LEDGER

Unlimited Results: Breaking Firmware Encryption of ESP32-V3

Karim M. Abdellatif, Olivier Hériveaux, and Adrian Thillard

Motivation

- ESP32 is deployed in hundreds of million devices as announced by Espressif [1]
- ESP32-V3 has been recently used as the main MCU in Jade hardware wallet (Blockstream)[2]
  - Encrypted firmware is stored in the external flash
  - The encryption key is stored in the eFuses of ESP32-V3

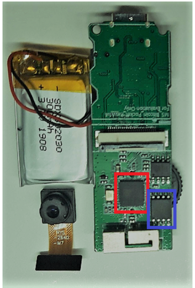
[1] Espressif, "Espressif Achieves the 100-Million Target for IoT Chip Shipments", 2018

[2] https://blockstream.com/jade/

Motivation

- ESP32 is deployed in hundreds of million devices as announced by Espressif [1]
- ESP32-V3 has been recently used as the main MCU in Jade hardware wallet (Blockstream)[2]
    - Encrypted firmware is stored in the external flash
    - The encryption key is stored in the eFuses of ESP32-V3

Jade wallet



ESP32-V3 + external flash

[1] Espressif, "Espressif Achieves the 100-Million Target for IoT Chip Shipments", 2018

[2] https://blockstream.com/jade/

ESP32-V1 vs ESP32-V3

ESP32-V1

- Flash encryption and secure boot were broken by LimitedResults[3] in 2019
- During the power-up eFuse protection bits are manipulated
- The main idea is to glitch the chip during the power-up

[3] LimitedResults, "Fatal Fury On ESP32: Time to Release HW Exploits", Blackhat Europe 2019

ESP32-V1 vs ESP32-V3

ESP32-V1

ESP32-V3

- Flash encryption and secure boot were broken by LimitedResults[3] in 2019
- During the power-up eFuse protection bits are manipulated
- The main idea is to glitch the chip during the power-up

- In the market since 2020 as a reaction against the previous attack
- New secure boot mechanism
- It is hardened against fault injection attacks in hardware and software as announced by the vendor

[3] LimitedResults, "Fatal Fury On ESP32: Time to Release HW Exploits", Blackhat Europe 2019

Outline

ESP32 Security Analysis

Fault Injection Setup

EMFI on ESP32-V1

EMFI on ESP32-V3

Breaking Firmware Encryption by SCAs

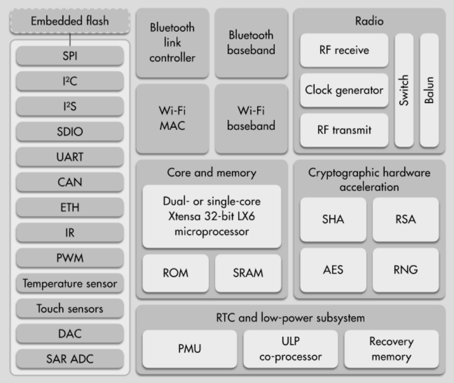
Practical Attack

Vendor reply and Conclusion

ESP32 SECURITY ANALYSIS

Security features

- Secure boot
- Flash memory encryption
- 1024-bit OTP, up to 768 bits for customers
- Cryptographic hardware accelerators: AES, SHA-2, RSA, Elliptic Curve Cryptography (ECC), and Random Number Generator (RNG)
- esptool[4] can be used to configure the above features



Source: Espressif

[4] https://github.com/espressif/esptool

eFuse organization

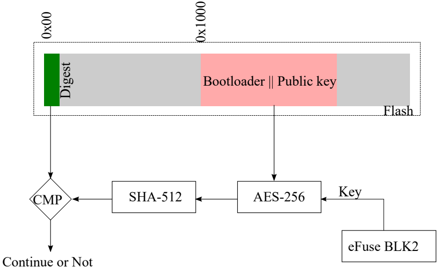
Reserved (System Purposes)	Flash Encryption Key	Secure Boot Key	User Application
BLK0	BLK1	BLK2	BLK3

- ESP32 (including V3) has a 1024-bits eFuse memory
- It is divided into 4 blocks of 256 bits each
- After burning these keys, can not be accessed (or updated) by any software
- Only the ESP32 hardware can read and use BLK1 and BLK2 for performing secure boot and flash encryption

Secure boot V1

$$Digest = \text{SHA-512}(\text{AES-256}((Bootloader \parallel public\ key), BLK2))) \tag{1}$$

```
burn_efuse ABS_DONE_0
```

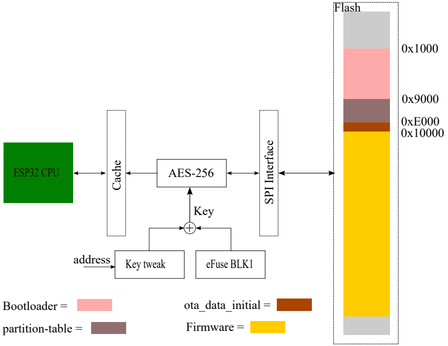


Signature verification

Flash encryption

- It encrypts all the flash content using AES-256 with BLK1 and stores it in the external memory
- Flash encryption uses AES decryption
- Flash decryption uses AES encryption
- During the power-up, the decryption process is performed
- BLK1 is “tweaked” with the offset address of each 32 bytes block of flash

```
burn_key flash_encryption encKey.bin
burn_efuse FLASH_CRYPT_CONFIG 0xf
burn_efuse FLASH_CRYPT_CNT
```

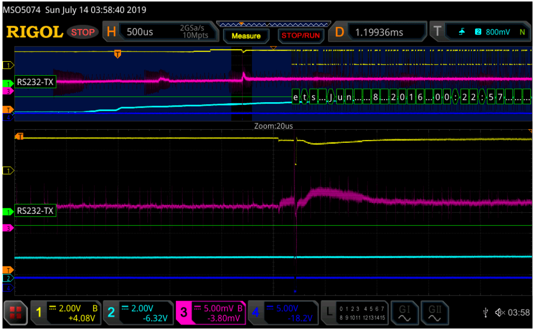


Flash decryption

LimitedResults attack

- eFuse protection bits are manipulated during the power-up
- Injecting faults using power glitching during the power-up can perturb these bits
- eFuse slots were attacked

```
Reset ESP32
ReadeFuse
```

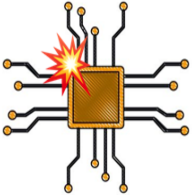


Source: LimitedResults

FAULT INJECTION SETUP

Fault attacks

- Perturbing the chip during sensitive operations
  - Secure boot [5]
  - Cryptographic operations (AES, DES, RSA, ...) [6]

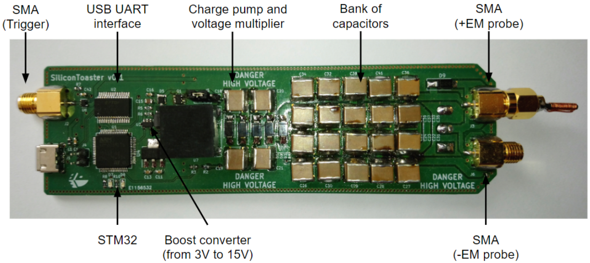


[5] Albert Spruyt and Niek Timmers, "Bypassing Secure Boot Using Fault Injection", Black Hat Europe 2016.

[6] Yifan Lu, "Attacking Hardware AES of PlayStation with DFA", 2019

Electromagnetic injection

- High voltage pulse is injected to the probe to create EMFI
- Localized faults
- Decapping the chip is not important (it depends)



SMA (Trigger)

USB UART interface

Charge pump and voltage multiplier

Bank of capacitors

SMA (+EM probe)

STM32

Boost converter (from 3V to 15V)

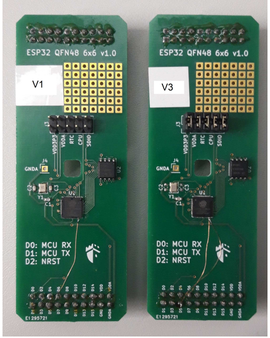
SMA (-EM probe)

EM Setup [7]

[7] Karim Abdellatif and Olivier Hériveaux , "SiliconToaster: A Cheap and Programmable EM Injector for Extracting Secrets", FDTC 2020.

A PCB for ESP32

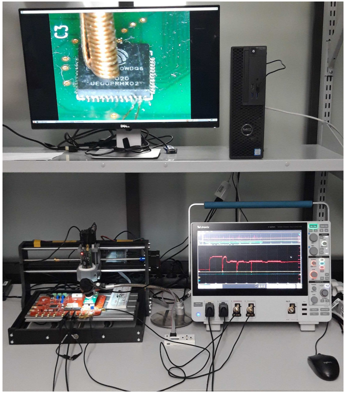
- For a stable setup, a PCB was fabricated
- ESP32 + external flash
- Several VDD pins are out to control
- An external oscillator



Fabricated PCB

Setup

- SiliconToaster for EM injection
- ESP32 on a scaffold[8] board
- An oscilloscope
- XYZ table



EM setup

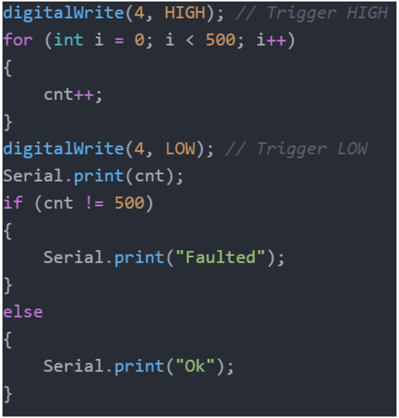
[8] Olivier Heriveaux, "https://github.com/Ledger-Donjon/scaffold"

Attack Plan

1. EM evaluation of ESP32-V1 using a glitchable application
2. Reproducing eFuse attack of LimitedResults by EM
3. EM evaluation of ESP32-V3 using a glitchable application
4. Performing eFuse attack on ESP32-V3

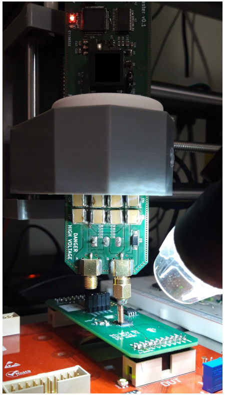
EMFI ON ESP32-V1

Glitchable application



```
digitalWrite(4, HIGH); // Trigger HIGH
for (int i = 0; i < 500; i++)
{
    cnt++;
}
digitalWrite(4, LOW); // Trigger LOW
Serial.print(cnt);
if (cnt != 500)
{
    Serial.print("Faulted");
}
else
{
    Serial.print("Ok");
}
```

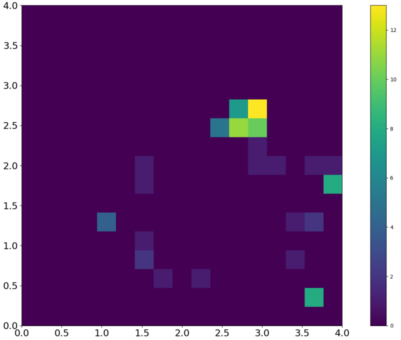
Glitchable code



EM probe scans the overall surface

Successful faults

- EM pulse = 500V
- Positive polarity
- 500 trials per spot
- Motor step = 0.2mm

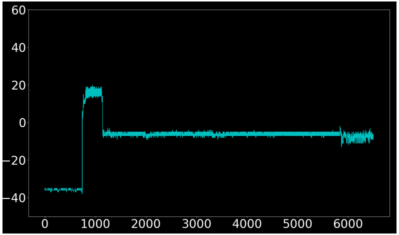


Vulnerable spots

After being sure from the setup settings, next step is to attack the eFuse slots.

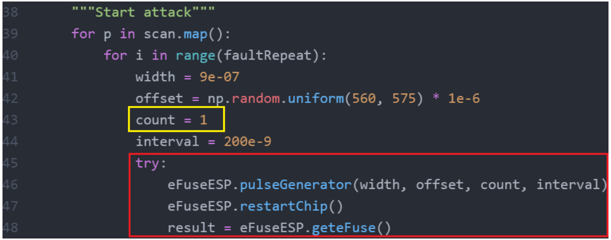
eFuse attack of ESP32-V1

```
burn_key flash_encryption encKey.bin
burn_efuse FLASH_CRYPT_CONFIG 0xf
burn_efuse FLASH_CRYPT_CNT
```



Power consumption during the power-up

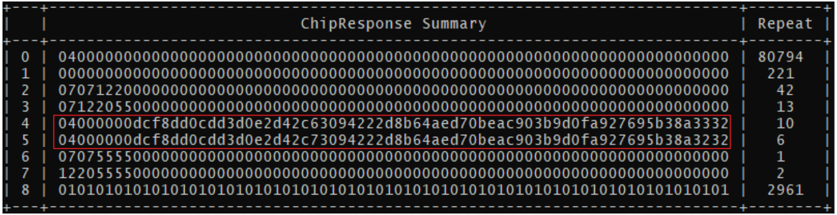
Attack scenario



```
    """Start attack"""
    for p in scan.map():
        for i in range(faultRepeat):
            width = 9e-07
            offset = np.random.uniform(560, 575) * 1e-6
            count = 1
            interval = 200e-9
            try:
                eFuseESP.pulseGenerator(width, offset, count, interval)
                eFuseESP.restartChip()
                result = eFuseESP.geteFuse()
```

Attack scenario

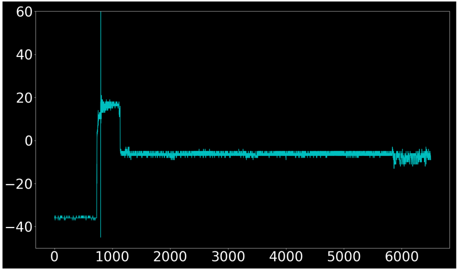
Successful faults



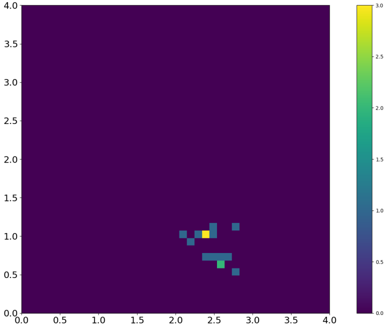
```
+---+--------------------------------------------------------------------------+---------+
|   |                           ChipResponse Summary                           | Repeat  |
+---+--------------------------------------------------------------------------+---------+
| 0 | 040000000000000000000000000000000000000000000000000000000000000000000000 | 80794   |
| 1 | 000000000000000000000000000000000000000000000000000000000000000000000000 |  221    |
| 2 | 070712200000000000000000000000000000000000000000000000000000000000000000 |   42    |
| 3 | 071220550000000000000000000000000000000000000000000000000000000000000000 |   13    |
| 4 | 04000000dcf8dd0cdd3d0e2d42c63094222d8b64aed70beac903b9d0fa927695b38a3332 |   10    |
| 5 | 04000000dcf8dd0cdd3d0e2d42c73094222d8b64aed70beac903b9d0fa927695b38a3232 |   6     |
| 6 | 070755550000000000000000000000000000000000000000000000000000000000000000 |   1     |
| 7 | 122055550000000000000000000000000000000000000000000000000000000000000000 |   2     |
| 8 | 010101010101010101010101010101010101010101010101010101010101010101010101 |  2961   |
+---+--------------------------------------------------------------------------+---------+
```

Experiment log

Successful faults



Power trace in case of a successful fault



Spots of eFuse successful attack

Discussion on ESP32-V1

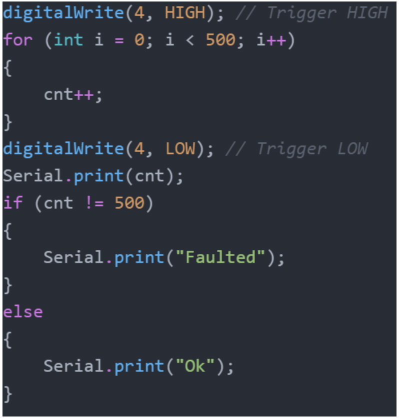
1. With EMFI, we managed to dump the eFuse slots of ESP32-V1
2. Only **ONE** single fault has been needed for this attack
3. The success rate is close to 0.6%

EMFI ON ESP32-V3

Recapping Espressif's countermeasures

1. New secure boot mechanism based on RSA
2. It is hardened against fault injection attacks in hardware and software as announced by the vendor
3. UART-disable to prevent eFuse reading command

Glitchable application



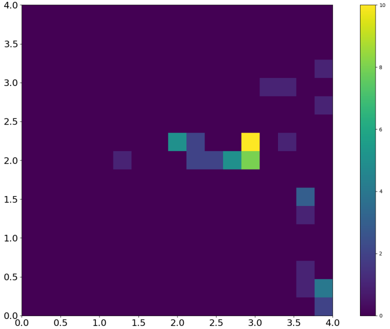
```
digitalWrite(4, HIGH); // Trigger HIGH
for (int i = 0; i < 500; i++)
{
    cnt++;
}
digitalWrite(4, LOW); // Trigger LOW
Serial.print(cnt);
if (cnt != 500)
{
    Serial.print("Faulted");
}
else
{
    Serial.print("Ok");
}
```

Glitchable code

EM probe scans the overall surface

Successful faults

- EM pulse = 500V
- Positive polarity
- 500 trials per spot
- Motor step = 0.2mm



Vulnerable spots

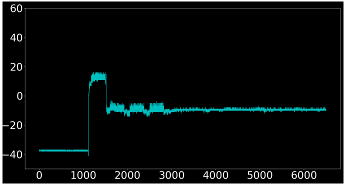
This confirms that ESP32-V3, is not hardened against fault injection attacks.

eFuse attack of ESP32-V3

```
burn_key flash_encryption encKey.bin
burn_efuse FLASH_CRYPT_CONFIG 0xf
burn_efuse FLASH_CRYPT_CNT
```

eFuse attack of ESP32-V3

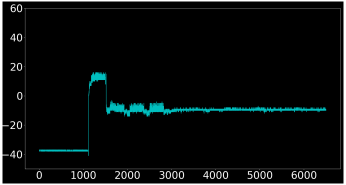
```
burn_key flash_encryption encKey.bin
burn_efuse FLASH_CRYPT_CONFIG 0xf
burn_efuse FLASH_CRYPT_CNT
```



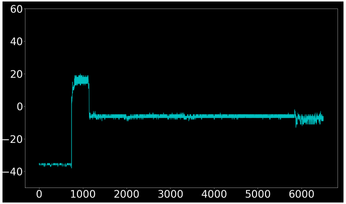
Power-up of ESP32-V3

eFuse attack of ESP32-V3

```
burn_key flash_encryption encKey.bin
burn_efuse FLASH_CRYPT_CONFIG 0xf
burn_efuse FLASH_CRYPT_CNT
```

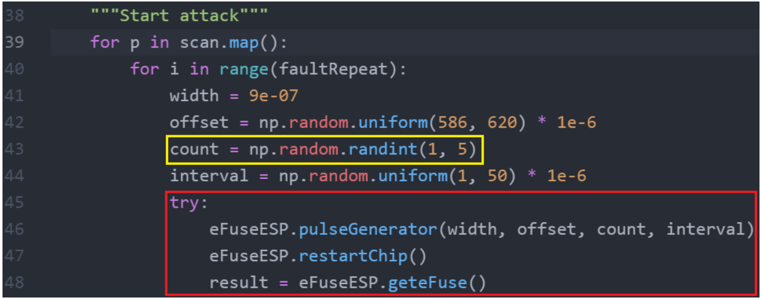


Power-up of ESP32-V3



Power-up of ESP32-V1

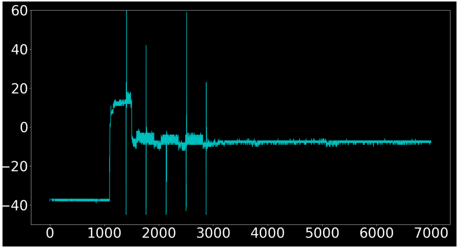
Attack plan



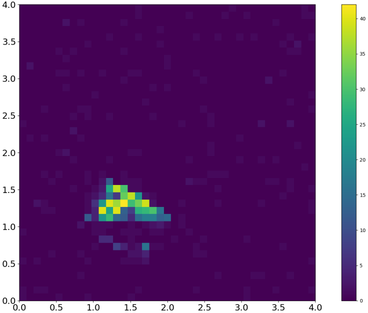
```
    """Start attack"""
    for p in scan.map():
        for i in range(faultRepeat):
            width = 9e-07
            offset = np.random.uniform(586, 620) * 1e-6
            count = np.random.randint(1, 5)
            interval = np.random.uniform(1, 50) * 1e-6
            try:
                eFuseESP.pulseGenerator(width, offset, count, interval)
                eFuseESP.restartChip()
                result = eFuseESP.geteFuse()
```

Multiple faults

Multiple Faults



Power trace in case of multiple faults



Spots of Timeout

The chip got crashed because of the multiple EM pulses.

Discussion on ESP32-V3

1. ESP32-V3 has a different boot ROM with countermeasures against fault injection
2. Multiple faults are needed
3. Until now, we haven't succeeded

BREAKING FIRMWARE ENCRYPTION BY SCAS

Moving to another attack path

- Motivation
  - A difficult attack using fault injection because of the boot ROM countermeasures
- Another attack path
  - A SCA on the flash encryption mechanism
    - Targeting the encryption process during the power up
    - Controlling the flash content to perform a CPA

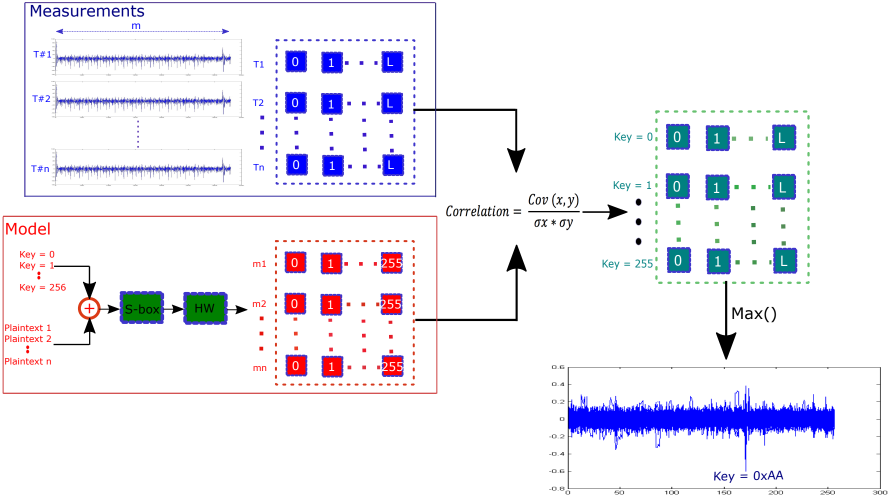
Leakage detection

- A methodology to identify leakage moments which contain sensitive information
- It reduces the computation complexity of security evaluation and improves the efficiency of the SCAs
- Several methods have been used to identify the amount of leakage such as SNR and NICV[9]

$$SNR = \frac{Var(E(x|y))}{E(Var(x|y))} \tag{2}$$

[9] S. Bhasin, J. Danger, and S. Guilley , "NICV: Normalized Inter-Class Variance for Detection of Side-Channel Leakage", SEC 2014

Correlation Power Analysis (CPA[10])

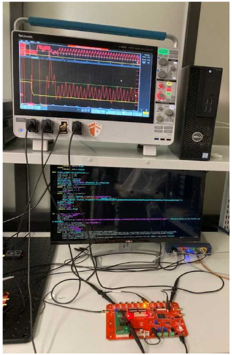


$$Correlation = \frac{Cov\,(x,y)}{\sigma x * \sigma y}$$

[10] E. Brier, C. Clavier, and F. Olivier , ”Correlation Power analysis with a leakage model”, CHES 2004

Side-channel attack setup

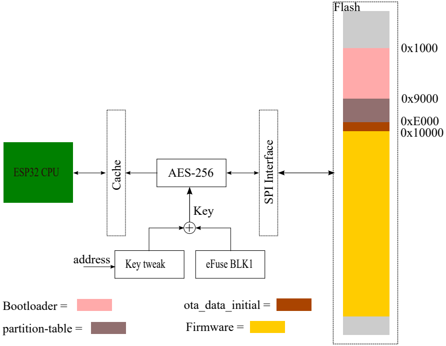
- High-end oscilloscope (6.25 Gs/s)
- ESP32 on a scaffold board
- Flash encryption has been enabled



SC setup

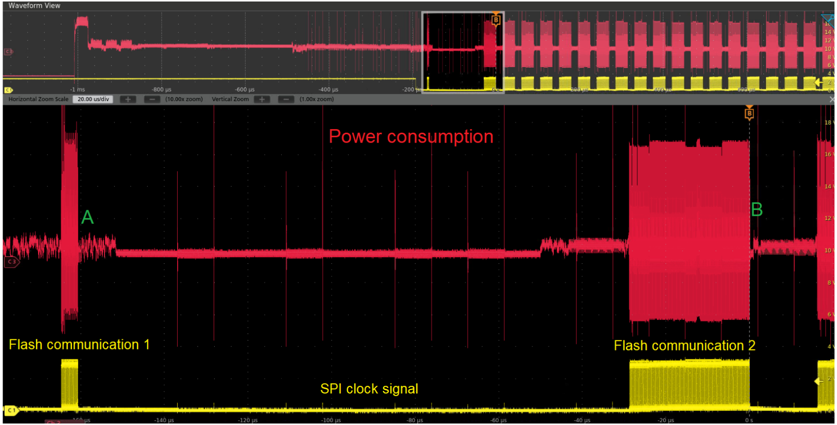
Flash encryption

- It encrypts all the flash content using AES-256 with BLK1 and stores it in the external memory
- During the power-up, the decryption process is performed
- First firmware part to get decrypted is the bootloader (stored at 0x1000)
- BLK1 is “tweaked” with the offset address of each 32 bytes block of flash



Flash decryption

Flash decryption during power-up



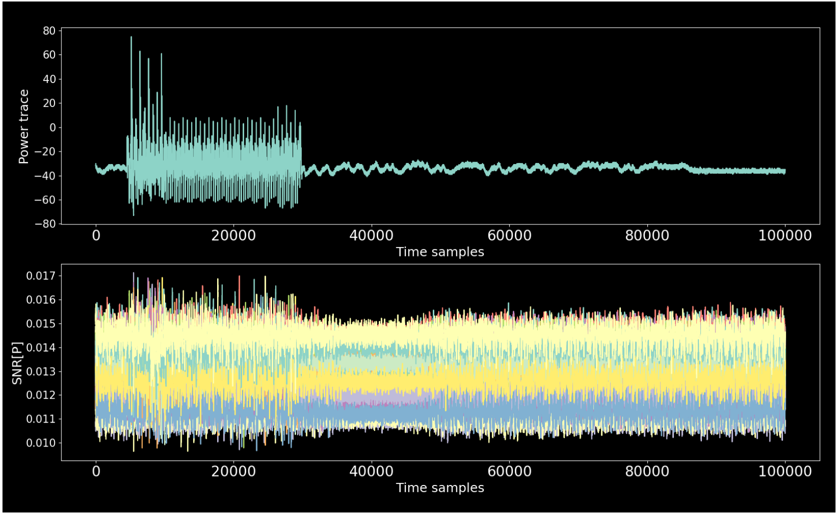
Power up with flash encryption

Scenario

Algorithm 1: Traces measurement sequence

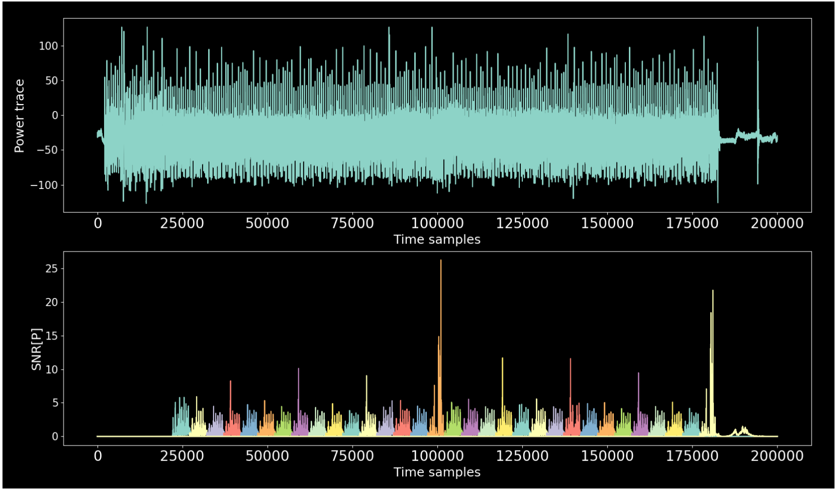
```
Data: N = No. traces = 100000
i = 0;
while True do
    FlashData = Random(32);
    EraseFlash();
    WriteFlash(FlashData,address = 0x1000);
    ChipRestart();
    CaptureTrace();
    i += 1;
    if (i == N) then
        break;
```

SNR on zone A



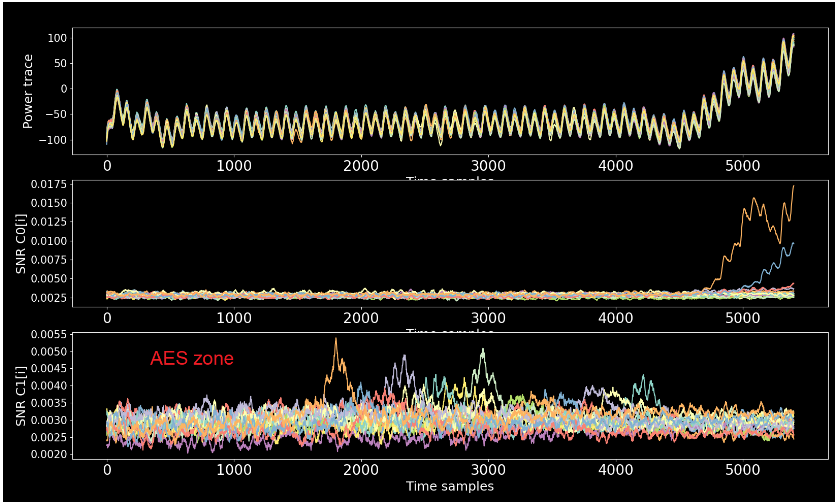
Power trace + SNR on zone A

SNR on zone B



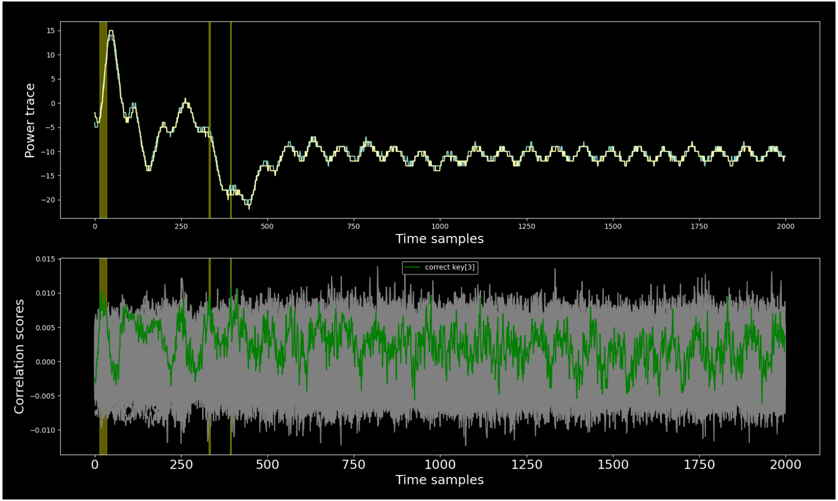
Power trace + SNR on zone B

SNR on Ciphertext



SNR on Ciphertexts

CPA results

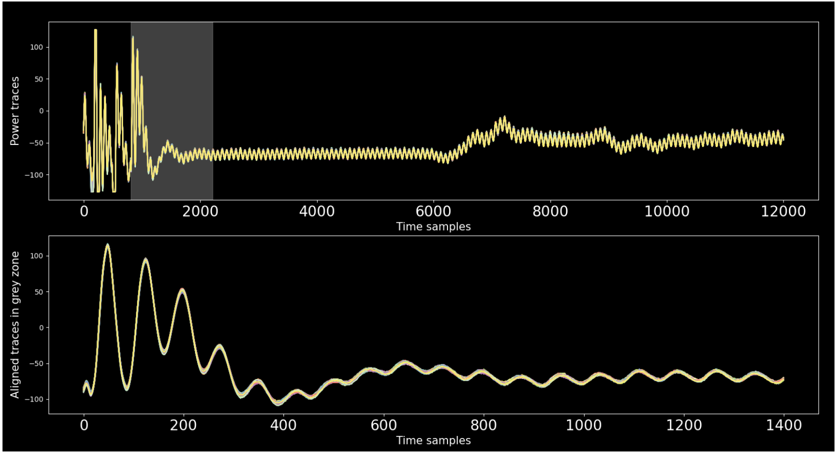


Correlation of Key[3] using 100K traces

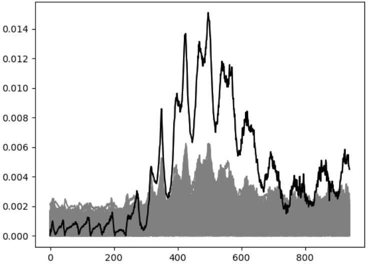
Flash limitations

1. The flash is limited in writing/erasing (around 110K times)
2. As a result, number of max traces $=$ 100K
3. Flash emulator was designed on scaffold

Power traces with flash emulator



CPA result

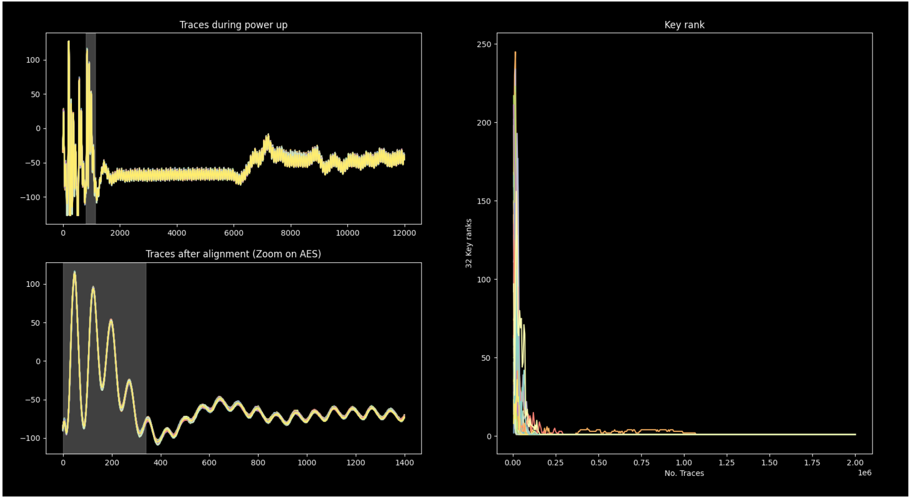


Correlation of Key[3] using 300K traces

$$Model_{round_0}[i] = HW(Sbox[P[i] \oplus guess]) \tag{3}$$

$$Model_{round_1}[i] = HW(Sbox[State_1[i] \oplus guess] \oplus Sbox[P[i] \oplus K[i]]) \tag{4}$$

Success rate

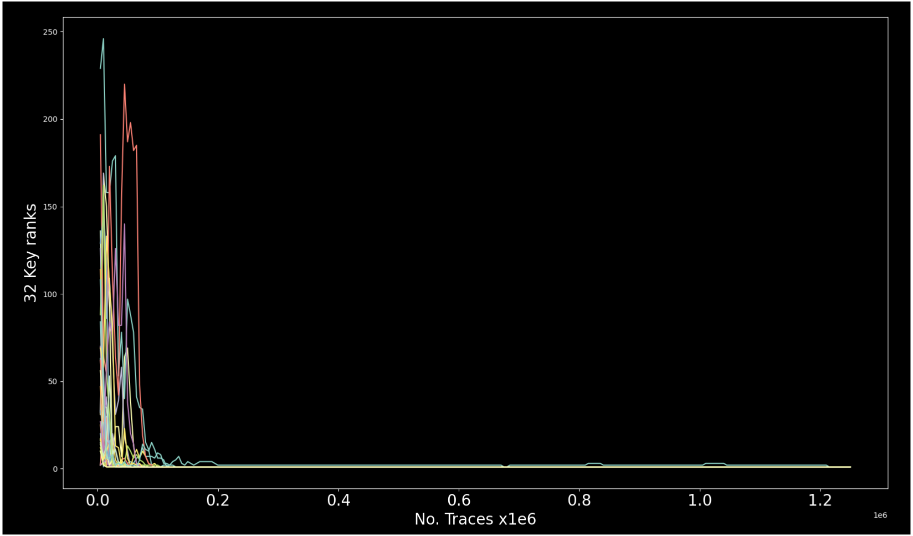


Success rate

Activating all security features

1. Secure boot
2. UART disable

Success rate

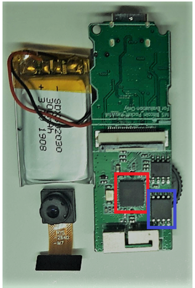


PRACTICAL ATTACK

Jade wallet

- Jade[11] is an open-source and open-hardware
- It doesn't store the user PIN in the external flash
- The PIN verification is performed remotely on the Blockstream's server by *blind_pin_server*[12]
- The external flash contains the user's private and public keys to communicate with this server

Jade wallet

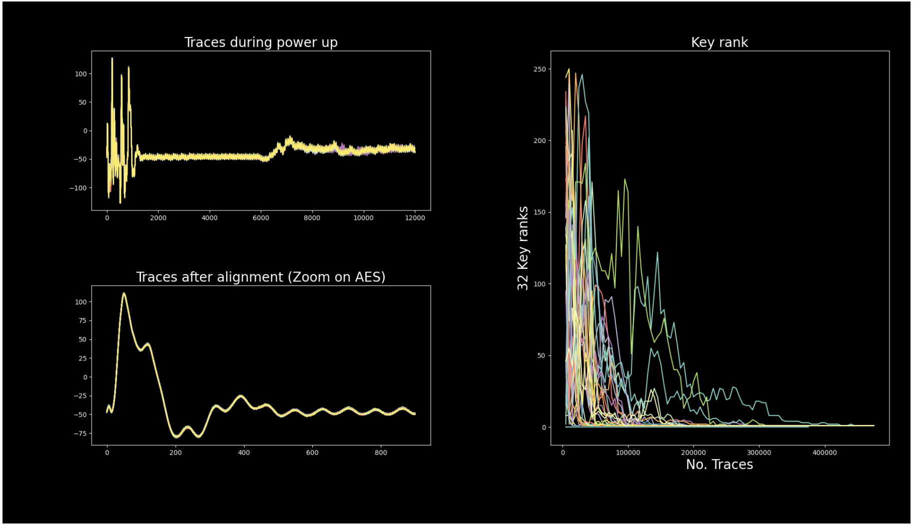


ESP32-V3 + external flash

[11] https://github.com/Blockstream/Jade

[12] https://github.com/Blockstream/*blind_pin_server*

Success rate

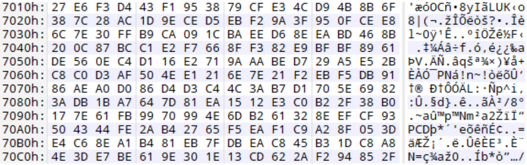


Traces during power up

Traces after alignment (Zoom on AES)

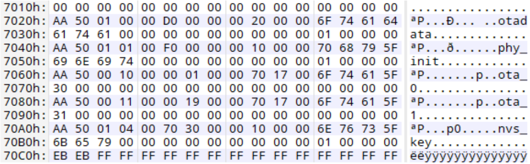
Key rank

Jade wallet



```
7010h: 27 E6 F3 D4 43 F1 95 38 79 CF E3 4C D9 4B 8B 6F  'æóÔCñ•8yÏãLÙK‹o
7020h: 38 7C 28 AC 1D 9E CE D5 EB F2 9A 3F 95 0F CE E8  8|(¬.žÎÕëòš?•.Îè
7030h: 6C 7E 30 FF B9 CA 09 1C BA EE D6 8E EA BD 46 8B  l~0ÿ¹Ê..ºîÖŽê½F‹
7040h: 20 0C 87 BC C1 E2 F7 66 8F F3 82 E9 BF BF 89 61   .‡¼Áâ÷f.ó‚é¿¿‰a
7050h: DE 56 0E C4 D1 16 E2 71 9A AA BE D7 29 A5 E5 2B  ÞV.ÄÑ.âqšª¾×)¥å+
7060h: C8 C0 D3 AF 50 4E E1 21 6E 7E 21 F2 EB F5 DB 91  ÈÀÓ¯PNá!n~!òëõÛ‘
7070h: 86 AE A0 D0 86 D4 D3 C4 4C 3A B7 D1 70 5E 69 82  †® Ð†ÔÓÄL:·Ñp^i‚
7080h: 3A DB 1B A7 64 7D 81 EA 15 12 E3 C0 B2 2F 38 B0  :Û.§d}.ê..ãÀ²/8°
7090h: 17 7E 61 FB 99 70 99 4E 6D B2 61 32 8E EF CF 93  .~aû™p™Nm²a2ŽïÏ“
70A0h: 50 43 44 FE 2A B4 27 65 F5 EA F1 C9 A2 8F 05 3D  PCDþ*´'eõêñÉ¢..=
70B0h: E4 C6 8E A1 B4 81 EB 7F DB EA C8 45 B3 1D C8 A8  äÆŽ¡´.ë.ÛêÈE³.È¨
70C0h: 4E 3D E7 BE 61 9E 30 1E 13 CD 62 2A F2 94 85 2F  N=ç¾až0..Íb*ò”…/
```

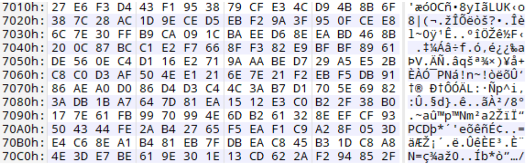
Encrypted firmware



```
7010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
7020h: AA 50 01 00 00 D0 00 00 00 20 00 00 6F 74 61 64  ªP...Ð... ..otad
7030h: 61 74 61 00 00 00 00 00 00 00 00 00 01 00 00 00  ata.............
7040h: AA 50 01 01 00 F0 00 00 00 10 00 00 70 68 79 5F  ªP...ð......phy_
7050h: 69 6E 69 74 00 00 00 00 00 00 00 00 01 00 00 00  init............
7060h: AA 50 00 10 00 00 01 00 00 70 17 00 6F 74 61 5F  ªP.......p..ota_
7070h: 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0...............
7080h: AA 50 00 11 00 00 19 00 00 70 17 00 6F 74 61 5F  ªP.......p..ota_
7090h: 31 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  1...............
70A0h: AA 50 01 04 00 70 30 00 00 10 00 00 6E 76 73 5F  ªP...p0.....nvs_
70B0h: 6B 65 79 00 00 00 00 00 00 00 00 00 01 00 00 00  key.............
70C0h: EB EB FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ëëÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

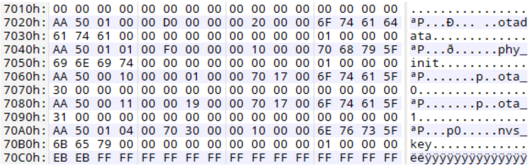
Decrypted firmware

Jade wallet



```
7010h: 27 E6 F3 D4 43 F1 95 38 79 CF E3 4C D9 4B 8B 6F 'æóÔCñ•8yÏãLUK‹o
7020h: 38 7C 28 AC 1D 9E CE D5 EB F2 9A 3F 95 0F CE E8 8|(¬.žÎÕëòš?•.Îè
7030h: 6C 7E 30 FF B9 CA 09 1C BA EE D6 8E EA BD 46 8B l~0ÿ¹Ê..ºîÖŽê½F‹
7040h: 20 0C 87 BC C1 E2 F7 66 8F F3 82 E9 BF BF 89 61  .‡¼Áâ÷f.ó‚é¿¿‰a
7050h: DE 56 0E C4 D1 16 E2 71 9A AA BE D7 29 A5 E5 2B ÞV.ÄÑ.âqšª¾×)¥å+
7060h: C8 C0 D3 AF 50 4E E1 21 6E 7E 21 F2 EB F5 DB 91 ÈÀÓ¯PNá!n~!òëõÛ'
7070h: 86 AE A0 D0 86 D4 D3 C4 4C 3A B7 D1 70 5E 69 82 †® Ð†ÔÓÄL:·Ñp^i‚
7080h: 3A DB 1B A7 64 7D 81 EA 15 12 E3 C0 B2 2F 38 B0 :Û.§d}.ê..ãÀ²/8°
7090h: 17 7E 61 FB 99 70 99 4E 6D B2 61 32 8E EF CF 93 .~aû™p™Nm²a2ŽïÏ"
70A0h: 50 43 44 FE 2A B4 27 65 F5 EA F1 C9 A2 8F 05 3D PCDþ*´'eõêñÉ¢..=
70B0h: E4 C6 8E A1 B4 81 EB 7F DB EA C8 45 B3 1D C8 A8 äÆŽ¡´.ë.ÛêÈE³.È¨
70C0h: 4E 3D E7 BE 61 9E 30 1E 13 CD 62 2A F2 94 85 2F N=ç¾až0..Íb*ò"…/
```

Encrypted firmware



```
7010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
7020h: AA 50 01 00 00 D0 00 00 00 20 00 00 6F 74 61 64 ªP...Ð... ..otad
7030h: 61 74 61 00 00 00 00 00 00 00 00 00 01 00 00 00 ata.............
7040h: AA 50 01 01 00 F0 00 00 00 10 00 00 70 68 79 5F ªP...ð......phy_
7050h: 69 6E 69 74 00 00 00 00 00 00 00 00 01 00 00 00 init............
7060h: AA 50 00 10 00 00 01 00 00 70 17 00 6F 74 61 5F ªP.......p..ota_
7070h: 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0...............
7080h: AA 50 00 11 00 00 19 00 00 70 17 00 6F 74 61 5F ªP.......p..ota_
7090h: 31 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 1...............
70A0h: AA 50 01 04 00 70 30 00 00 10 00 00 6E 76 73 5F ªP...p0.....nvs_
70B0h: 6B 65 79 00 00 00 00 00 00 00 00 00 01 00 00 00 key.............
70C0h: EB EB FF FF FF FF FF FF FF FF FF FF FF FF FF FF ëëÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

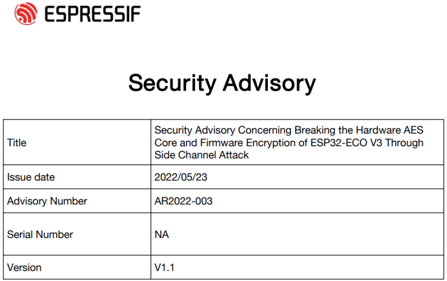
Decrypted firmware

Cloning the wallet + Injecting a backdoor to perform transactions to substituted addresses = evil maid attack

VENDOR REPLY AND CONCLUSION

Espressif's reply

- First e-mail was sent in October 2021
- **ESP32-S2, ESP32-C3 and ESP32-S3 are also impacted**
- Future products from Espressif **will** contain **countermeasures** against SCAs



ESPRESSIF

Security Advisory

Title	Security Advisory Concerning Breaking the Hardware AES Core and Firmware Encryption of ESP32-ECO V3 Through Side Channel Attack
Issue date	2022/05/23
Advisory Number	AR2022-003
Serial Number	NA
Version	V1.1

Espressif's advisory

Conclusion

- By experimental results, ESP32-V3 has a hardened boot ROM against fault injection (FI)

Conclusion

- By experimental results, ESP32-V3 has a hardened boot ROM against fault injection (FI)
- The presented side-channel attack is **generic** and works on all products based on all ESP32 versions (including V3)

Conclusion

- By experimental results, ESP32-V3 has a hardened boot ROM against fault injection (FI)
- The presented side-channel attack is **generic** and works on all products based on all ESP32 versions (including V3)
- Protection against fault injection (FI) **doesn't prevent side-channel attacks (SCAs)**

THANK YOU. QUESTIONS?

Karim M. Abdellatif, PhD
e-mail: karim.abdellatif@ledger.fr