LEDGER DONJON

Triple Exploit Chain with Laser Fault Injection on a Secure Element

Olivier Hériveaux

Hardwear.io 2023

OUTLINE

CONTEXT

Security assessment of Coldcard Mk3 hardware wallets.
Securely stores user's Bitcoin private seed.

Secure memory ATECC.
Unlock with PIN code.

Seed split in two shares:

- First share in the MCU (STM32L496),
- Second share in the SE (ATECC).

Both circuits must be attacked.

2019 ATECC508A: Single fault attack vulnerability.
Not recommended for new designs

2021 ATECC608A: Double fault attack vulnerability.
JIL High rating[1]
Not recommended for new designs

2022 ATECC608B: Multiple fault attack vulnerabilities.
JIL High rating[1]
In production

[1]Stated by Microchip in their product details.

2019 ATECC508A: Single fault attack vulnerability.
Not recommended for new designs

2021 ATECC608A: Double fault attack vulnerability.
JIL High rating[1]
Not recommended for new designs

2022 ATECC608B: Multiple fault attack vulnerabilities.
JIL High rating[1]
In production

[1] Stated by Microchip in their product details.

SETUP / HIGH-END EQUIPMENT

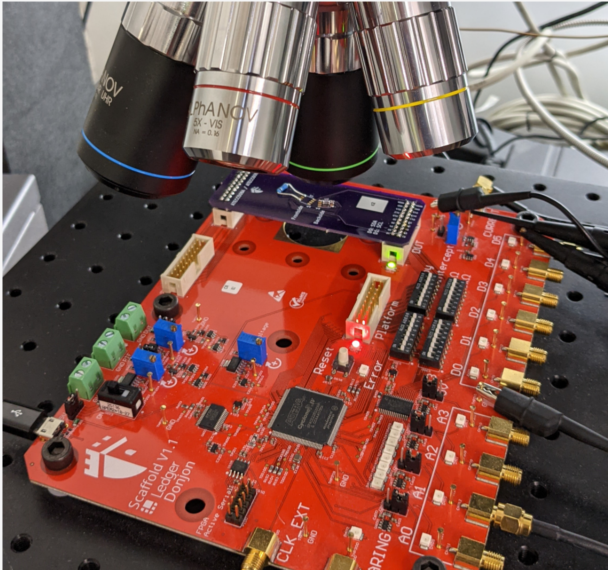
Backside access, no silicon thinning.

Scaffold[2] board for communication, laser triggering and power trace monitoring.

IR camera and microscope.

AlphaNov PDM 2+ IR laser source.

[2] https://github.com/Ledger-Donjon/scaffold



Attack recently reproduced with our low-cost test bench.
No microscope.
No IR camera.

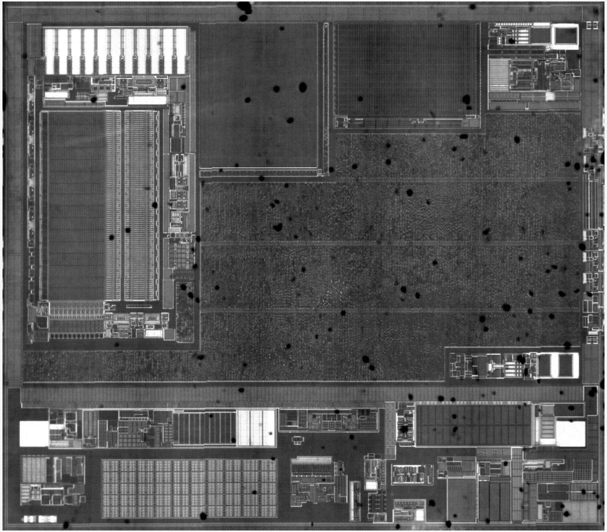
Lower success rate.

Credits: Michaël Mouchous

https://blog.ledger.com/laser-bench-low-price/

All circuit revisions are based on the **same silicon hardware**

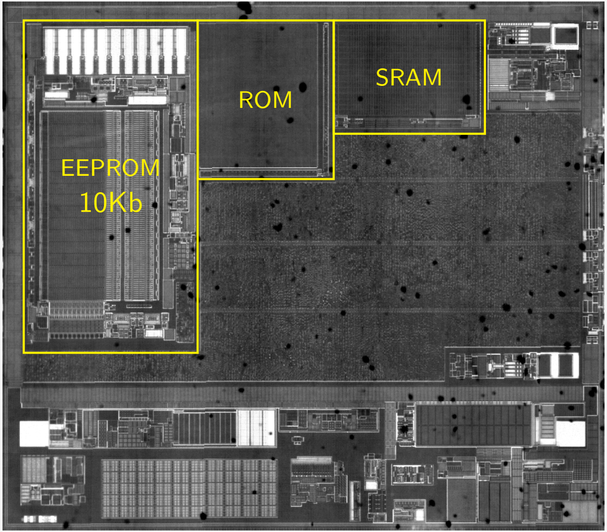
Only the ROM is updated.



ATECC DESCRIPTION / FLOORPLAN

All circuit revisions are based on the **same silicon hardware**

Only the ROM is updated.



ATECC DESCRIPTION / EEPROM ORGANIZATION

Config
128 bytes

Data 0
36 bytes

Data 1
36 bytes

Data 2
36 bytes

Data 3
36 bytes

OTP
64 bytes

Data 4
36 bytes

Data 5
36 bytes

Data 6
36 bytes

Data 7
36 bytes

Data 8
416 bytes

Data 9
72 bytes

Data 10
72 bytes

Data 11
72 bytes

Data 12
72 bytes

Data 13
72 bytes

Data 14
72 bytes

Data 15
72 bytes

ATECC DESCRIPTION / EEPROM ORGANIZATION

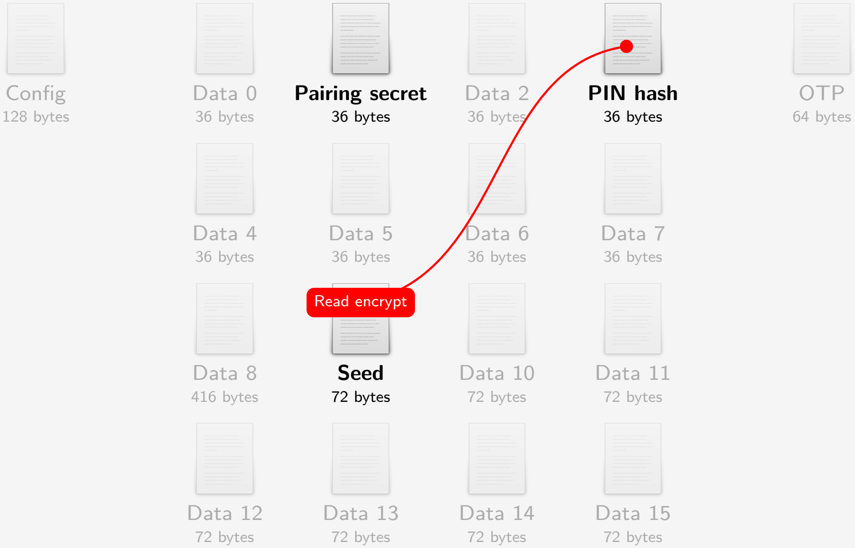
- Public files are stored in plaintext.
  No integrity protection.

- Configuration file is stored in plaintext.
  Integrity checked with checksum at boot time.

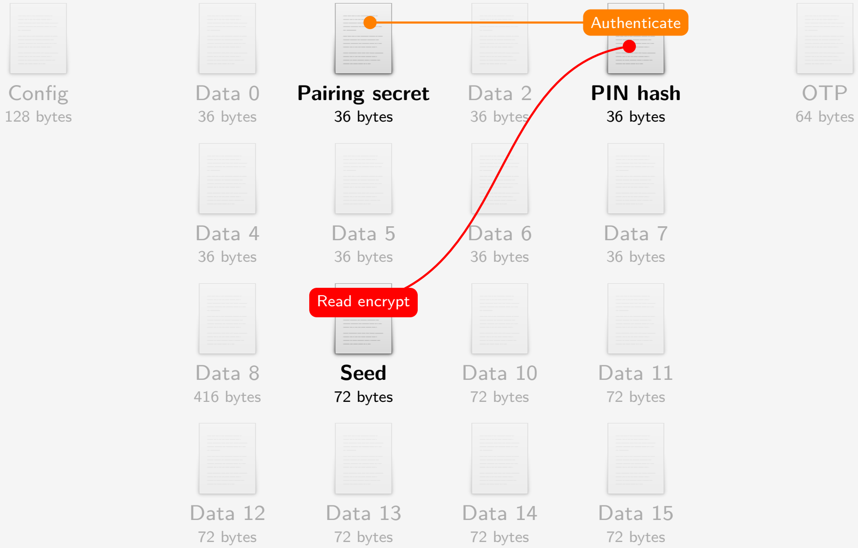
- Private files are stored encrypted with AES-128.
  Integrity protected (algorithm unknown).

ATECC DESCRIPTION / COLDCARD CONFIGURATION

Config 128 bytes	Data 0 36 bytes	**Pairing secret** 36 bytes	Data 2 36 bytes	**PIN hash** 36 bytes	OTP 64 bytes
	Data 4 36 bytes	Data 5 36 bytes	Data 6 36 bytes	Data 7 36 bytes	
	Data 8 416 bytes	**Seed** 72 bytes	Data 10 72 bytes	Data 11 72 bytes	
	Data 12 72 bytes	Data 13 72 bytes	Data 14 72 bytes	Data 15 72 bytes	


Config
128 bytes
Data 0
36 bytes
Pairing secret
36 bytes
Data 2
36 bytes
PIN hash
36 bytes
OTP
64 bytes
Data 4
36 bytes
Data 5
36 bytes
Data 6
36 bytes
Data 7
36 bytes
Read encrypt
Data 8
416 bytes
Seed
72 bytes
Data 10
72 bytes
Data 11
72 bytes
Data 12
72 bytes
Data 13
72 bytes
Data 14
72 bytes
Data 15
72 bytes

ATECC DESCRIPTION / COLDCARD CONFIGURATION



Accessing the file:

1. **Nonce** + **CheckMac** commands:
   Prove to the SE knowledge of the MCU $\leftrightarrow$ SE pairing secret.
2. **Nonce** + **GenDig** commands:
   Generate a session key for the next command encryption, derived using a MCU $\leftrightarrow$ SE shared secret.
3. **Read** command:
   Get the content of the file.
   Returned data is encrypted with the session key.

Vulnerability #1: Read command

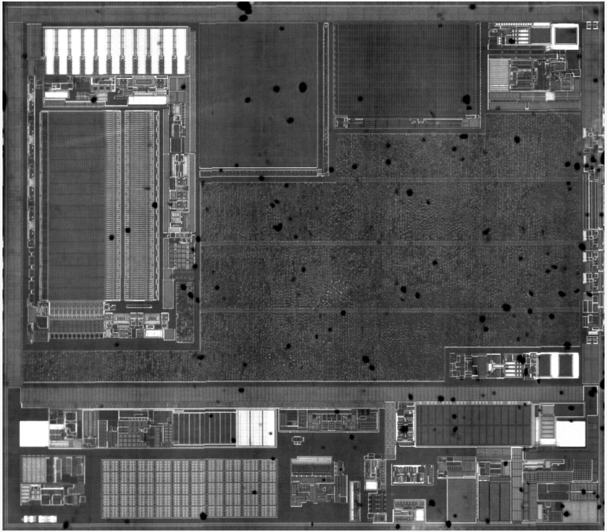
A straightforward attack path is to fault the **Read** command.

File access conditions are stored in EEPROM.
Fool the circuit, switch the configuration from **secret** to **public**.

- Works with a single fault on ATECC508A.
- Works with a double fault on ATECC608A.
- Let's investigate on ATECC608B!

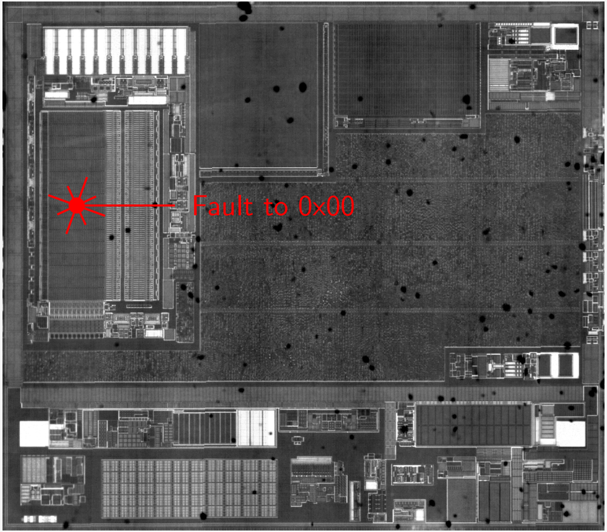
VULNERABILITY #1: READ COMMAND / SEED FILE ACCESS

The EEPROM is the weakness of the circuit.



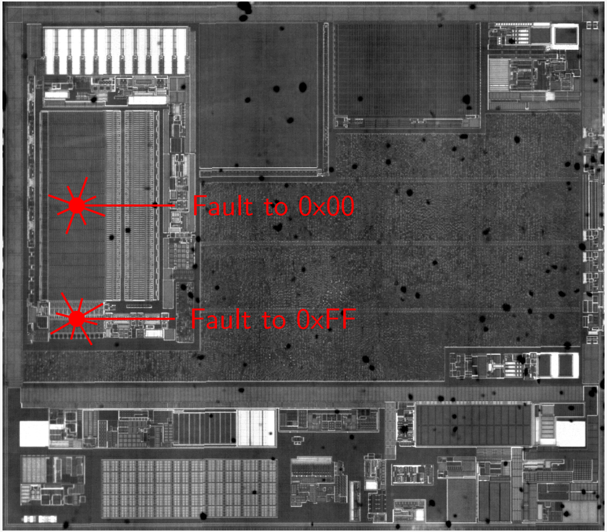
The EEPROM is the weakness of the circuit.

High fault success rate (~99%)
Powerful fault model



The EEPROM is the weakness of the circuit.

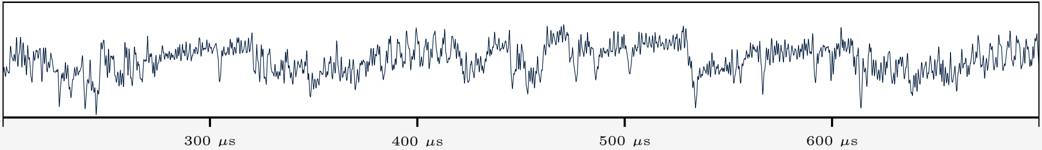
High fault success rate (~99%)
Powerful fault model



Microchip really hardened this command in the ATECC608B revision.

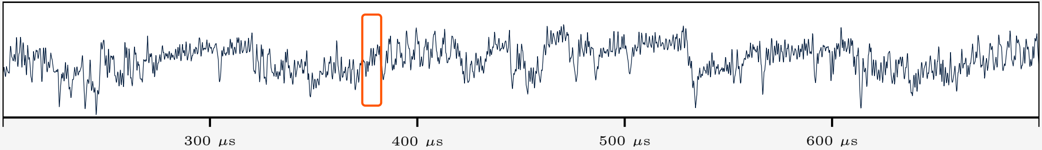
- Up to **8 security checks** instead of 2,
- New software **jitter** counter-measure.

VULNERABILITY #1: READ COMMAND / NEW COUNTER-MEASURES



ATECC608A

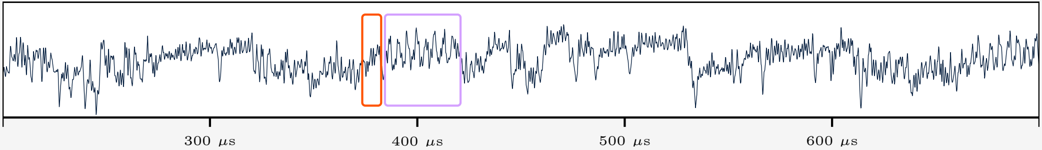
VULNERABILITY #1: READ COMMAND / NEW COUNTER-MEASURES



ATECC608A

Security check

VULNERABILITY #1: READ COMMAND / NEW COUNTER-MEASURES

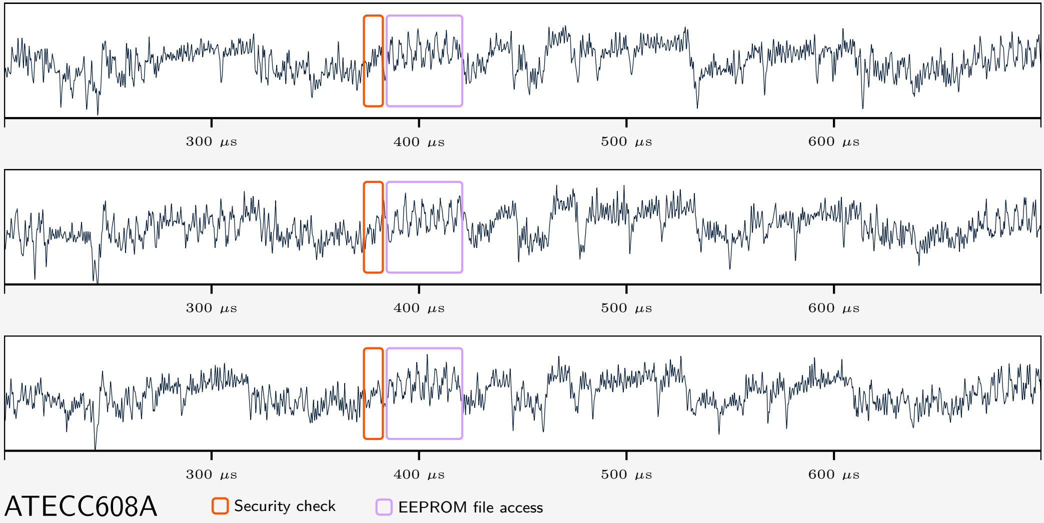


ATECC608A

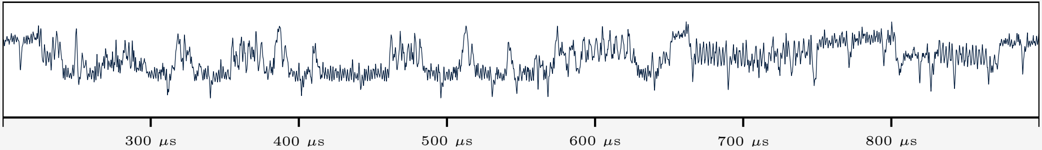
Security check

EEPROM file access

VULNERABILITY #1: READ COMMAND / NEW COUNTER-MEASURES

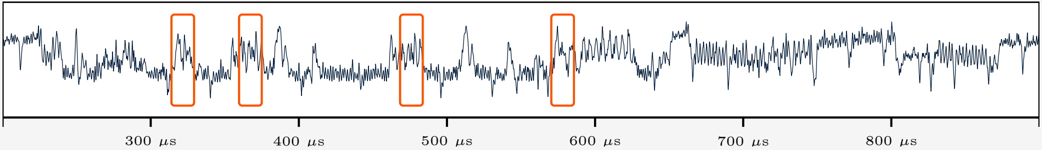


VULNERABILITY #1: READ COMMAND / NEW COUNTER-MEASURES



ATECC608B

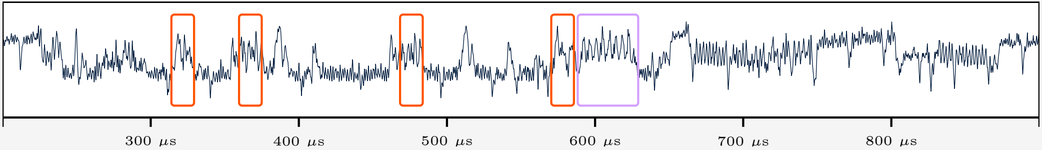
VULNERABILITY #1: READ COMMAND / NEW COUNTER-MEASURES



ATECC608B

Security check

VULNERABILITY #1: READ COMMAND / NEW COUNTER-MEASURES

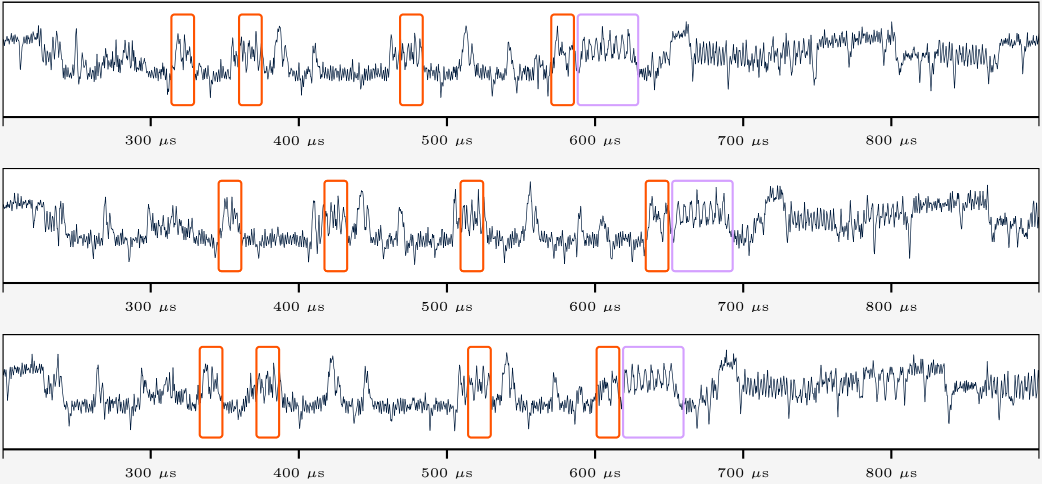


ATECC608B

Security check

EEPROM file access

VULNERABILITY #1: READ COMMAND / NEW COUNTER-MEASURES

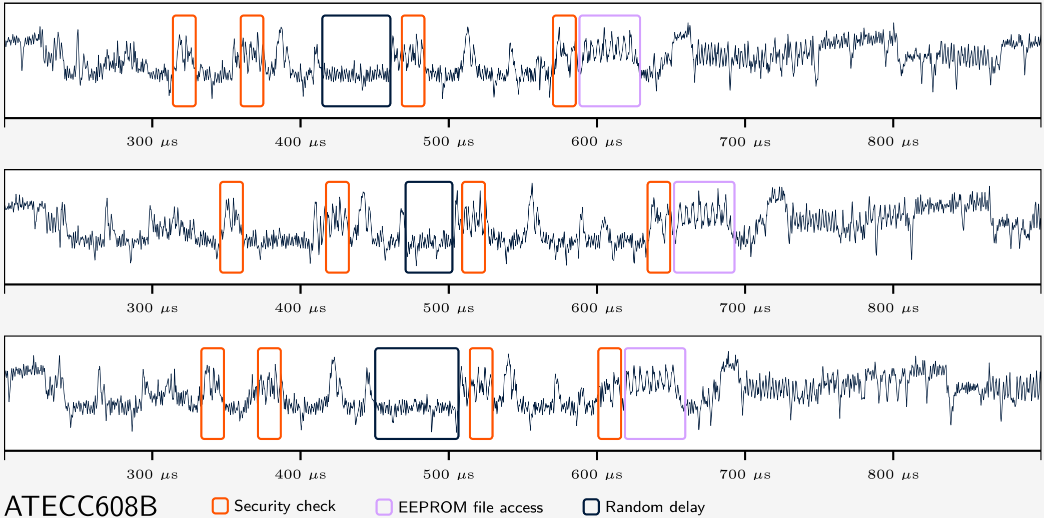


ATECC608B

Security check

EEPROM file access

VULNERABILITY #1: READ COMMAND / NEW COUNTER-MEASURES



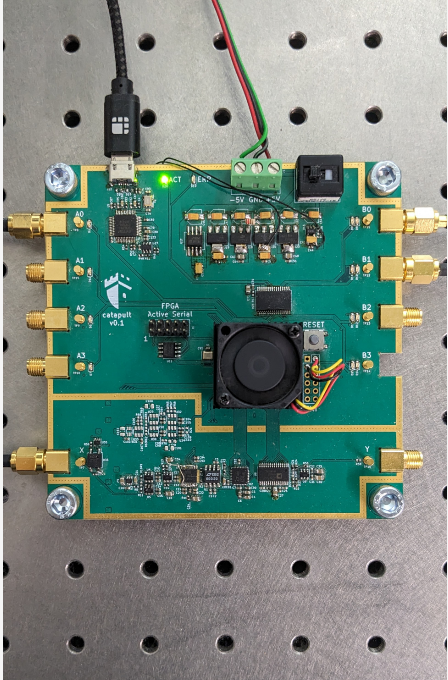
ATECC608B

VULNERABILITY #1: READ COMMAND

Because of jitter, success rate drops a lot.

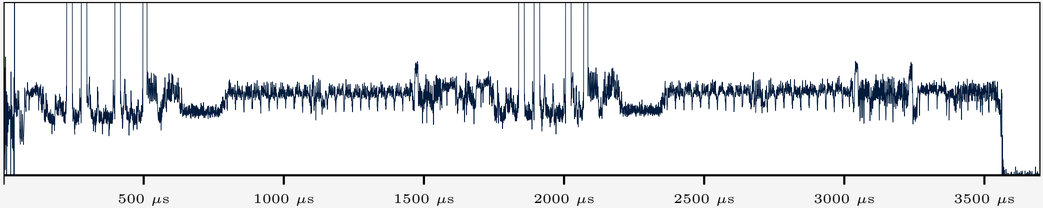
Patterns in the power trace can be easily identified.
Software random delays are flat on the power trace.

Use of real-time resynchronization hardware is possible.



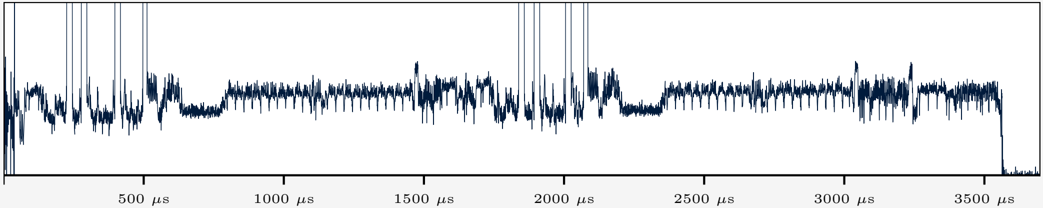
VULNERABILITY #1: READ COMMAND

We managed to **bypass all 8 security checks using 8 faults**.
Measured power trace matched the signature in case of allowed read.
Success rate is low ~0.1%.



VULNERABILITY #1: READ COMMAND

We managed to **bypass all 8 security checks using 8 faults**.
Measured power trace matched the signature in case of allowed read.
Success rate is low ~0.1%.



Returned data was incorrect.
File decryption key may be derived from the file configuration, which is corrupted during our attack.

ATECC have separate memories:

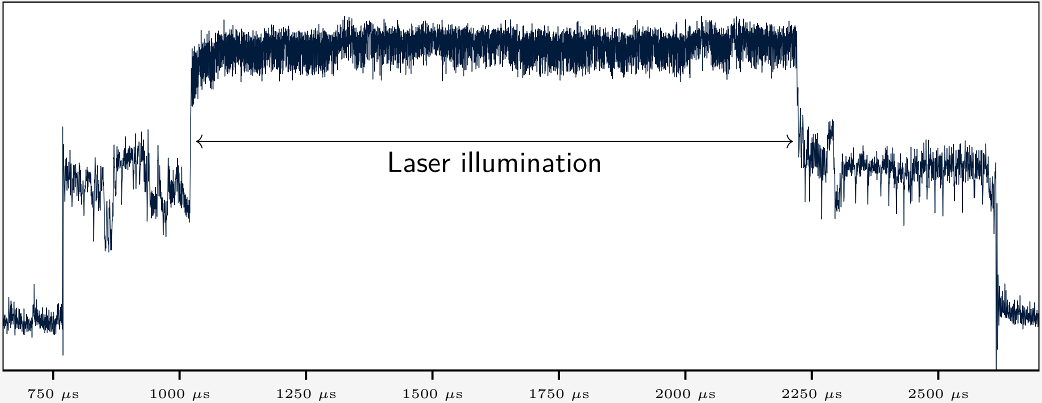
- User data and device configuration stored in EEPROM memory
- Firmware instructions stored in ROM memory
- Firmware program variables stored in RAM memory

Illuminating the EEPROM only faults EEPROM accesses, instruction fetches remain unmodified.

Shooting for almost the whole duration of the **Read** command:

1. Bypasses all security checks,
2. Disables file decryption,
3. Overrides EEPROM read content with zeros.

Power trace of long pulse injection during the **Read** command:



VULNERABILITY #1: READ COMMAND / LONG LASER PULSE TRICK

Expected response:

0000000000000000000000000000000000000000000000000000000000000000000000

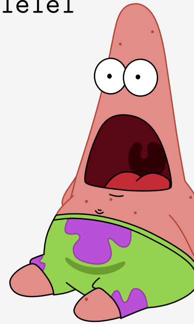
VULNERABILITY #1: READ COMMAND / LONG LASER PULSE TRICK

Expected response:

0000000000000000000000000000000000000000000000000000000000000000

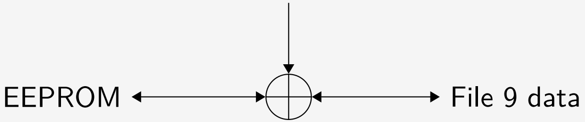
Obtained response:

10103030909 0d0d0a4a4e4e4b1b1f1f18080a0a08080c0c0a4a4e4e4a1a1e1e1



We discovered an internal **EEPROM masking key**:

m_9 = 101030309090d0d0a4a4e4e4b1b1f1f18080a0a08080c0c0a4a4e4e4a1a1e1e1



Key is **derived from the file number**, and is different from chip to chip.
Key derivation mechanism is unknown, and has (obviously) low entropy.

It takes a few minutes to extract all 16 masking keys.

Hypothesis confirmed with fault model complementary:

Obtained response by faulting EEPROM to **0x00**:

`101030309090d0d0a4a4e4e4b1b1f1f18080a0a08080c0c0a4a4e4e4a1a1e1e1`

Obtained response by faulting EEPROM to **0xFF**:

`efefcfcf6f6f2f2f5b5b1b1b4e4e0e0e7f7f5f5f7f7f3f3f5b5b1b1b5e5e1e1e`

VULNERABILITY #1: READ COMMAND / EEPROM MASKING KEYS DISCOVERY

m_0	58523832d0d0d0d0ece6ece6f1f1f1f1c8c2a8a2c0c0c0c0ece6ece6e1e1e1e1
m_1	50503030d0d0d0d0e4e4e4e4f1f1f1f1c0c0a0a0c0c0c0c0e4e4e4e4e1e1e1e1
m_2	5a423a22d2c0d2c0ee66ee66f371f371ca42aa22c240c240ee66ee66e361e361
m_3	52403220d2c0d2c0e664e664f371f371c240a220c240c240e664e664e361e361
m_4	78727872f0f0f0f0e8e2e8e2f0f0f0f0e8e2e8e2e0e0e0e0e8e2e8e2e0e0e0e0
m_5	70707070f0f0f0f0e0e0e0e0f0f0f0f0e0e0e0e0e0e0e0e0e0e0e0e0e0e0e0e0
m_6	7a627a62f2e0f2e0ea62ea62f270f270ea62ea62e260e260ea62ea62e260e260
m_7	72607260f2e0f2e0e260e260f270f270e260e260e260e260e260e260e260e260
m_8	181238329090d0d0aca6ece6b1b1f1f18882a8a28080c0c0acb6ece6a1a1e1e1
m_9	101030309090d0d0a4a4e4e4b1b1f1f18080a0a08080c0c0a4a4e4e4a1a1e1e1
m_{10}	1a023a229280d2c0ae26ee66b331f3718a02aa228200c240ae26ee66a321e361
m_{11}	120032209280d2c0a624e664b331f3718200a2208200c240a624e664a321e361
m_{12}	38327872b0b0f0f0a8a2e8e2b0b0f0f0a8a2e8e2a0a0e0e0a8a2e8e2a0a0e0e0
m_{13}	30307070b0b0f0f0a0a0e0e0b0b0f0f0a0a0e0e0a0a0e0e0a0a0e0e0a0a0e0e0
m_{14}	3a227a62b2a0f2e0aa22ea62b230f270aa22ea62a220e260aa22ea62a220e260
m_{15}	32207260b2a0f2e0a220e260b230f270a220e260a220e260a220e260a220e260

VULNERABILITY #1: READ COMMAND / EEPROM MASKING KEYS DISCOVERY

m_0	58523832d0d0d0d0ece6ece6f1f1f1f1c8c2a8a2c0c0c0c0ece6ece6e1e1e1e1
m_1	50503030d0d0d0d0e4e4e4e4f1f1f1f1c0c0a0a0c0c0c0c0e4e4e4e4e1e1e1e1
m_2	5a423a22d2c0d2c0ee66ee66f371f371ca42aa22c240c240ee66ee66e361e361
m_3	52403220d2c0d2c0e664e664f371f371c240a220c240c240e664e664e361e361
m_4	78727872f0f0f0f0e8e2e8e2f0f0f0f0e8e2e8e2e0e0e0e0e8e2e8e2e0e0e0e0
m_5	70707070f0f0f0f0e0e0e0e0f0f0f0f0e0e0e0e0e0e0e0e0e0e0e0e0e0e0e0e0
m_6	7a627a62f2e0f2e0ea62ea62f270f270ea62ea62e260e260ea62ea62e260e260
m_7	72607260f2e0f2e0e260e260f270f270e260e260e260e260e260e260e260e260
m_8	18123832909 0d0d0aca6ece6b1b1f1f18882a8a28080c0c0acb6ece6a1a1e1e1
m_9	10103030909 0d0d0a4a4e4e4b1b1f1f18080a0a08080c0c0a4a4e4e4a1a1e1e1
m_{10}	1a023a229280d2c0ae26ee66b331f3718a02aa228200c240ae26ee66a321e361
m_{11}	120032209280d2c0a624e664b331f3718200a2208200c240a624e664a321e361
m_{12}	38327872b0b0f0f0a8a2e8e2b0b0f0f0a8a2e8e2a0a0e0e0a8a2e8e2a0a0e0e0
m_{13}	30307070b0b0f0f0a0a0e0e0b0b0f0f0a0a0e0e0a0a0e0e0a0a0e0e0a0a0e0e0
m_{14}	3a227a62b2a0f2e0aa22ea62b230f270aa22ea62a220e260aa22ea62a220e260
m_{15}	32207260b2a0f2e0a220e260b230f270a220e260a220e260a220e260a220e260

attack1.mp4

VULNERABILITY #1: READ COMMAND / NEW PATH

Accessing the file:

- **Nonce** + **CheckMac** commands:
  Prove to the SE knowledge of the MCU ↔ SE pairing secret.
- **Nonce** + **GenDig** commands:
  Generate a session key for the next command encryption, derived using a MCU ↔ SE shared secret.
- **Read** command:
  Get the content of the file.
  Returned data is encrypted with the session key.

Accessing the file:

- **Nonce** + **CheckMac** commands:
  Prove to the SE knowledge of the MCU ↔ SE pairing secret.
- **Nonce** + **GenDig** commands:
  Generate a session key for the next command encryption, derived using a MCU ↔ SE shared secret.
- **Read** command:
  Get the content of the file.
  Returned data is encrypted with the session key.

New attack path: **get authenticated** and **enforce a chosen session key**.

Vulnerability #2: CheckMac command

VULNERABILITY #2: CHECKMAC COMMAND

With **CheckMac** challenge, MCU proves knowledge of the pairing secret d, by answering the correct digest value h:

$$h = \text{SHA-256}(d \mid r \mid o)$$

VULNERABILITY #2: CHECKMAC COMMAND

With **CheckMac** challenge, MCU proves knowledge of the pairing secret d, by answering the correct digest value h:

$$h = \text{SHA-256}(d \mid r \mid o)$$

During verification, the secret d is read by the firmware from the EEPROM:

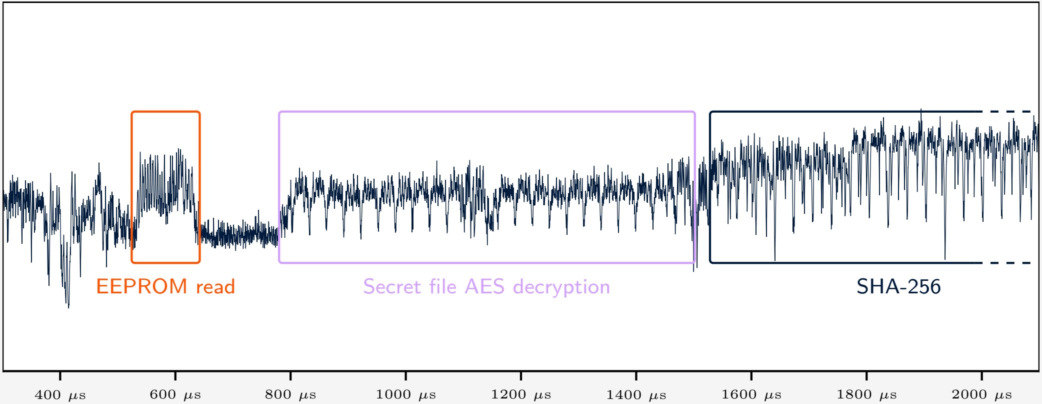
Secret files are stored encrypted

$$d = \text{AES}^{-1}(e \oplus m_1)$$

EEPROM raw output

Internal secret mask

VULNERABILITY #2: CHECKMAC COMMAND / POWER TRACE



$$d = \mathsf{AES}^{-1}(e \oplus m_1)$$

With a single 200 μs long laser illumination, we managed to:

VULNERABILITY #2: CHECKMAC COMMAND / EXPLOITATION

$$d = \cancel{\mathsf{AES}^{-1}}(e \oplus m_1)$$

With a single 200 μs long laser illumination, we managed to:

1. Disable file decryption

$$d = \cancel{\text{AES}}^{-1}(0 \oplus m_1)$$

With a single 200 μs long laser illumination, we managed to:

1. Disable file decryption
2. Override EEPROM output with zeros (32 bytes faulted)

VULNERABILITY #2: CHECKMAC COMMAND / EXPLOITATION

$$d = m_1$$

With a single 200 μs long laser illumination, we managed to:

1. Disable file decryption
2. Override EEPROM output with zeros (32 bytes faulted)

VULNERABILITY #2: CHECKMAC COMMAND / EXPLOITATION

$$d = m_1$$

With a single 200 μs long laser illumination, we managed to:

1. Disable file decryption
2. Override EEPROM output with zeros (32 bytes faulted)

This leads to:

$$h = \text{SHA-256}(m_1 \mid r \mid o)$$

As we extracted m_1, we managed to answer this faulted challenge and get authenticated.

attack2.mp4

Vulnerability #3: GenDig command

VULNERABILITY #3: GENDIG COMMAND / EXPLOITATION

The same attack method works for the key derivation **GenDig** command.
Laser pulse delay and duration slightly different.

$$k = \text{SHA-256}(\text{AES}^{-1}(e \oplus m_3) \mid o \mid r)$$

VULNERABILITY #3: GENDIG COMMAND / EXPLOITATION

The same attack method works for the key derivation **GenDig** command.
Laser pulse delay and duration slightly different.

$$k = \text{SHA-256}(\cancel{\text{AES}^{-1}}(e \oplus m_3) \mid o \mid r)$$

VULNERABILITY #3: GENDIG COMMAND / EXPLOITATION

The same attack method works for the key derivation **GenDig** command.
Laser pulse delay and duration slightly different.

$$k = \text{SHA-256}((e \oplus m_3) \mid o \mid r)$$

VULNERABILITY #3: GENDIG COMMAND / EXPLOITATION

The same attack method works for the key derivation **GenDig** command.
Laser pulse delay and duration slightly different.

$$k = \text{SHA-256}((0 \oplus m_3) \mid o \mid r)$$

VULNERABILITY #3: GENDIG COMMAND / EXPLOITATION

The same attack method works for the key derivation **GenDig** command.
Laser pulse delay and duration slightly different.

$$k = \text{SHA-256}(m_3 \mid o \mid r)$$

The same attack method works for the key derivation **GenDig** command.
Laser pulse delay and duration slightly different.

$$k = \text{SHA-256}(m_3 \mid o \mid r)$$

This session key from faulted **GenDig** execution can be calculated by the attacker.

We can then just call the **Read** command, **without faulting it**.

Remark: When **GenDig** attack fails, decrypted content is invalid, and unique. There is no "success" status code.

Attack is performed many times, until some decrypted output is found twice.

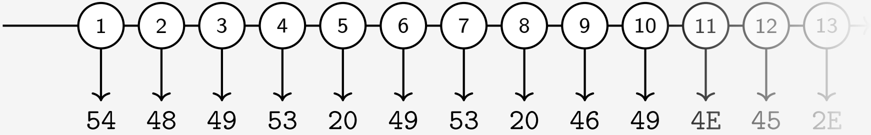
1. Corrupt the **Read** command twice with laser to get masks m_1 and m_3.
   Success rate ~50% / ~2%.
2. Hijack the **CheckMac** command with laser and knowledge of m_1.
   This attack results in successful authentication, allowing usage of **GenDig**.
   Success rate ~40% / ~5%.
3. Hijack the **GenDig** command with laser and knowledge of m_3. This attack generates a session key for the **Read** command.
   Success rate ~20% / ~1%.
4. Call (without fault injection) the **Read** command to get the secret.
   Chip's response is decrypted with the session key.

attack3.mp4

Counter-measure

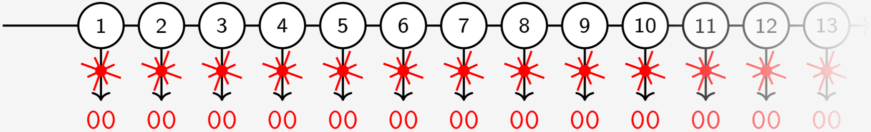
COUNTER-MEASURE

EEPROM 32 bytes readout easily manipulated:



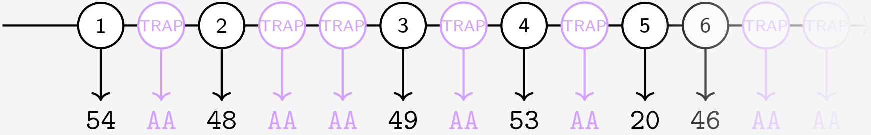
COUNTER-MEASURE

EEPROM 32 bytes readout easily manipulated:



COUNTER-MEASURE

Insert dummy trap memory accesses, returning a verified magic number:



Faulting any **trap access** to 0x00 or 0xFF can be detected by the firmware.

Coldcard Mk3 Challenge

COLDCARD MK3 CHALLENGE

Coinkite shipped us 3 preconfigured wallets.

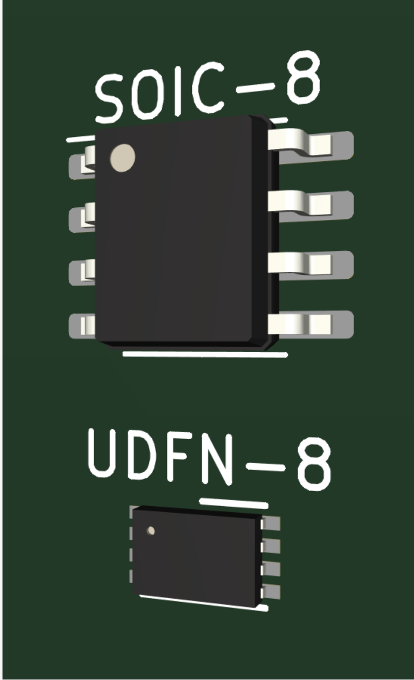
Attacking real devices is not that easy.

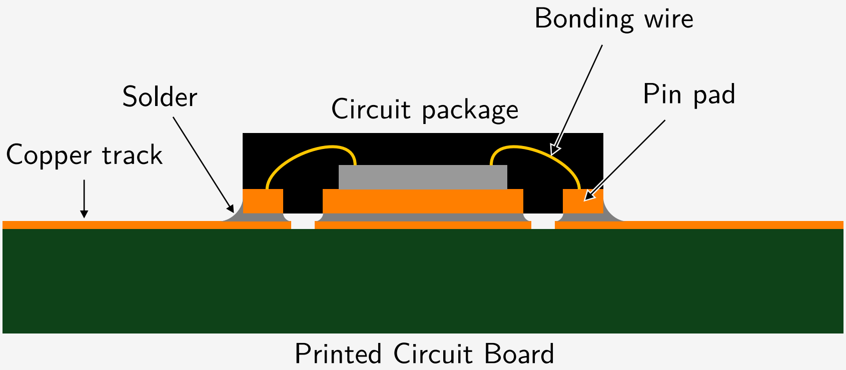
During research, we trained using **SOIC-8**[3] packages to ease sample preparation.

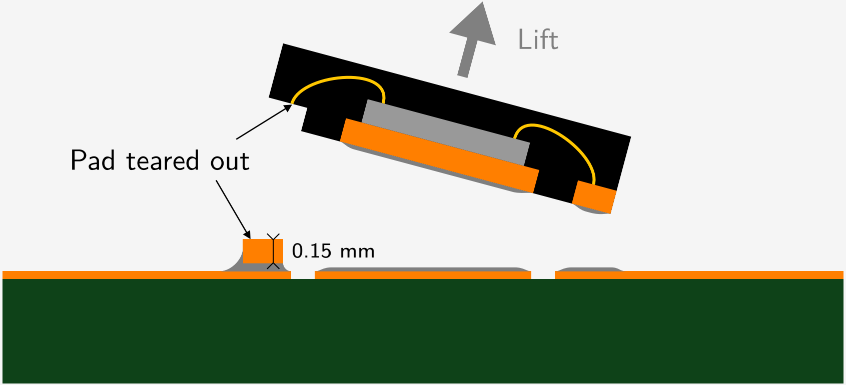
Coldcard wallets embeds **UDFN-8**[4] packages.

Package	Width	Height	Thickness
SOIC-8	6 mm	3.9 mm	1.25 mm
UDFN-8	3 mm	2 mm	0.5 mm

[3]Small Outline Integrated Circuit, 8 pins
[4]Ultra Thin Plastic Dual Flat No Lead, 8 pins




Bonding wire
Solder
Circuit package
Pin pad
Copper track
Printed Circuit Board

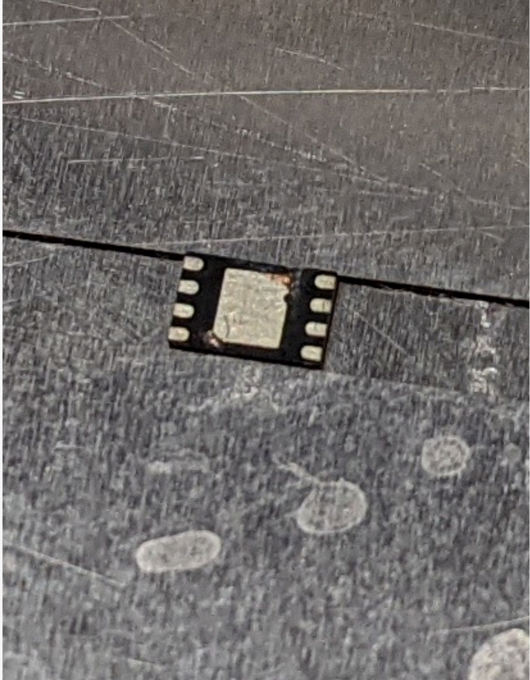

Lift
Pad teared out
0.15 mm

COLDCARD MK3 CHALLENGE

Package surface area is small.
Adhesive tape is not strong enough to hold sample.

Sample is locked with 4 metal sheets.

Validated on a few testing samples.

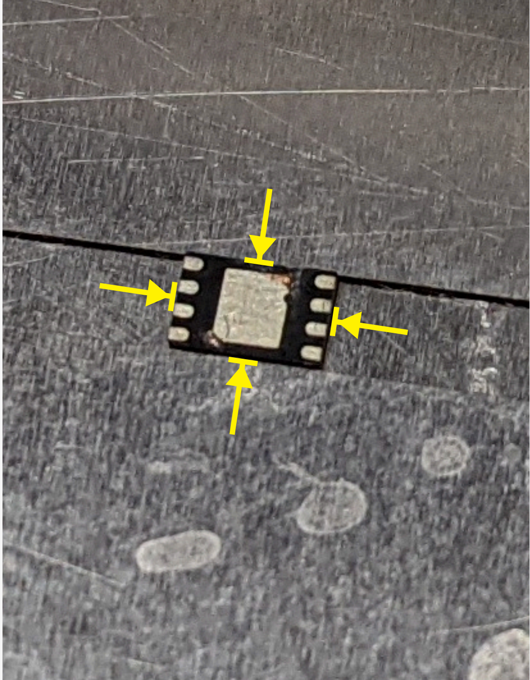


COLDCARD MK3 CHALLENGE

Package surface area is small.
Adhesive tape is not strong enough to hold sample.

Sample is locked with 4 metal sheets.

Validated on a few testing samples.

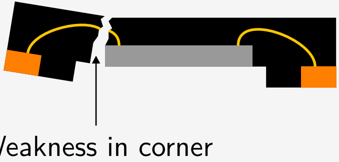


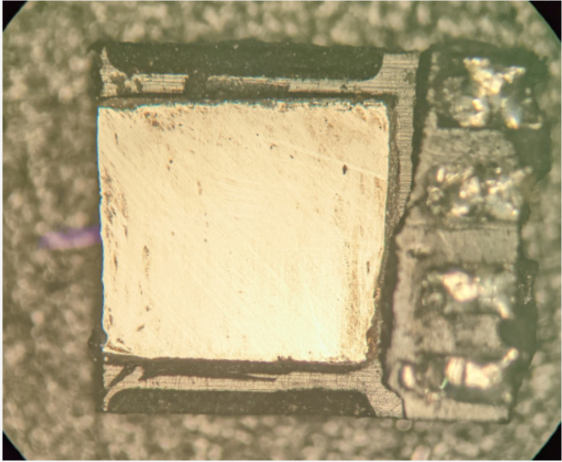
COLDCARD MK3 CHALLENGE

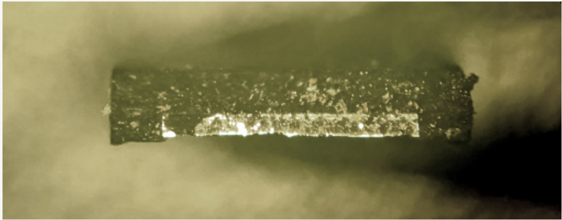
Package broke during milling.

Result of excessive mechanical stress.

Package weakened by high temperature during desoldering?







Package resining

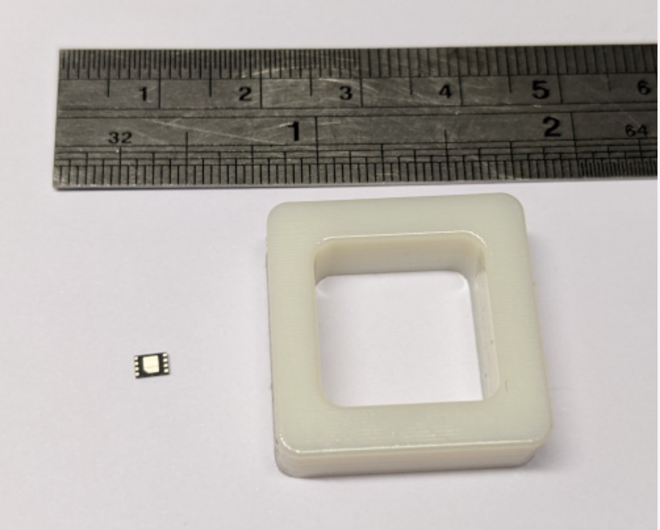
PACKAGE RESINING

Sample is too small and fragile.

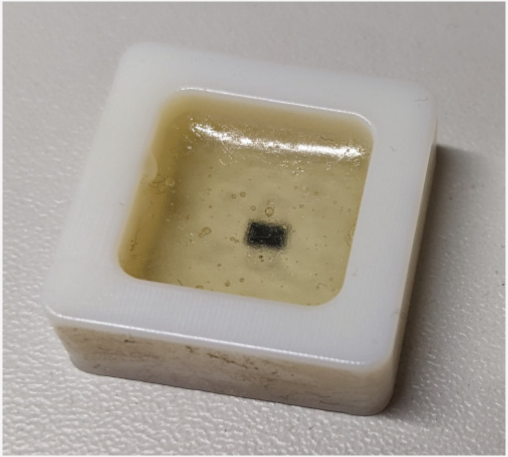
Simple solution:
We can make a stronger package using **epoxy resin**.

Note:
+24 hours required for hardening.
Attack is no more possible within a day.

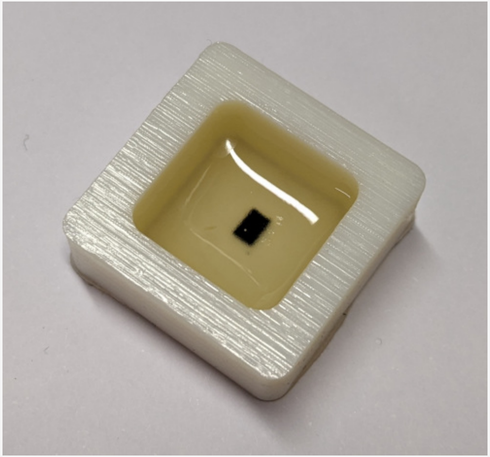
PACKAGE RESINING / RESING MOULD



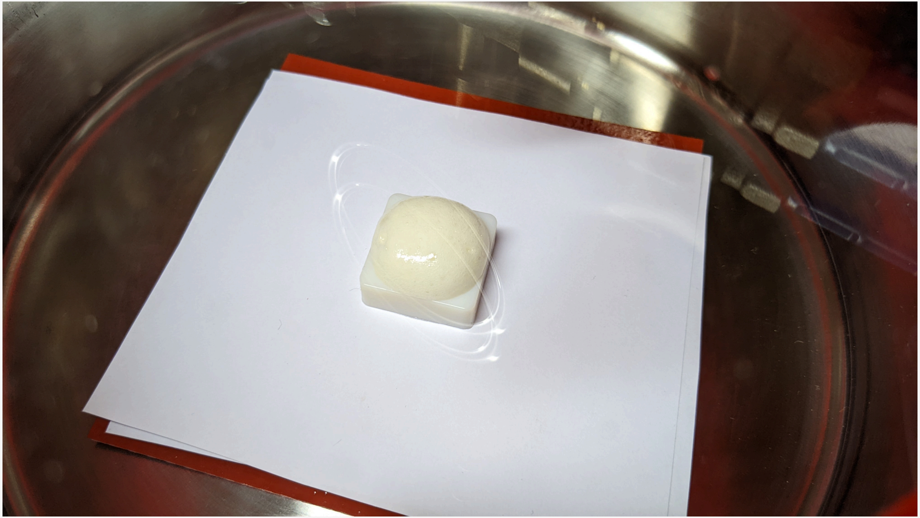
PACKAGE RESINING / RESIN POURING



First attempt

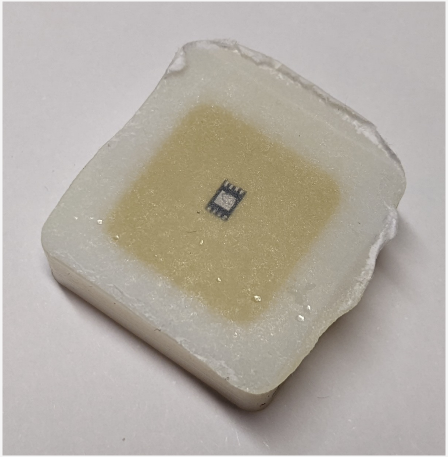


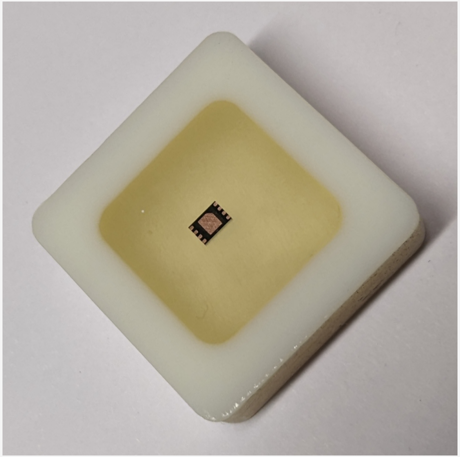
Bubbles removed



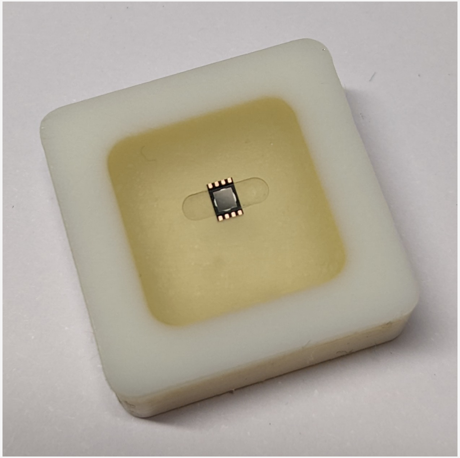
It is easier to remove bubbles before pouring.

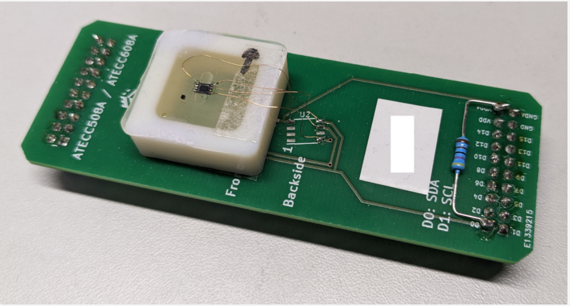
PACKAGE RESINING / SANDING





PACKAGE RESINING / DEPACKAGING AND SOLDERING





RESULTS AND CONCLUSION

Challenge result: **1/3 wallet seed recovered.**
Yield increases with experience.

- **ATECC608B**: Fault injection attack safe.
  Sample preparation risky.
- **STM32L496**: Fault injection attack very risky.
  Sample preparation easy.

We demonstrated the attack is practical.
Cryptocurrencies hardware wallets are high value targets, hence such attacks should be considered realistic.

RESULTS AND CONCLUSION

Vulnerabilities were responsibly disclosed to Coinkite and Microchip.
A very long period of time was granted to vendors before publication.

A secondary SE was added to Coldcard Mk4 hardware wallet.[5]
Secret is now split in 3 shares.

ATECC family security greatly enhanced over years.
EEPROM still remains a weakness to consider, other commands may be vulnerable.

Microchip **released ATECC608C** in August 2023.
Not yet investigated.

[5] Karim M. Abdellatif et al.: DeepCover DS28C36: A Hardware Vulnerability Identification and Exploitation Using T-Test and Double Laser Fault Injection - FDTC 2023

Questions?