Mask off

Analysis of a secure SD card

Nicolas Oberli - Balda

Me?

- Security researcher, Switzerland
- Mostly interested in embedded devices

- BlackAlps organization team

- Hydrabus core developer

Our target

- Flexxon X-mask
- Password protection added to the card
- Once password is set, the card boots into “locked” mode
  - Reading blocks return an empty (ie. all 0xFF) block
- Once unlocked, blocks can be read normally

https://www.flexxon.com/x-mask/

Our

- Fle
- Pa
- On
  “lo
    - block
- On

Hidden Card

Hide crucial data and keep the card invisible to prying eyes. The user will not be able to see the card even if it is inserted into the device and thus, forbids unauthorized access to invade the data.

https://www.flexxon.com/x-mask/

Our

- Fle
- Pas
- On
  “loc
    - R
- On

block

lly

om/x-mask/

Data Encryption

Protect the records with data encryption done using data scrambling and hash algorithms in the firmware and hardware level. The inline proprietary obfuscation tool protects the side-channel attacks and the potential hackers cannot crack the passcode for non-standardization.

Our

- Fle
- Pas
- On
  “loc
  - R
- On

Prot e using
data firmware
and ha bfuscation
tool protec d the potential
hackers ode for non-
standardization.

block

lly

om/x-mask/

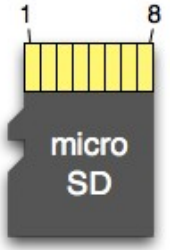
SD card ?

- Secure Digital
  - Available since the 2000s
- Multiple formats (SD, SDHC, SDXC, SDUC)
- Multiple form factors (SD, MiniSD, MicroSD)
- Multiple communication protocols
  - Two (now three) modes: SPI, SD bus, SD express

https://en.wikipedia.org/wiki/SD_card

SD bus protocol ?

- Communication protocol for SD cards
  - Also used in eMMC
- Uses at least 3 signals
  - CLK, CMD, D0
  - Can use 4 data lines (D0-D3)
- Simplified specs available online
  - https://www.sdcard.org/



Pin	SD	SPI
1	DAT2	X
2	CD/DAT3	CS
3	CMD	DI
4	VDD	VDD
5	CLK	SCLK
6	VSS	VSS
7	DAT0	DO
8	DAT1	X

https://www.electroniccircuitsdesign.com/pinout/sd-microsd-card-pinout.html

SD commands

- Commands are sent to the CMD line
  - Card replies with a status after each command
- Command index is the function to call
  - Named in the standard by this index
    - Example: CMD1 => SEND_OP_COND
  - Only one 32-bit argument

Bit position	47	46	[45:40]	[39:8]	[7:1]	0
Width	1	1	6	32	7	1
Description	Start bit	Transmission bit	Command index	Argument	CRC7	End bit

SD block I/O

- Reading and writing data is done with 512 bytes blocks
  - CMD17 – READ_SINGLE_BLOCK
  - CMD24 – WRITE_BLOCK
  - CMD argument is the block address (LBA)
- After command is sent, block data is sent to the data line(s)
  - CRC is appended at the end of the block

SD block I/O

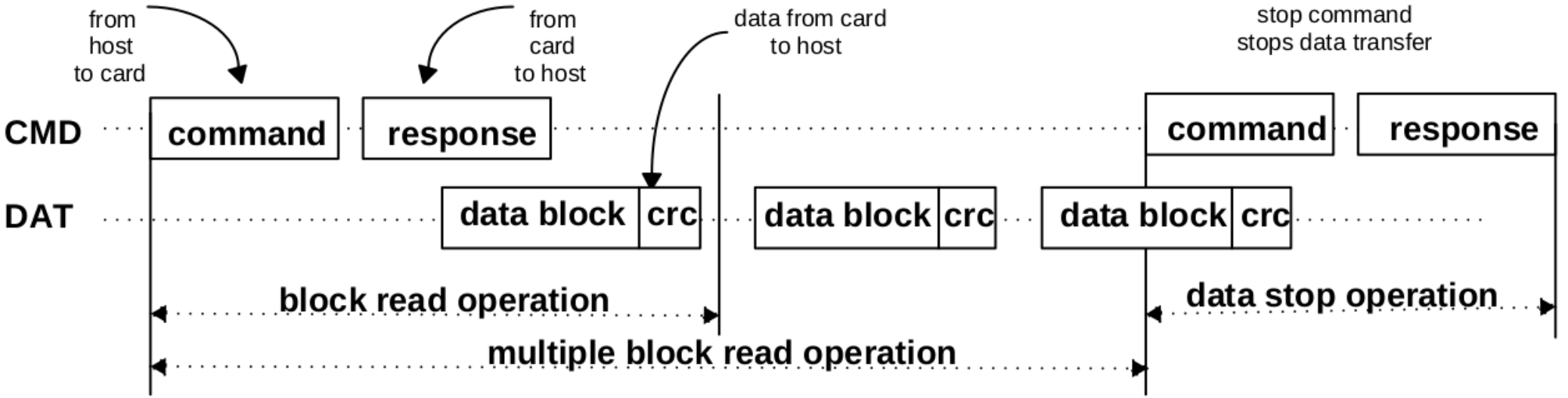


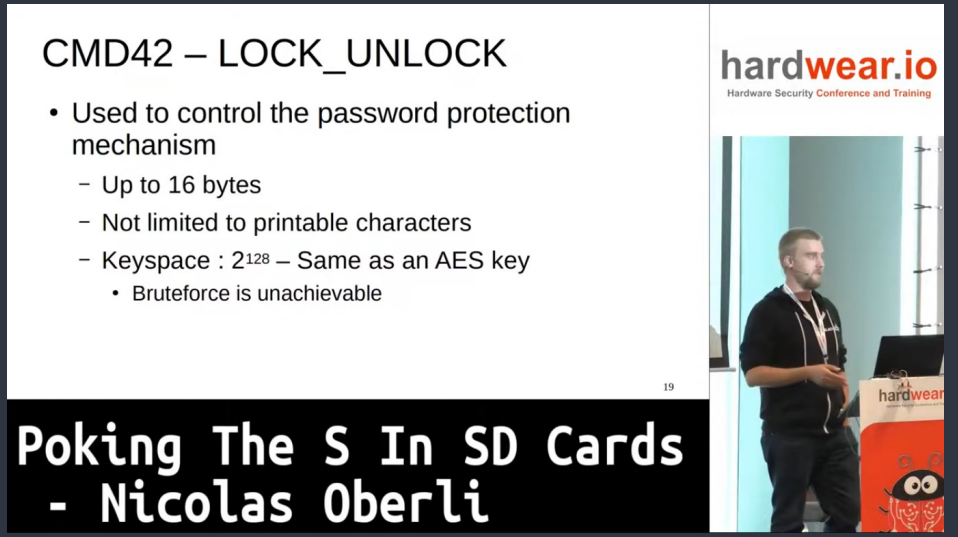
Figure 3-3: (Multiple) Block Read Operation

SD password protection

- Actually, the SD standard already has a builtin password protection
  - Using CMD42 - LOCK_UNLOCK
- Attacks exist
  - HWIO 2019
- Not supported by most Operating Systems

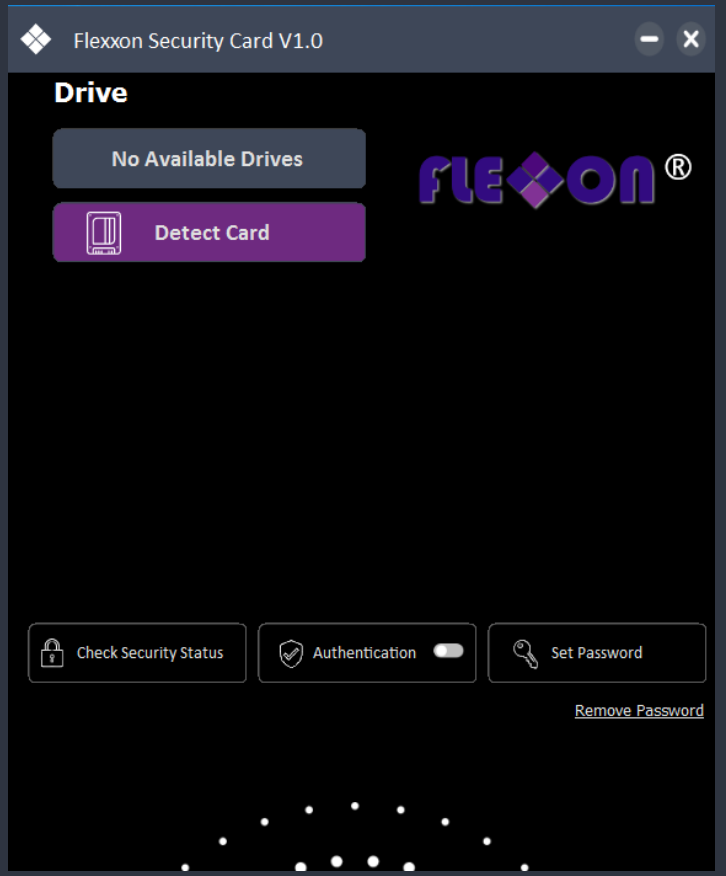
SD password protection

- Actually, the SD standard already has a builtin password protection
  - Using CMD42 - LOCK_UNLOCK
- Attacks exist
  - HWIO 2019
- Not supported by most Operating Systems



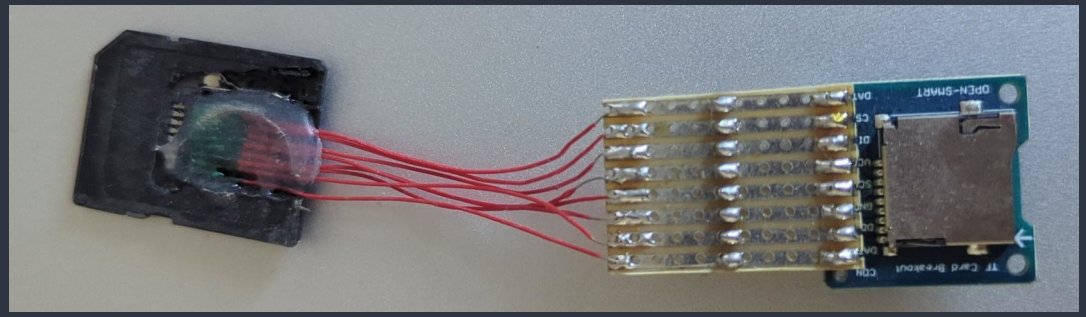
Card usage

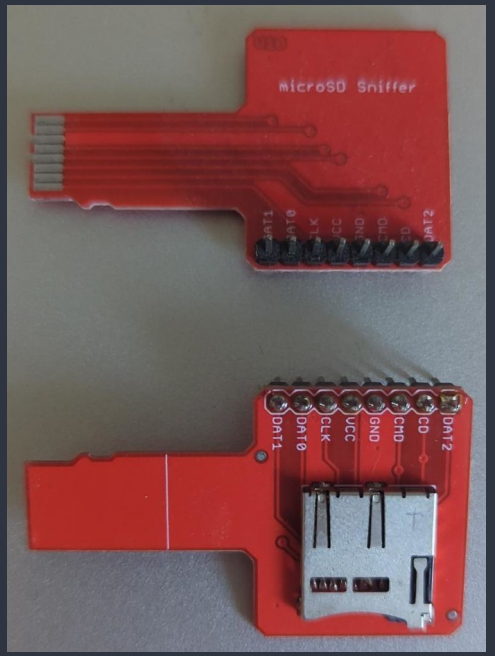
- Card is unlocked by default
- Need a dedicated tool to setup the password protection
- Tool is also needed for unlocking the card
- Windows only :(



Sniffing data

- First attempt: logic analyzer





- Problem: Windows constantly talks to the card
  - Hard to trigger on correct messages
  - Buffer size is also an issue

Sniffing data

- Second attempt: *usbmon*
- USB card reader and sniff data using Wireshark
  - Software runs in a Windows VM
- Can capture long traces and easily filter data

Sniffing data

No.	Time	Source	Destination	Protocol	Length	Info
1041	2025-12-08 21:51:35.7778…	host	7.21.2	USBMS	95	SCSI Write(10) LUN: 0×00 512 bytes (1 blocks) at LBA: 0×00000000
→ 1043	2025-12-08 21:51:35.7799…	host	7.21.2	USB	576	URB_BULK out
1047	2025-12-08 21:51:35.7841…	host	7.21.2	USBMS	95	SCSI Read(10) LUN: 0×00 512 bytes (1 blocks) at LBA: 0×00000005
1050	2025-12-08 21:51:35.7981…	7.21.1	host	USB	576	URB_BULK in
1053	2025-12-08 21:51:35.8013…	host	7.21.2	USBMS	95	SCSI Read(10) LUN: 0×00 512 bytes (1 blocks) at LBA: 0×00000005
1056	2025-12-08 21:51:35.8035…	7.21.1	host	USB	576	URB_BULK in
1065	2025-12-08 21:51:35.8138…	host	7.21.2	USBMS	95	SCSI Write(10) LUN: 0×00 512 bytes (1 blocks) at LBA: 0×00000000
1067	2025-12-08 21:51:35.8159…	host	7.21.2	USB	576	URB_BULK out
1077	2025-12-08 21:51:35.8264…	host	7.21.2	USBMS	95	SCSI Write(10) LUN: 0×00 512 bytes (1 blocks) at LBA: 0×00000000
1079	2025-12-08 21:51:35.8284…	host	7.21.2	USB	576	URB_BULK out
1097	2025-12-08 21:51:35.8471…	host	7.21.2	USBMS	95	SCSI Write(10) LUN: 0×00 512 bytes (1 blocks) at LBA: 0×00000000
1099	2025-12-08 21:51:35.8492…	host	7.21.2	USB	576	URB_BULK out
1109	2025-12-08 21:51:35.8609…	host	7.21.2	USBMS	95	SCSI Write(10) LUN: 0×00 512 bytes (1 blocks) at LBA: 0×00000000
1111	2025-12-08 21:51:35.8629…	host	7.21.2	USB	576	URB_BULK out
1121	2025-12-08 21:51:35.8734…	host	7.21.2	USBMS	95	SCSI Write(10) LUN: 0×00 512 bytes (1 blocks) at LBA: 0×00000000
1123	2025-12-08 21:51:35.8755…	host	7.21.2	USB	576	URB_BULK out
1133	2025-12-08 21:51:35.8859…	host	7.21.2	USBMS	95	SCSI Write(10) LUN: 0×00 512 bytes (1 blocks) at LBA: 0×00000000

```
▸ Frame 1043: Packet, 576 bytes on wire (4608 bits), 576 bytes captured (4608 bits) on interface usbmon7, id 0
▸ USB URB
  Leftover Capture Data […]: 3639363638333031000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
0000  80 90 f9 03 4c 8f ff ff  53 03 02 15 07 00 2d 00   ····L··· S·····-·
0010  57 3a 37 69 00 00 00 00  ca e6 0b 00 8d ff ff ff   W:7i···· ········
0020  00 02 00 00 00 02 00 00  00 00 00 00 00 00 00 00   ········ ········
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0040  36 39 36 36 38 33 30 31  00 00 00 00 00 00 00 00   69668301 ········
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
```

What do we learn ?

- When setting a password:
  - Write block 0 (fixed data)
  - Read block 5 twice
    - Second time, block contains fixed strings and random data
  - Write block 8 (random data)
  - Write block 0 (fixed data)

What do we learn ?

- When unlocking the card:
  - Write block 0 (fixed data)
  - Read block 5 twice
    - Second time, block contains fixed strings and random data
  - Write block **6** (random data)
  - Write block 0 (fixed data)

What do we learn ?

- When unlocking the card:
  - Write block 0 (fixed data) <= Enter command mode
  - Read block 5 twice
    - Second time, block contains fixed strings and random data
  - Write block 6 (random data)
  - Write block 0 (fixed data)

What do we learn ?

- When unlocking the card:
  - Write block 0 (fixed data)
  - Read block 5 twice <= Read challenge
    - Second time, block contains fixed strings and random data
  - Write block 6 (random data)
  - Write block 0 (fixed data)

What do we learn ?

- When unlocking the card:
  - Write block 0 (fixed data)
  - Read block 5 twice
    - Second time, block contains fixed strings and random data
  - Write block 6 (random data) <= Encrypted payload
  - Write block 0 (fixed data)

What do we learn ?

- When unlocking the card:
  - Write block 0 (fixed data)
  - Read block 5 twice
    - Second time, block contains fixed strings and random data
  - Write block 6 (random data)
  - Write block 0 (fixed data) <= Exit command mode

What do we learn ?

- When unlocking the card:
  - Write block 0 (fixed data)
  - Read block 5 twice
    - Second time, block contains fixed strings and random data
  - Write block 6 (random data)
  - Write block 0 (fixed data) <= Exit command mode
- Commands are encoded on the block address!

Encryption scheme ?

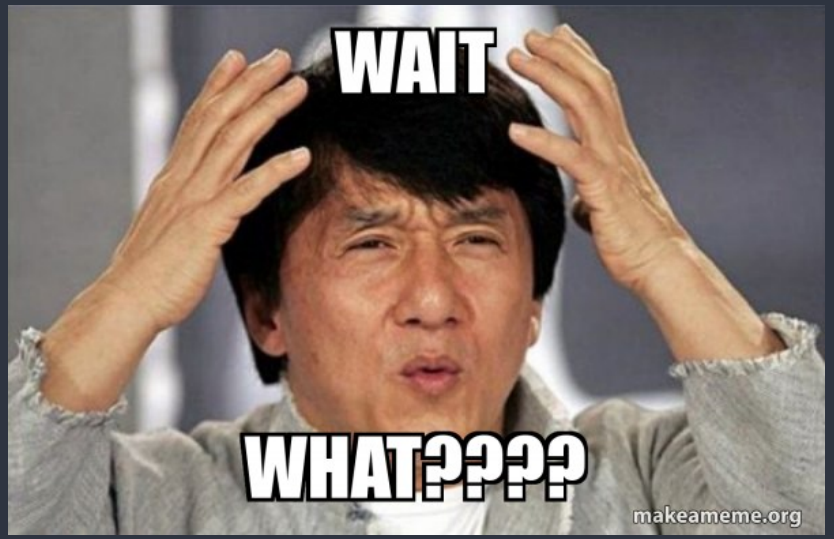
- Some kind of challenge response
  - Reverse the tool using Ghidra
  - Library contains some symbols to help
- Interesting function called between the read and the write
  - Takes the argument to send and the *Challenge* block data as input, looking good!

Encryption scheme

- Start by filling response buffer with random values using a Mersenne Twister
  - Seeded with current time. Never transmitted to card

Encryption scheme

- Start by filling response buffer with random values using a Mersenne Twister
  - Seeded with current time. Never transmitted to card



Encryption “scheme”

- Take values from the *challenge* block

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0x00000000	53	49	30	45	54	31	32	38	38	00	90	03	A2	00	00	00	SI0ET1288.......
0x00000010	00	00	00	00	04	00	03	66	00	00	00	00	00	00	00	00	.......f........
0x00000020	00	00	00	01	45	DE	94	93	76	51	00	00	4E	20	00	00	....E...vQ..N ..
0x00000030	04	32	00	05	00	00	00	0C	2B	00	00	01	44	00	00	00	.2......+...D...
0x00000040	01	00	00	0A	D8	05	05	01	02	00	00	00	00	2D	00	00	.............-..
0x00000050	04	32	00	00	00	00	00	00	00	00	4F	C0	00	00	00	74	.2........O....t
0x00000060	4A	60	46	4C	58	6D	44	10	17	05	00	07	01	65	2B	40	J`FLXmD......e+@
0x00000070	0E	00	32	5B	59	00	00	1C	D7	7F	80	0A	40	00	41	02	..2[Y.......@.A.
0x00000080	85	80	83	00	00	00	00	0F	FF	00	00	00	00	00	00	0A	....
0x00000090	00	00	00	00	00	00	20	00	00	00	00	00	00	00	00	00	....
0x000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0x000000B0	00	00	00	00	00	00	00	00	78	6E	6B	BC	02	BE	76	9E	........xnk...v.
0x000000C0	03	65	4E	29	0C	F7	10	66	13	A6	72	EB	4A	66	24	DF	.eN)...f..r.Jf$.
0x000000D0	3B	E3	5E	48	48	E7	06	FC	09	65	75	93	21	04	01	6C	;.^HH...eu.!..l

Legend:

- header (offset 0, size 3)
- A (offset 60, size 1)
- B (offset 68, size 1)
- Key ((0x44+0xd8)&0xff)+0xb8 (offset 212, size 8)
- Starting position (offset 212, size 1)
- Bit to change (&0x7) (offset 213, size 1)

“Encryption” “scheme” (steg100)

- Encode argument in bits (MSB first)
- Encode key in bits (LSB first)
- For each bit of the key, encode 4 bits of the argument
  - If key bit is 0, invert argument bits

~~Encryption scheme~~ steg100

- Each bit of the encoded argument is “fused” in each byte of the response buffer
  - Starts at *pos* (ie. *key[0]*)
  - Bit to set is defined by *key[1]&0b111*

Example: argument bit is 1, bit to set is 3

0b0100**0**010 ==> 0b0100**1**010

0x42 ==> 0x4A

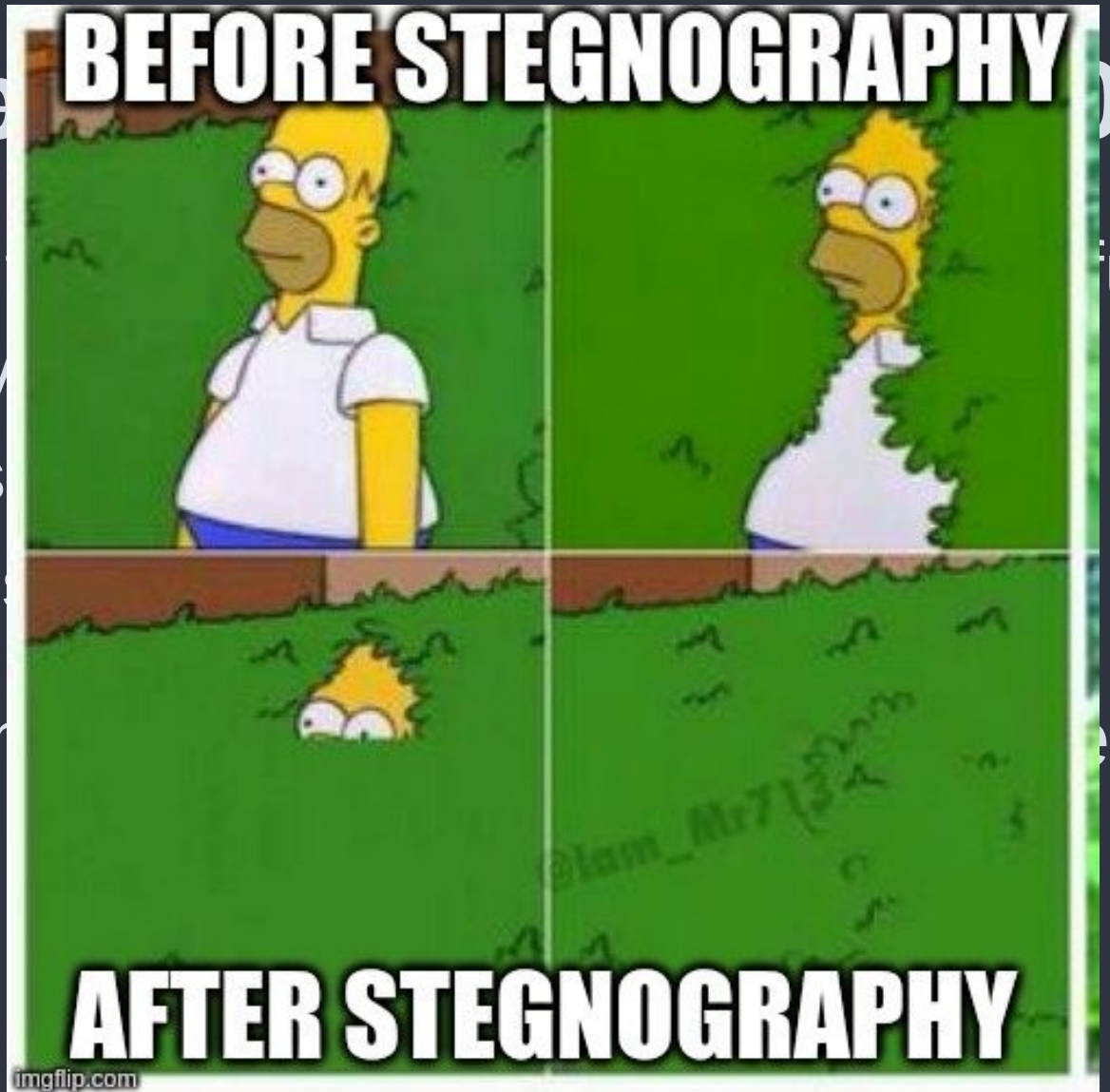
Encryp

- Each bi
  - Starts
  - Bit to

Exam

used" in each by

t is 3



Testing

- Using Hydrabus SDIO mode
  - No OS interaction, can control all transactions on the card
- Wrote an arbitrary test block with known value
- Reimplemented the algorithm in Python
  - Skipped the Mersenne Twister (useless)

```python
# Login method
init_hydrabus()
init_target()
print("Before:")
hexdump.hexdump(read_block(0x2222)[:128])

enter_cmd2()
read_block(5)
P1 = read_block(5)
P2 = encode_response(P1, b'123456')
P2 = cksum(P2)
write_block(6, P2)
while s.send_short(13,raddr) != b'\x00\t\x00\x00':
    pass
leave_cmd2()
print("After:")
hexdump.hexdump(read_block(0x2222)[:128])
```

```
Before:
00000000: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000010: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000020: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000030: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000040: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000050: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000060: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000070: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
After:
00000000: 3D 53 45 43 52 45 54 3D  3D 53 45 43 52 45 54 3D  =SECRET==SECRET=
00000010: 3D 53 45 43 52 45 54 3D  3D 53 45 43 52 45 54 3D  =SECRET==SECRET=
00000020: 3D 53 45 43 52 45 54 3D  3D 53 45 43 52 45 54 3D  =SECRET==SECRET=
00000030: 3D 53 45 43 52 45 54 3D  3D 53 45 43 52 45 54 3D  =SECRET==SECRET=
00000040: 3D 53 45 43 52 45 54 3D  3D 53 45 43 52 45 54 3D  =SECRET==SECRET=
00000050: 3D 53 45 43 52 45 54 3D  3D 53 45 43 52 45 54 3D  =SECRET==SECRET=
00000060: 3D 53 45 43 52 45 54 3D  3D 53 45 43 52 45 54 3D  =SECRET==SECRET=
00000070: 3D 53 45 43 52 45 54 3D  3D 53 45 43 52 45 54 3D  =SECRET==SECRET=
```

Exploring features

- Now possible to call any card “commands” and control argument
- Command 9 looked interesting
  - Allows to set the block address from where the “mask” feature is enabled
  - Most probably used to set a “public” and “private” partition

Authentication bypass

- Command 9 is accepted even if the card is in locked mode
- Setting to 0xffffffff allows to disable the protection and read all data **without knowing the password**

Linux tool

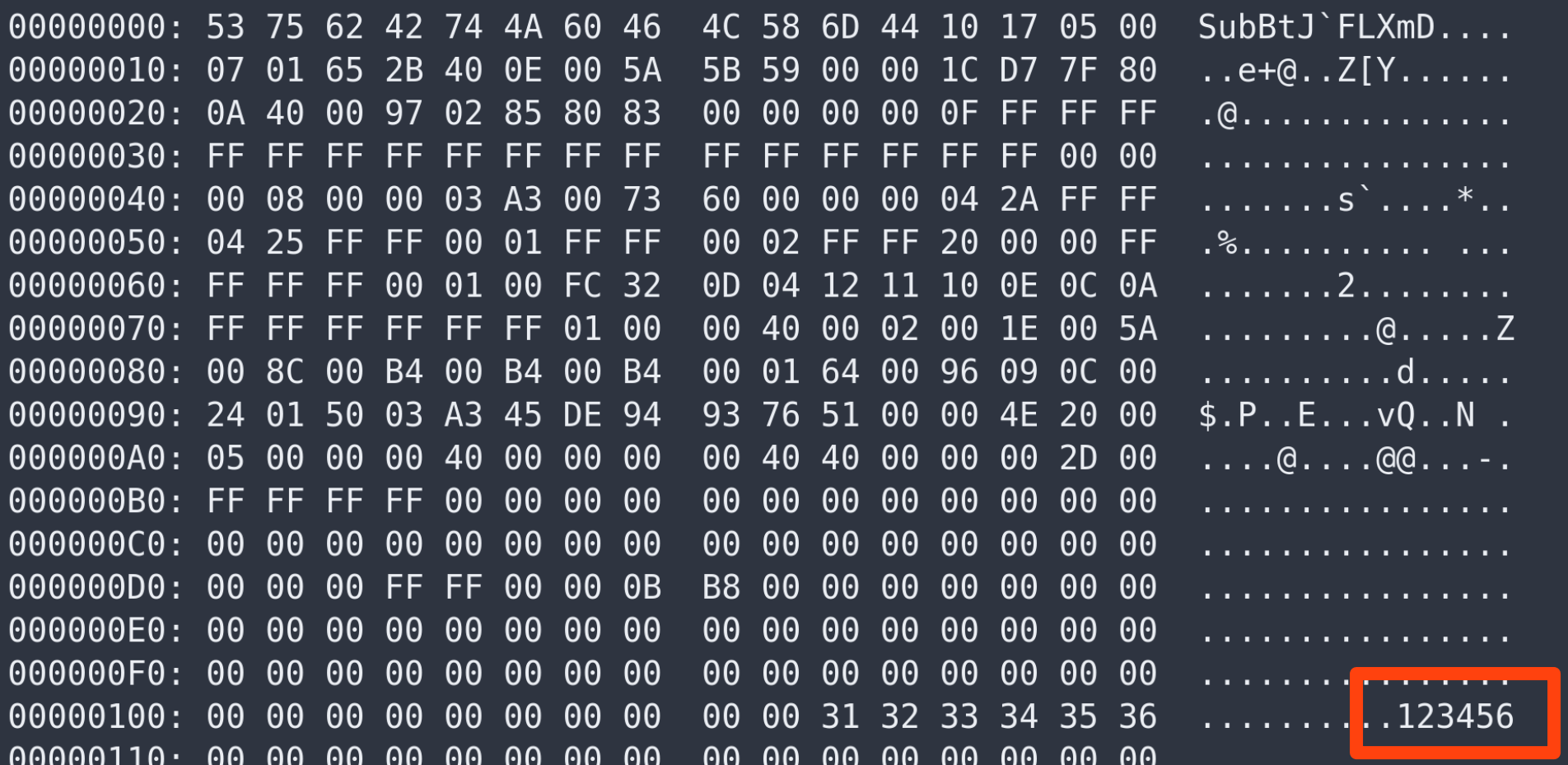
- Since there is no Linux support, vibe-coded a tool to interact with X-mask cards
- Too lazy to support user password, using the vulnerability to lock and unlock the card
- https://github.com/Baldanos/flexxon-tool

Demo !

More ?

- Looking for other valid command IDs
- Enable command mode, then read all sectors sequentially
- Mostly empty blocks, until

```
00000000: 53 75 62 42 74 4A 60 46  4C 58 6D 44 10 17 05 00  SubBtJ`FLXmD....
00000010: 07 01 65 2B 40 0E 00 5A  5B 59 00 00 1C D7 7F 80  ..e+@..Z[Y......
00000020: 0A 40 00 97 02 85 80 83  00 00 00 00 0F FF FF FF  .@..............
00000030: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF 00 00  ................
00000040: 00 08 00 00 03 A3 00 73  60 00 00 00 04 2A FF FF  .......s`....*..
00000050: 04 25 FF FF 00 01 FF FF  00 02 FF FF 20 00 00 FF  .%.......... ...
00000060: FF FF FF 00 01 00 FC 32  0D 04 12 11 10 0E 0C 0A  .......2........
00000070: FF FF FF FF FF FF 01 00  00 40 00 02 00 1E 00 5A  .........@.....Z
00000080: 00 8C 00 B4 00 B4 00 B4  00 01 64 00 96 09 0C 00  ..........d.....
00000090: 24 01 50 03 A3 45 DE 94  93 76 51 00 00 4E 20 00  $.P..E...vQ..N .
000000A0: 05 00 00 00 40 00 00 00  00 40 40 00 00 00 2D 00  ....@....@@...-.
000000B0: FF FF FF FF 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000D0: 00 00 00 FF FF 00 00 0B  B8 00 00 00 00 00 00 00  ................
000000E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000100: 00 00 00 00 00 00 00 00  00 00 31 32 33 34 35 36  ..........123456
00000110: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000120: 00 00 00 00 00 00 00 00  00 00 FF FF FF FF FF 04  ................
```


00000000: 53 75 62 42 74 4A 60 46 4C 58 6D 44 10 17 05 00 SubBtJ`FLXmD....
00000010: 07 01 65 2B 40 0E 00 5A 5B 59 00 00 1C D7 7F 80 ..e+@..Z[Y......
00000020: 0A 40 00 97 02 85 80 83 00 00 00 00 0F FF FF FF .@..............
00000030: FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00 00 ....
00000040: 00 08 00 00 03 A3 00 73 60 00 00 00 04 2A FF FF ....s`....*..
00000050: 04 25 FF FF 00 01 FF FF 00 02 FF FF 20 00 00 FF .%.......... ...
00000060: FF FF FF 00 01 00 FC 32 0D 04 12 11 10 0E 0C 0A ....2........
00000070: FF FF FF FF FF FF 01 00 00 40 00 02 00 1E 00 5A ....@.....Z
00000080: 00 8C 00 B4 00 B4 00 B4 00 01 64 00 96 09 0C 00 ....d.....
00000090: 24 01 50 03 A3 45 DE 94 93 76 51 00 00 4E 20 00 $.P..E...vQ..N .
000000A0: 05 00 00 00 40 00 00 00 00 40 40 00 00 00 2D 00 ....@....@@...-.
000000B0: FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000D0: 00 00 00 FF FF 00 00 0B B8 00 00 00 00 00 00 00 ....
000000E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000100: 00 00 00 00 00 00 00 00 00 00 31 32 33 34 35 36 123456
00000110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000120: 00 00 00 00 00 00 00 00 00 00 FF FF FF FF FF 04 ....

Backdoor?

- Reading block 0x6980 in command mode shows the password in cleartext
- Added to the tool

Demo !

Going PRO

- In September 2025, Flexxon launched the new X-mask PRO

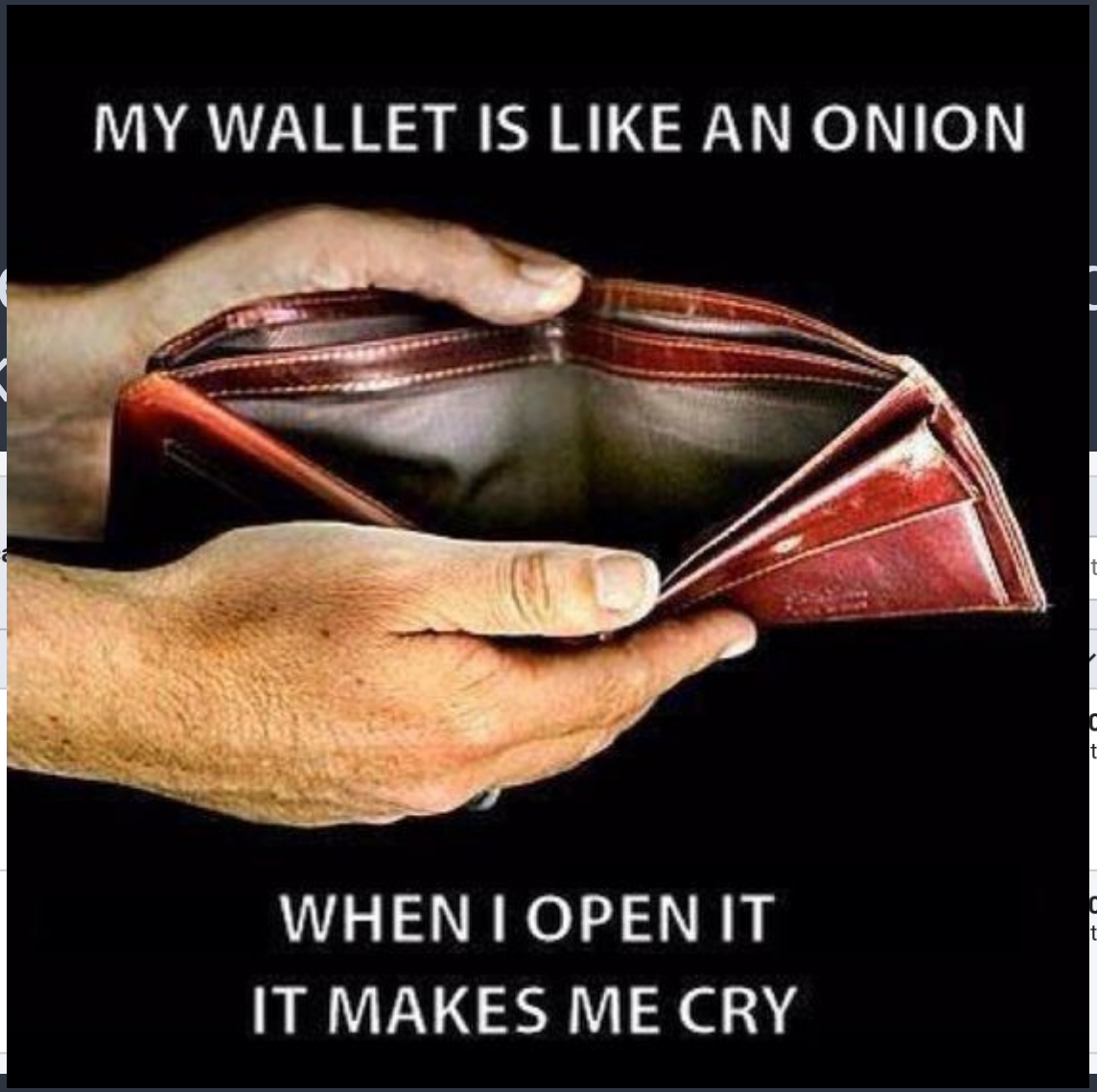
Going PRO

- In September 2025, Flexxon launched the new X-mask PRO

Référence fabricant	Quantité disponible	Prix	Série
FDMM128GBG-XM00 X-MASK PRO MICROSD 128GB 3D TLC *Flexxon Pte Ltd*	**5** En stock	1 : **Fr. 313.40000** Plateau	X-Mask Pro
FDMM064GBG-XM00 X-MASK PRO MICROSD 64GB 3D TLC G *Flexxon Pte Ltd*	**0** En stock **Vérifier le délai d'approvisionnement**	1 : **Fr. 160.34000** Plateau	X-Mask Pro

Going

- In Sept d the new X-mask



Référence fabrica	té	Série
	000 teau	X-Mask Pro
	000 teau	X-Mask Pro

Going PRO

```
diff --git a/README.md b/README.md
index 2a6598f..bfb6829 100644
--- a/README.md
+++ b/README.md
@@ -1,6 +1,7 @@
 # Linux Flexxon tool

 This tool allows you to use Flexxon [X-mask](https://www.flexxon.com/x-mask/)
+and [X-mask PRO](https://www.flexxon.com/x-mask-pro-microsd-card/)
 on Linux systems.
```

Going PRO

```
diff --git a/README.md b/README.md
index 2a6598f..bfb6829 100644
--- a/README.md
+++ b/README.md
@@ -1,6 +1,7 @@
 # Linux Flexxon tool

 This tool allows you to use Flexxon [X-mask](https://www.flexxon.com/x-mask/)
+and [X-mask PRO](https://www.flexxon.com/x-mask-pro-microsd-card/)
 on Linux systems.
```

- Password leak doesn’t work on PRO :(

Responsible disclosure

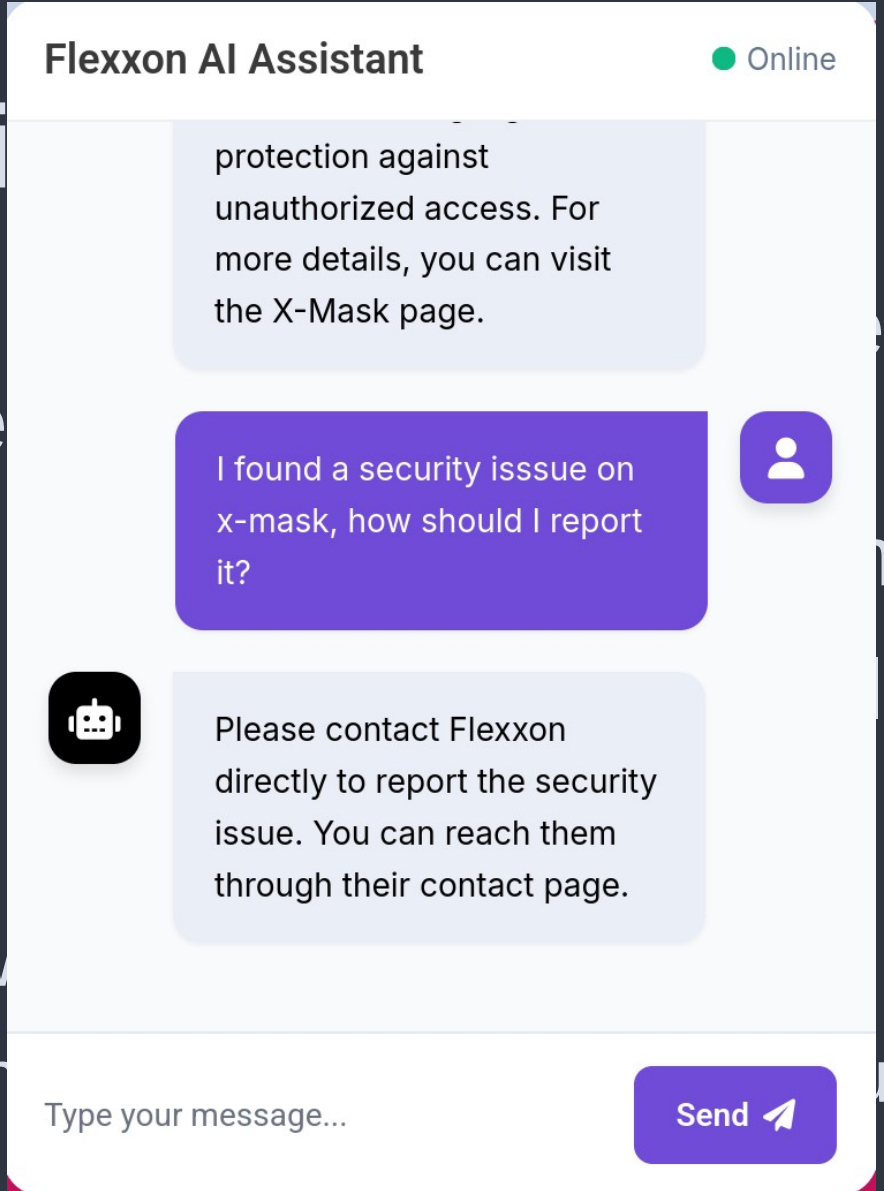
- 10.12.2025: First contact using the “contact us” form on their website
- 17.12.2025: Second contact by email
- 07.02.2026: Third contact by email

- No replies whatsoever
  - not sure they could do anything about it anyway

Responsi

- 10.12.2025 “contact us” form on the
- 17.12.2025 ail
- 07.02.2026

- No replies v
  - not sure th ut it anyway



Conclusions

- Password protection fully bypassed
  - Encryption has no impact
- As a company, please create a security contact
  - Worst case, make sure your sales forward messages to the engineering teams
- Full disclosure is still an option

Thank you !

Masks off: analysis of a secure SD card

Nicolas Oberli
@baldanos
https://balda.ch

BlackAlps CFP is open !
https://blackalps.ch/

Backup – About encryption

- Flash memory has a limited number of erase cycles
  - Pages have to be erased before being written back
- In order to enhance the number of write cycles, block data is randomized
- This is called wear leveling

Backup – About encryption

- AES is often used to perform wear leveling
  - And acts as a good marketing feature
  - “Military grade”, looks cool on datasheets ;)
- Used at the controller level, transparently applied
  - Any logic bug in the controller will still decrypt the data