcKnowledge: 0x08 means MIFARE Classic 1K
- ATQA: the two bytes that answered REQA — UID size and anticollision scheme
- the last 8 bytes are the manufacturer's own
- the card answers REQA, ANTICOLL and SELECT from this identity page — before any key

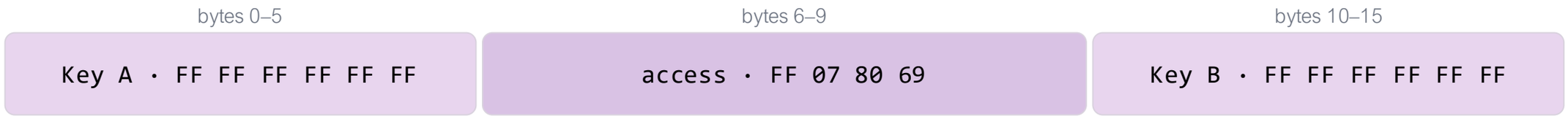


our card: 42 1b 57 05 | 0b | 08 | 04 00 | 49 45 52 47 34 38 35 31

Reading block 0 later shows these same bytes — read-only, and only after authenticating sector 0.

The sector trailer

- 6 bytes Key A · 4 bytes access bits · 6 bytes Key B
- bytes 6–8 hold 12 bits — C1 C2 C3 for each of the four blocks; **byte 9 is user data**
- every bit is written twice — true and complemented — so a corrupted copy is caught
- the factory ships FF FF FF FF FF FF for both keys
- Key A never leaves the card: a trailer read shows six zero bytes there



FF 07 80 69 = every operation on this sector, with Key A or Key B.

Decoding FF 07 80 69

- byte 6 = FF, byte 7 = 07, byte 8 = 80, byte 9 = 69 (user data)
- C1 = 07 high = 0000 · C2 = 80 low = 0000 · C3 = 80 high = 1000
- bit 3 is block 3, so blocks 0–2 are 000 and the trailer is 001

block	C1 C2 C3	what it allows
0	000	read · write · increment · decrement — Key A or Key B
1	000	same
2	000	same
3	001	trailer: Key A write with A · access bits with A · Key B readable with A

Transport configuration: every data block is writable with either key. Key B is readable with Key A here, which is why Key B cannot authenticate.

The access table: data blocks

- for blocks 0–2, three bits choose which key each operation needs

C1C2C3	read	write	increment	decrement / transfer	application
`000`	A|B	A|B	A|B	A|B	transport configuration
`010`	A|B	never	never	never	read/write block
`100`	A|B	B	never	never	read/write block
`110`	A|B	B	B	A|B	value block
`001`	A|B	never	never	A|B	value block, no recharge
`011`	B	B	never	never	read/write block
`101`	B	never	never	never	read/write block
`111`	never	never	never	never	read/write block

A|B means either key; “never” means the operation is refused whatever key you hold.

The access table: the sector trailer

- block 3 has its own table — it governs the keys themselves

C1C2C3	Key A read	Key A write	access read	access write	Key B read	Key B write
000	never	A	A|B	never	A|B	A|B
010	never	never	A|B	never	A|B	never
100	never	B	A|B	never	never	B
110	never	never	A|B	never	never	never
001	never	A	A|B	A|B	A|B	A|B
011	never	B	A|B	B	never	B
101	never	never	A|B	B	never	never
111	never	never	A|B	never	never	never

Transport configuration is 001. Where Key B is readable it cannot authenticate — use Key A.

Datablock example

- our simulated card's sector 1 — the same bytes nfctool prints

block	the sixteen bytes	what they are
4	`66 61 63 74 6f 72 79 20 6b 65 79 00 00 00 00 00`	“factory key”, then padding
5	`49 45 52 47 34 38 35 31 5f 4e 46 43 5f 44 45 46`	“IERG4851_NFC_DEF”
6	`41 55 4c 54 4b 45 59 00 00 00 00 00 00 00 00 00`	“AULTKEY”, then padding
7	`ff ff ff ff ff ff ff 07 80 69 ff ff ff ff ff ff`	Key A · access · Key B

Blocks 4–6 are data: whatever the application writes. Block 7 is the trailer, and its bytes are not free data.

Value blocks: byte by byte

- 16 bytes: value, ~value, value, then addr, ~addr, addr, ~addr
- the card trusts the block only if every copy agrees
- increment and decrement check, add, and rewrite all three

value	~value	value	addr
0A 00 00 00	F5 FF FF FF	0A 00 00 00	08 F7 08 F7

10 credits at block 8. ~10 = F5 FF FF FF. The address repeats as 08 F7 08 F7.

A plain write replaces the bytes without the check — so it needs a stricter access bit.

Inside the card

- no CPU, no operating system, no software — a state machine with hard-wired crypto

antenna + rectifier	the reader's 13.56 MHz field becomes the card's power supply
clock recovery	the carrier is also the clock — no crystal, no battery
command decoder	a fixed set: AUTH, READ, WRITE, INCREMENT, DECREMENT…
Crypto1 engine	a 48-bit LFSR; XORs each bit with the keystream as it arrives
nonce generator	produces the 32-bit nt — a weak PRNG in many real cards
EEPROM + access logic	1 KB, the keys, the access bits, and the rules that enforce them

"Decrypting" is just XOR with a keystream the hardware generates — no code runs.

Fixed logic means the cipher could never be patched: the fix was a new chip (Plus, DESFire).

How does the reader know the key?

- the card holds its sector keys in the trailer — that is the only copy on the card
- the reader was given the same key when the system was deployed
- either one key for every card, or a per-card key computed at the reader:
  - `K_card = f(K_master, UID)`
    - the master stays in the reader; the card never sends its key anywhere
- an attacker with a reader and no key is stuck — unless the key is default, or f() is public

A read, end to end

```
READ      30 04                                    read block 4  (sector 1, block 0)
  <-      66 61 63 74 6f 72 79 20 6b 65 79 00 00 00 00 00   + CRC_A

          16 bytes: "factory key", then five bytes of zero padding
```

One command per block, four to a sector. In the factory configuration the trailer cannot be read at all — and even where it can, Key A comes back as six zeros. 16 bytes + CRC is one standard frame: the 4-byte limit belonged to anticollision.

How Crypto1 fell

- Crypto1: a 48-bit LFSR stream cipher, one key per sector
  - each authentication leaks parity bits — 16 bits of keystream
  - 2008 — Nohl and Plötz reverse the cipher (USENIX Security)
  - darkside: recover one key with no known key, a few thousand tries
  - nested: with one key, every other sector key in seconds
  - tools: mfoc, mfcuk, Proxmark3 — a backpack, not a lab

Your campus card: CU Link

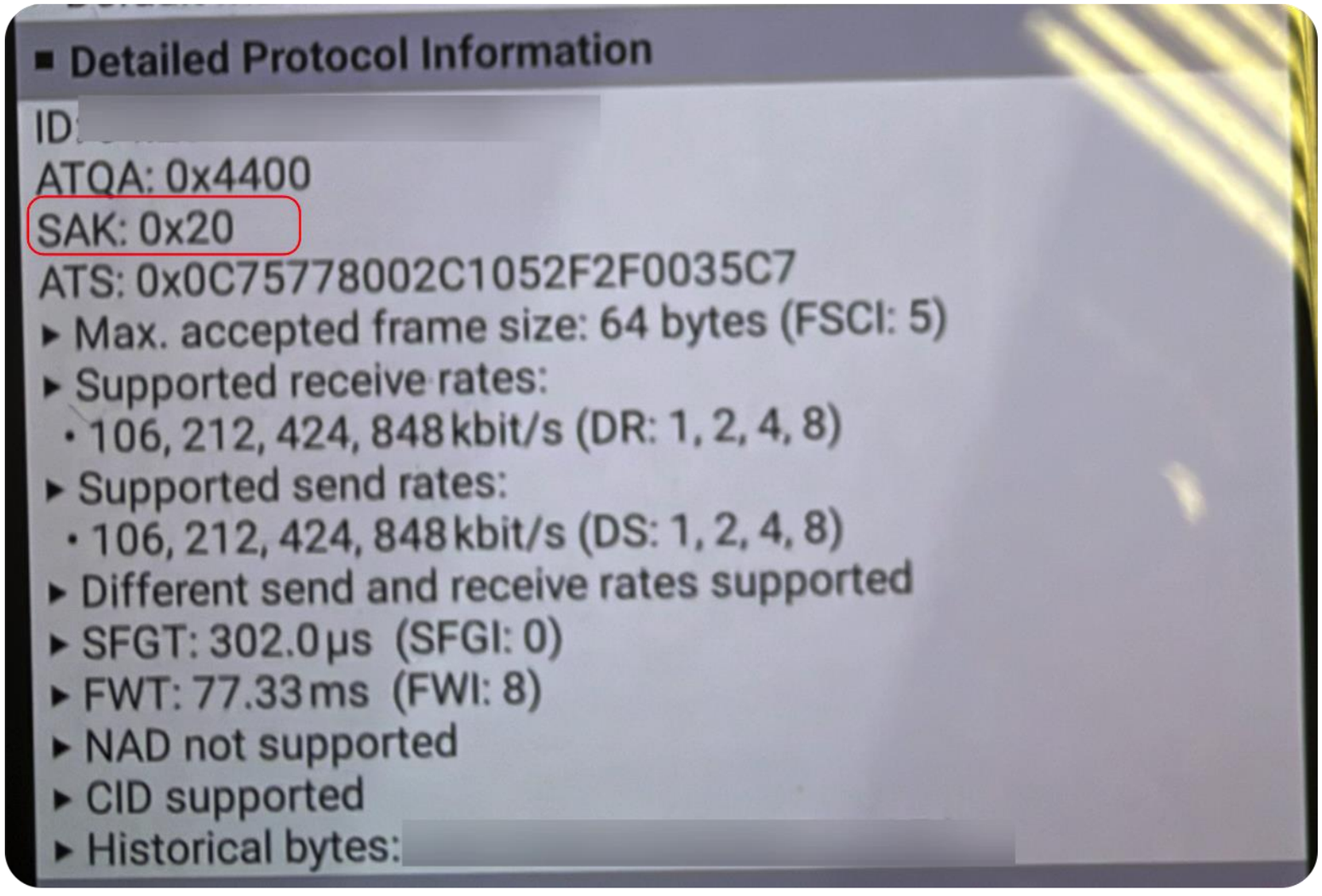
- 1999 — a MULTOS card: a CPU card, contact plates only
- 2009 — contact and contactless in one card
- 2017 — the contactless chip becomes MIFARE Plus
- 2019 — the contact chip is dropped; contactless only
- today: **ISO 14443A, AES, Classic-compatible memory**

1999 MULTOS	2009 contact + contactless	2017 MIFARE Plus	2019 contactless only	2020 Mifare ID on back

Source: culink.cuhk.edu.hk/about-cu-link/history

Your campus card: CU Link

- **MIFARE Plus (SL3)**



Memory card vs CPU card

- a memory card is a state machine with a fixed command set; its keys sit in memory
- a CPU card runs an operating system and answers APDUs: MULTOS, Java Card, DESFire
- the CPU card does the cryptography itself and never hands the key over
- that is why your bank card is a CPU card — and why it costs more than a Classic

memory: MIFARE Classic, Ultralight	CPU: EMV, SIM, MULTOS, DESFire

Classic was cheap, and it spread everywhere — which is why it is still on so many doors.

What actually fails in deployments

- the factory key, left in place — FF FF FF FF FF FF
- a per-card key derived from the public UID
- access bits that let the wrong key do the wrong operation
- each one is a deployment decision, not a cipher break

Demo

- tap the card on the reader; the exchange runs in front of you
- REQA, ANTICOLL, SELECT — then authenticate and read
- each message appears in the direction it travels
- watch what the card says before you type a key

- `http://<lab-machine>:8081/nfc-demo.html`

Demo1

- Default key

Demo2

- Key with uid

- python3 -c "import hashlib; u=bytes.fromhex('f3a79496'); print(hashlib.md5(u).hexdigest()[:12])"
- → 3bb5d6460107

Demo — the card and the reader

- Writable value block

The lab: five cards

- https://yajin.org/teaching/IERG4851/2026/materials/week8-contactless-lab.html

Rules, and the explain round

- attack only the cards assigned to you

Wrap

- a card is a radio with a serial number — everything before auth is public
- the keys live in the trailer; the access bits decide what each may do
- a key derived from public data is not a key
- and the fix is always the same: do not derive a secret from what you publish