LEDGER DONJON

Triple Exploit Chain with Laser Fault Injection on a Secure Element

Olivier Hériveaux - FDTC 2022

PREVIOUS WORK

ATECC508A single laser fault injection vulnerability reported in 2020.
ATECC<u>6</u>08A double laser fault injection vulnerability reported in 2021.

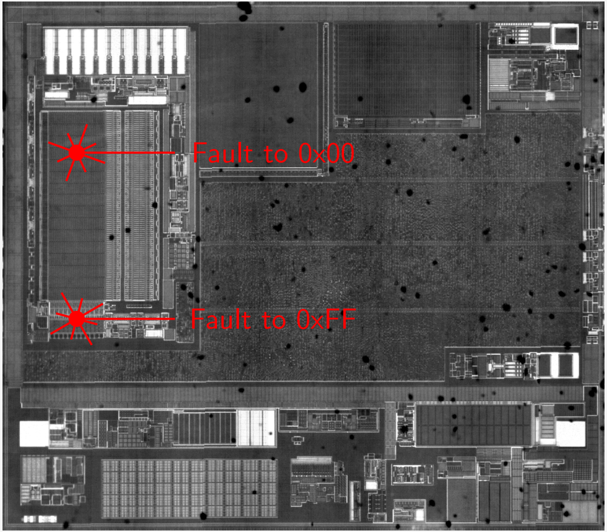
Following those reports, Microchip released a new revision of the circuit.

This work highlights the new counter-measures in the **ATECC608<u>B</u>** and presents three newly identified vulnerabilities in other commands.

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

Only the ROM is updated.

The EEPROM is the weakness of the circuit.



NEW COUNTER-MEASURES

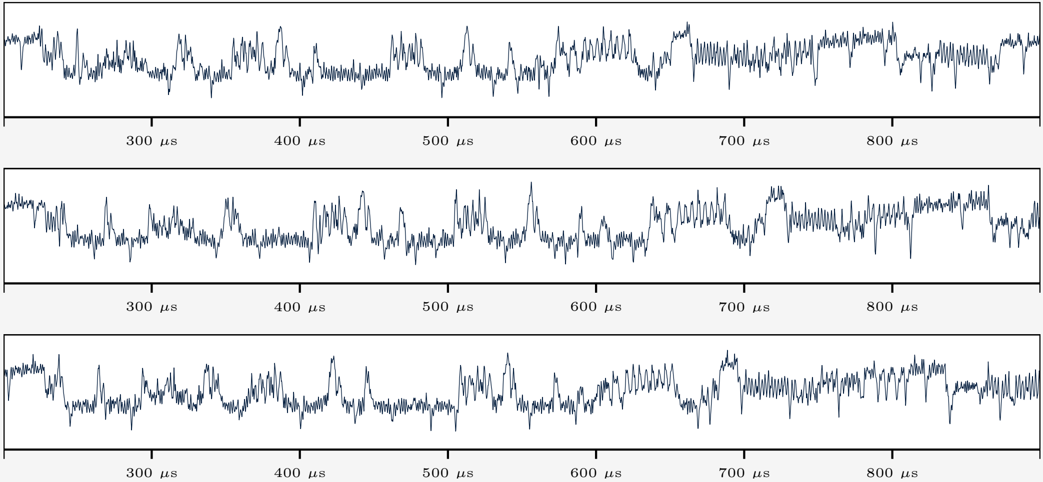
User assets are stored inside "data slots".
Data slots can be configured as public or secret.

In previous silicon revisions, the **Read** command could be faulted to switch a data slot configuration from **secret** to **public**.

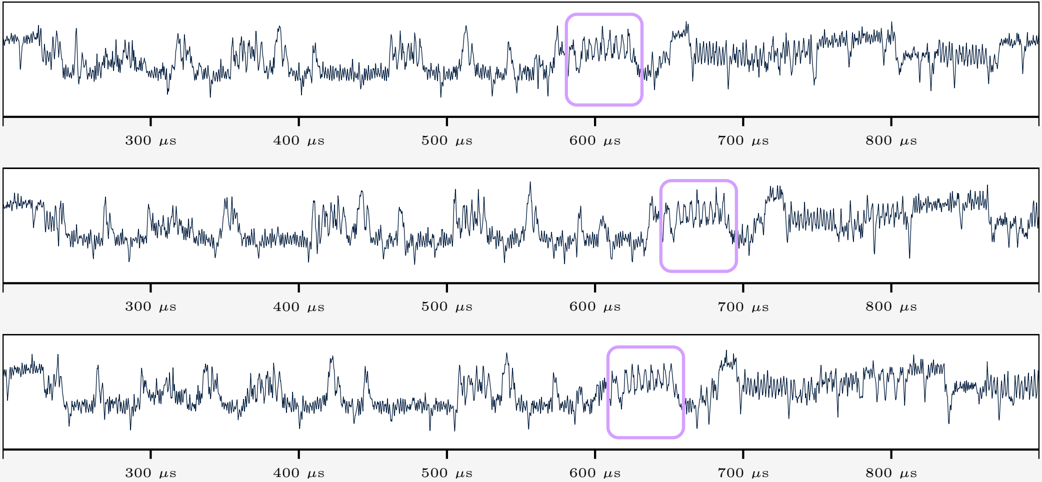
Microchip hardened this command in the ATECC608B revision:

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

NEW COUNTER-MEASURES / JITTER



NEW COUNTER-MEASURES / JITTER



READ COMMAND ATTACK TRIAL

Using realtime resynchronization, we managed to bypass all 8 security checks, using 8 faults.

but...

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

EEPROM MASKING KEYS DISCOVERY

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

- Bypasses all security checks,
- Disables data slot decryption,
- Overrides EEPROM read content with zeros.

Expected result:
0000000000000000000000000000000000000000000000000000000000000000

EEPROM MASKING KEYS DISCOVERY

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

- Bypasses all security checks,
- Disables data slot decryption,
- Overrides EEPROM read content with zeros.

Expected result:
0000000000000000000000000000000000000000000000000000000000000000

Obtained result:
101030309090d0d0a4a4e4e4b1b1f1f18080a0a08080c0c0a4a4e4e4a1a1e1e1

EEPROM MASKING KEYS DISCOVERY

Obtained result:
`101030309090d0d0a4a4e4e4b1b1f1f18080a0a08080c0c0a4a4e4e4a1a1e1e1`

We extracted an internal **EEPROM masking key**.

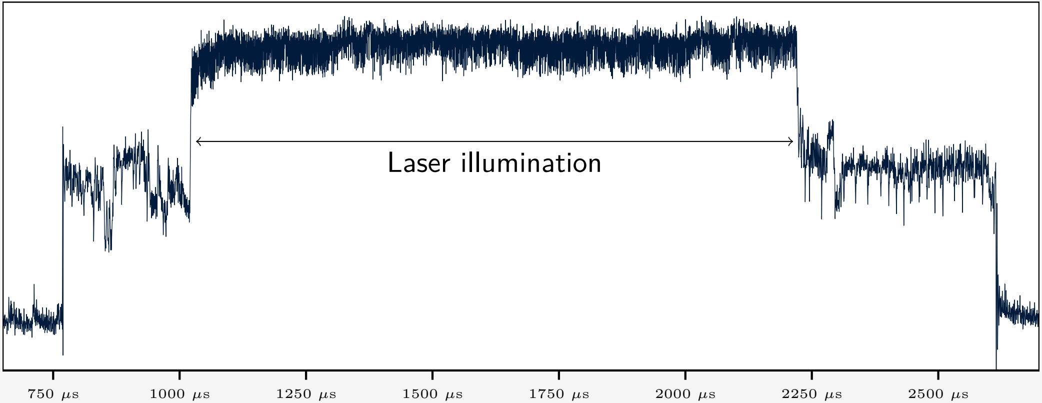
Key is **derived from the data slot number**, and is different from chip to chip.

Key derivation mechanism is unknown, and has low entropy.

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

EEPROM MASKING KEYS DISCOVERY

Power trace of masking key extraction during the **Read** command:



NEW ATTACK PATH

Accessing the file:

- **Nonce** + **CheckMac** commands:
  Prove to the SE knowledge of the MCU $\leftrightarrow$ SE pairing secret.

NEW ATTACK PATH

Accessing the file:

- **Nonce** + **CheckMac** commands:
  Prove to the SE knowledge of the MCU $\leftrightarrow$ SE pairing secret.
- **Nonce** + **GenDig** commands:
  Generate a session key for the next command encryption, derived using a MCU $\leftrightarrow$ SE shared secret.

NEW ATTACK PATH

Accessing the file:

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

NEW ATTACK PATH

Accessing the file:

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

New attack path: **get authenticated and generate a known session key**.

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

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

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 data slots are stored encrypted

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

EEPROM raw output

Internal secret mask

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

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

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

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

1. Disable data slot decryption

NEW ATTACK PATH / HIJACKING CHECKMAC COMMAND

$$d = \cancel{\mathsf{AES}^{-1}}(0 \oplus m)$$

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

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

NEW ATTACK PATH / HIJACKING CHECKMAC COMMAND

$$d = m$$

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

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

$$d = m$$

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

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

This leads to:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1. Corrupt the **Read** command twice with laser to get masks m_1 and m_2.
   Success rate ~50%.
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%.
3. Hijack the **GenDig** command with laser and knowledge of m_2. This attack generates a session key for the **Read** command.
   Success rate ~20%.
4. Call (without fault injection) the **Read** command to get the secret.
   Chip's response is decrypted with the session key.

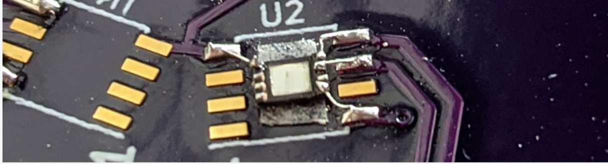
We successfully recovered secret data in three hardware wallets using this method.

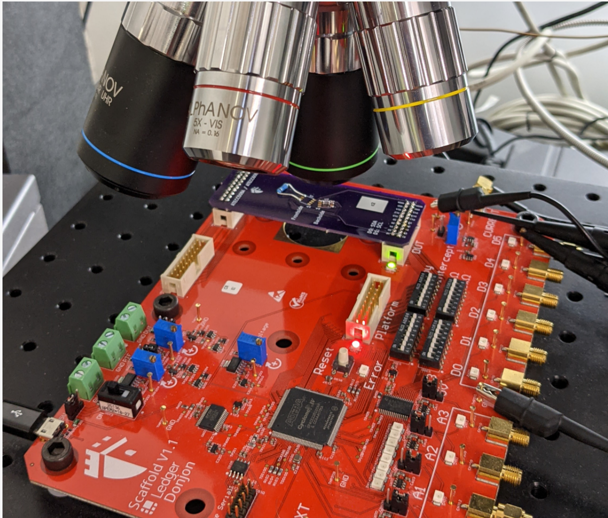
SETUP

Backside access, no silicon thinning

Scaffold board for communication and power trace monitoring

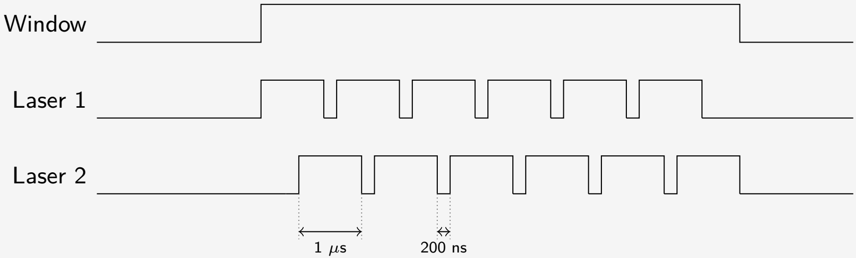
AlphaNov PDM+ IR laser sources





Train of pulses

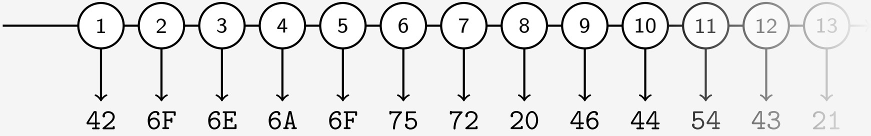
Two laser sources with dephased control signals



$\Rightarrow$ Always at least one laser ON during illumination window

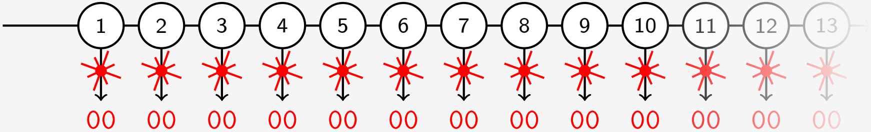
COUNTER-MEASURE

EEPROM 32 bytes readout easily manipulated:



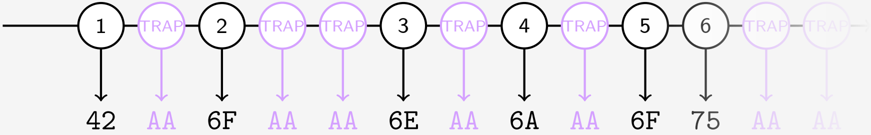
COUNTER-MEASURE

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` will be detected by the firmware.

Questions?