raelize

HackInBo Spring 2023 Edition
20ª EDIZIONE

Acquisition in the billions: Breaking cryptographic keys with fast SCA

Niek Timmers
niek@raelize.com
@tieknimmers

Cristofaro Mune
cristofaro@raelize.com
@pulsoid

Introduction.

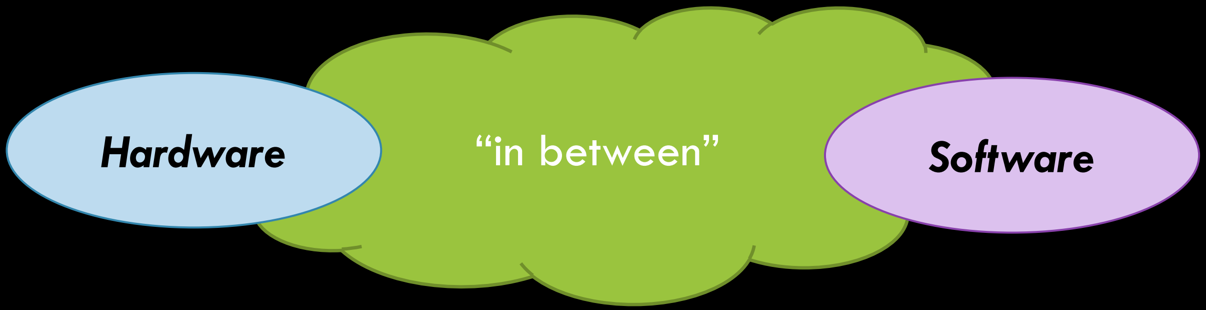
Me

Cristofaro Mune

- Co-Founder at Raelize; Security Researcher
- 20+ years in security
- 15+ years analyzing the security of complex systems and devices

rælize

- Based in The Netherlands. Specialized in Device Security
- Security testing, Consultancy and Training
- Low level software, hardware security:
  - Secure Boot, TEE, Fault injection,…



Our research: https://raelize.com/blog

Goals

- Outline usage of cryptographic keys in modern devices
- Introduce *side-channel-attacks (SCA)*
- Breaking AES via *power analysis*
  - on a modern *System-on-Chip (SoC)*
- Demonstrate techniques for fast acquisition
  - Billions of traces per day
- Reflect on implications

Raise awareness

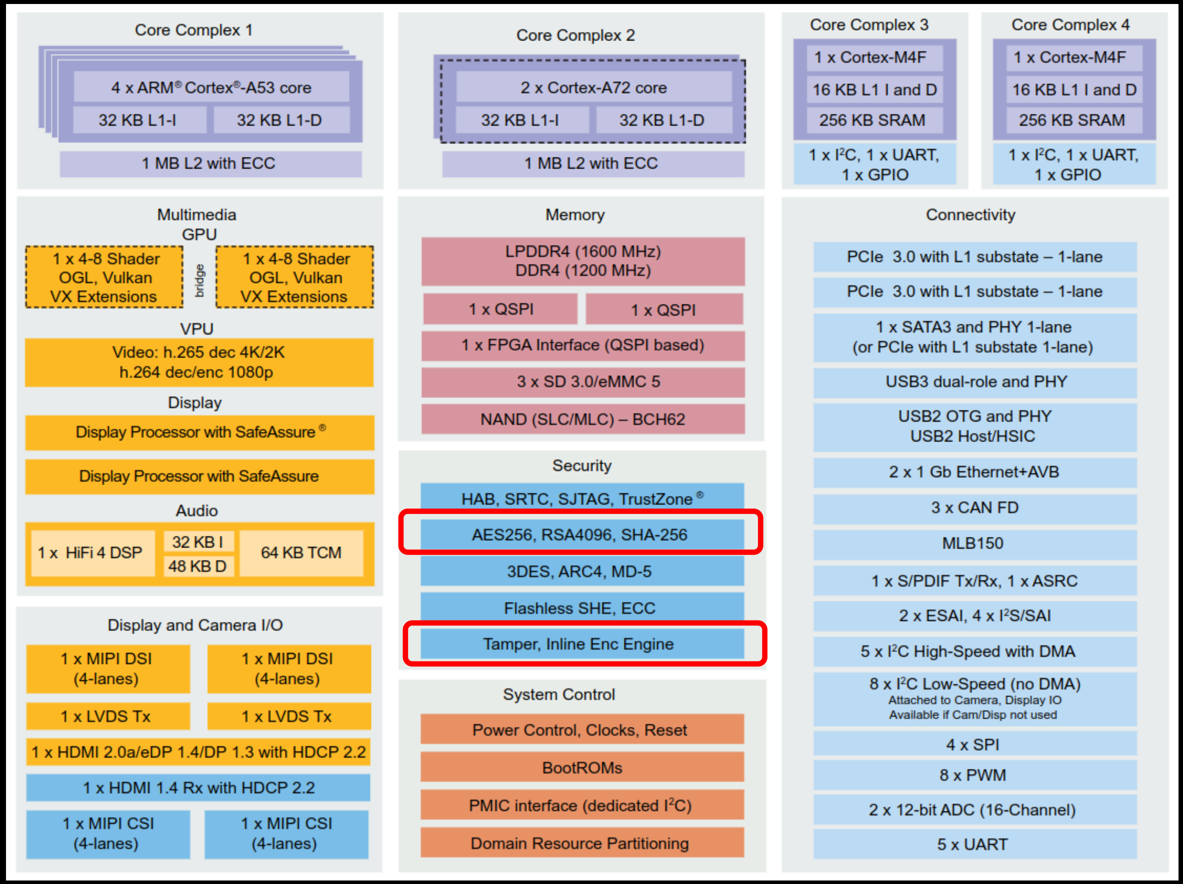
Of devices, keys and crypto(-graphy).

Devices: cryptographic operations

- Several designs (and implementations) available
- For our purposes, let's consider the following:
  - Pure software (SW):
    - Also in white-box cryptography (WBC) form
  - Hardware-assisted:
    - i.e. make use of hardware (HW) cryptographic accelerators

No intention of completeness!

Modern device: HW cryptographic accelerators

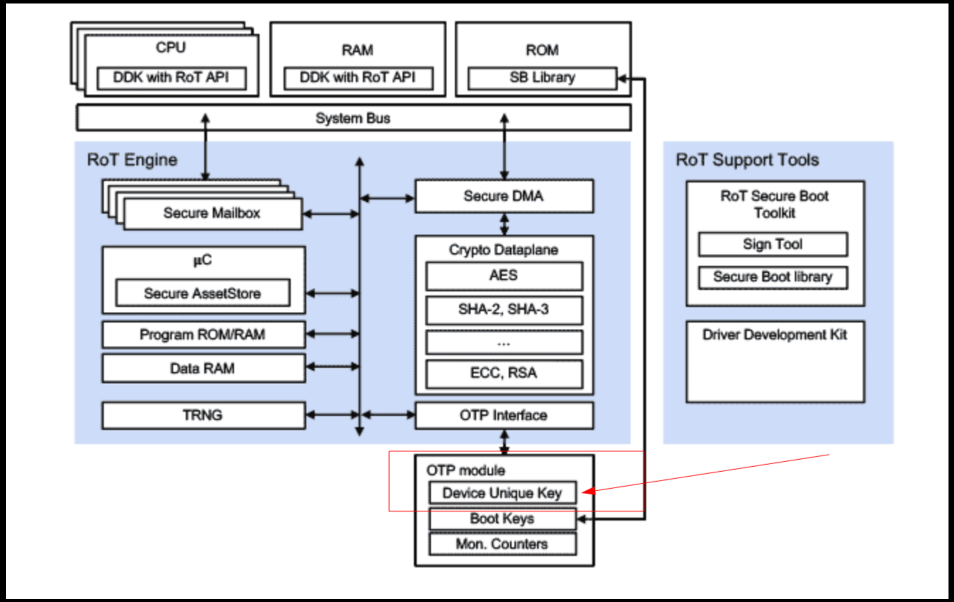


NXP i.MX8 SoC

Hack In Bo Spring 2023 Edition
20° EDIZIONE

Hardware: Keys inaccessible to SW

- Device Unique Key(s):
  - Stored in (e-)fuses or in the actual digital logic (rarely)
- Directly loaded in HW crypto engines slots
- No way for SW to read such keys:
  - No interface available



Rambus - RT-260 Root of Trust

storage systems, which protects data from a compromised "global" key. Device binding is based on a per-device unique identity which is baked into the hardware and cannot be altered by software after the devices are initialized.

never need to be accessed by software. This is necessary to enable device bound encryption as the device unique key can only be used on the device. In addition to hardware keys, software key set can be stored in the internal memory of

Qualcomm - Guard Your Data with the Qualcomm Snapdragon Mobile Platform (2019)

AES in brief

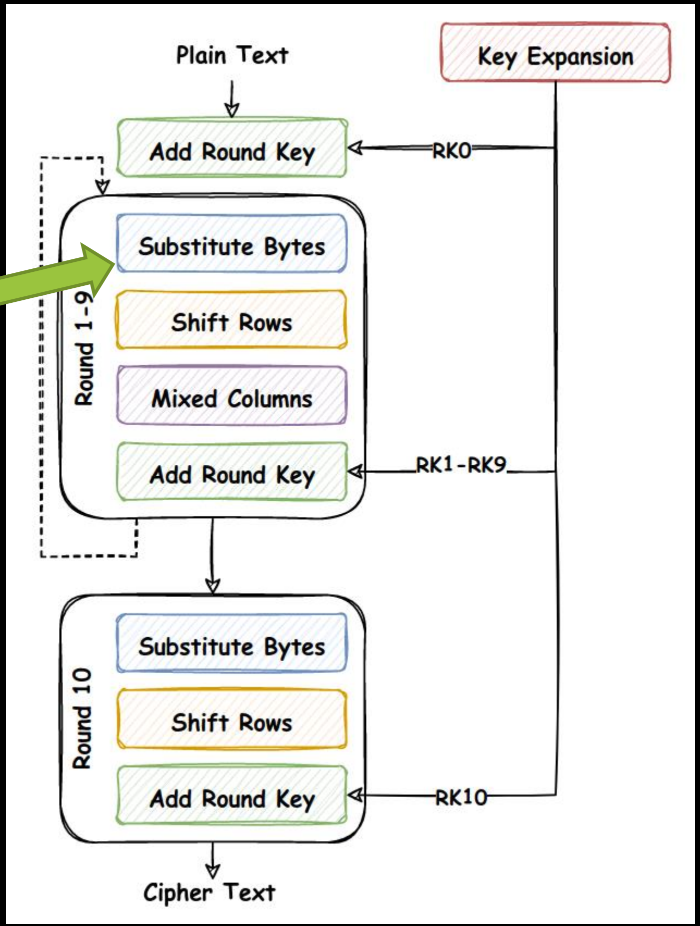
- Advanced Encryption Standard (AES)
  - FIPS PUB 197: Advanced Encryption Standard (AES)
  - ISO/IEC 18033-3: Block ciphers

- Features:
  - Symmetric cipher
  - Block cipher: 128 bits (regardless of key size)
  - Keys: 128, 192 and 256 bits
  - Number of rounds depends on key size (10, 12 or 14 rounds)

Algorithm (128-bit key, encryption)

- Key Expansion
  - avoid using the same key each round
- Add Round Key
  - state ⊕ RKn
- Substitute Bytes
  - apply S-Box to each byte of the state
- Shift Rows
  - Row bytes rotation
- Mix Columns

S-Box



A "gentle" intro to…
Side channel analysis (SCA)

HackInBo Spring 2023 Edition
20° EDIZIONE

“A side channel is some observable aspect of a system that reveals secrets within that system.”

– The Hardware Hacking Handbook

Inception: Bell 131-B2 (1943)

- Encrypted teletype used by U.S. Army and Navy:
  - One-time pads encryption
- Bell researcher noticed spikes in an oscilloscope nearby
- Disbelief: Is it really dangerous? Prove it!
  - Recovered 75% of plaintext from a different building (~25m away)
  - U.S. Army started clearing 30m perimeter
- Rediscovered in 1951
  - Recovered plaintext over power lines (400m away)

TEMPEST: A signal problem
NSA Declassified document

Power consumption

- Toggling data lines cause current spikes through VDD (+) and GND (-)

- Energy is a function of current flowing through circuit:
  - And so is Power!

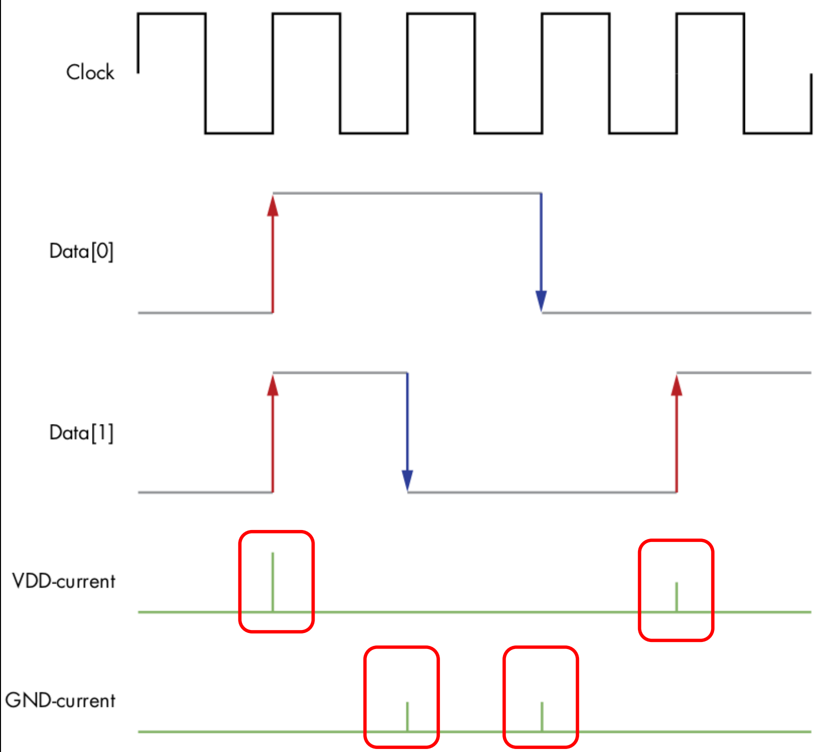


Figure 10-2: Monitoring current spikes when toggling data lines, showing current flowing in for both 0→1 and 1→0 transitions

Hardware Hacking Handbook
by Woudenberg & O'Flynn

Power leaks!

- Power consumption can leak information
  - E.g. during usage of (supposedly) secret data
-

  This includes intermediates of a cryptographic algorithm!

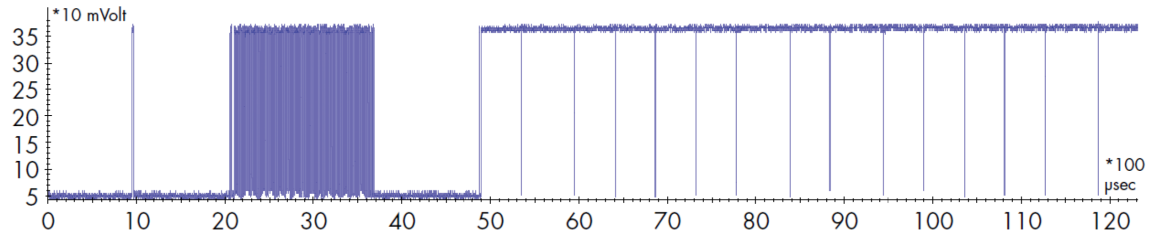


Figure 8-7: Power consumption trace of a square-and-multiply execution

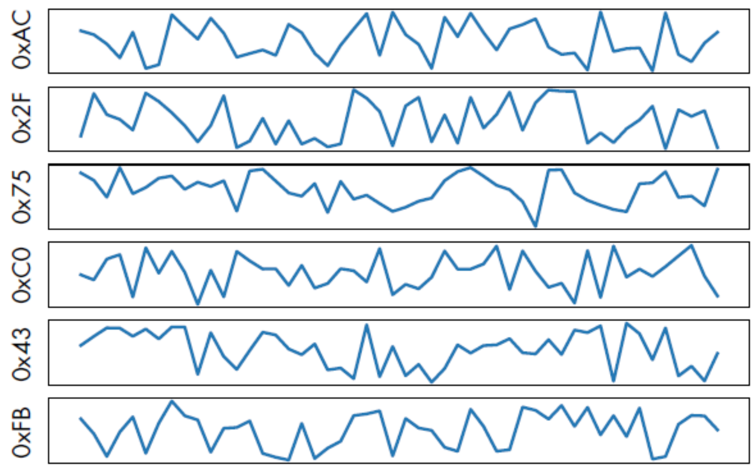


Figure 10-6: Input data vs. associated power trace

[Hardware Hacking Handbook](#)
by Woudenberg & O'Flynn

Electromagnetic field leaks too.

- Electromagnetic emissions can also leak information:
  - Program flow
  - Usage of (secret) data

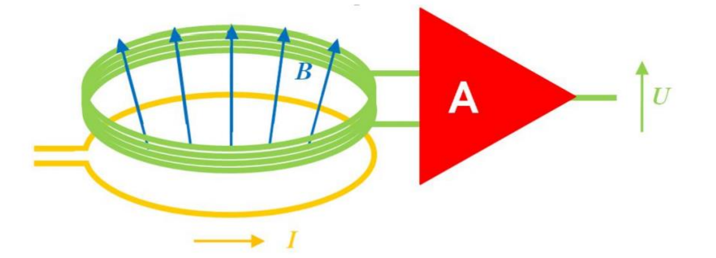


Fig. 4 Orientation of current loop (yellow) and sensor coil (green)

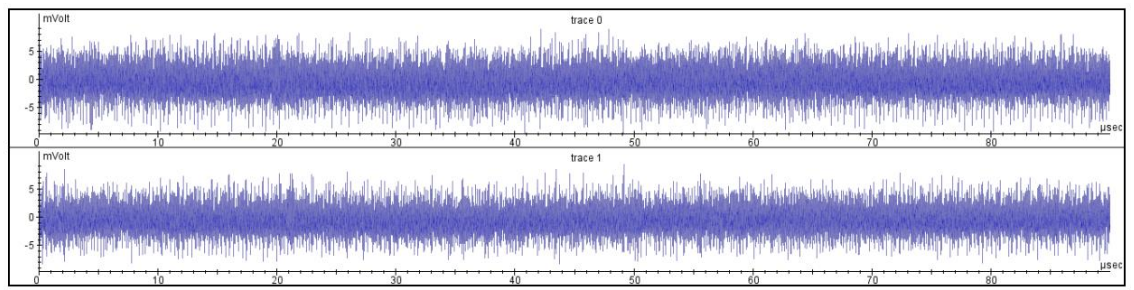


Fig. 12 Two unprocessed EM signal traces

Practical Electro-Magnetic Analysis
by Beer, Witteman, Gedrojc and Sheng

The challenge

- We have a device (target) performing AES encryptions
  - Using a HW cryptographic accelerator
- They key is hidden in HW
  - SW cannot access it
  - i.e. ANY code execution will not give you the key
- We can encrypt:
  - Whatever we want
  - As many times as we want

Can we recover the key?

Notes

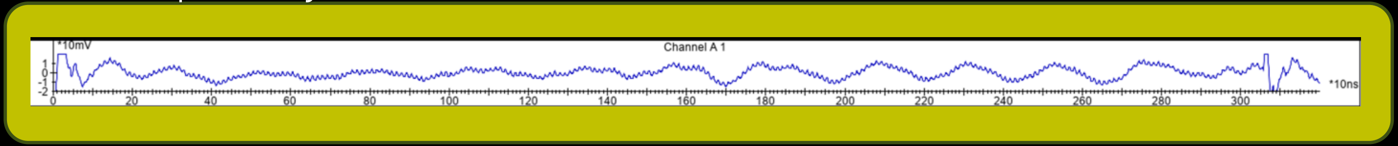
- Next slides provide a simplified overview of Differential Power Analysis (DPA):
  - Well…we only have 45m for this talk ☺
- DPA is a renowned SCA technique:
  - Used in many labs around the world for security evaluations
  - Supported by many academic papers and…
  - …many many keys extracted from real devices!

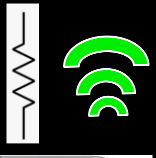
Feel free to ask for more info

The idea: Measure during encryption

Oscilloscope

Oscilloscope Memory



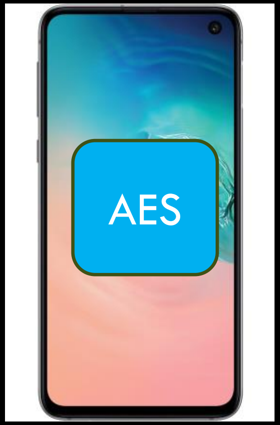


Plaintext

42424242424242424242424242424242

Cyphertext

31e33a6e5250909a7e518ce76d2c9f79

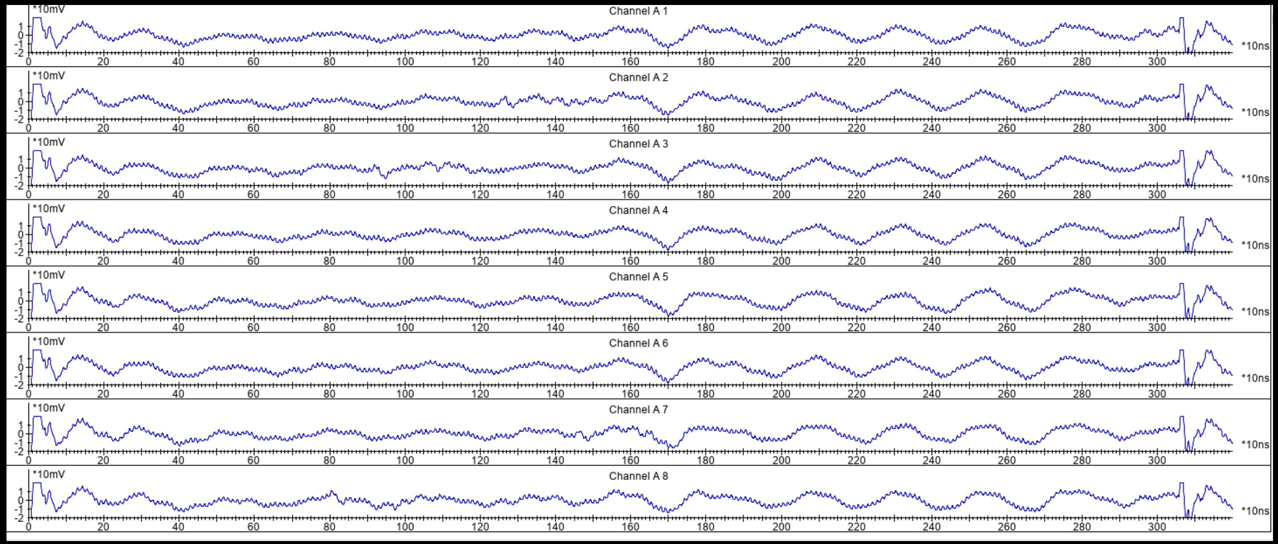


Key:

????????????????????????????????

The idea: Acquire MANY traces

- Execute the cryptographic algorithm a large number of times:
  - Vary the input randomly
- Acquire power traces:
  - while the cryptographic algorithm is being executed
- For each trace store:
  - The power profile
  - The input and the output (If available) data



Differential Power Analysis (DPA)

- Algorithm computes intermediate values:
  - Their actual value depends on the input and the key
  - Power profiles give information on the intermediate values
- Only one key can:
  - generate the right intermediates for all the input values
  - "match" the generated power consumption profiles

DPA: “Guessing the key”

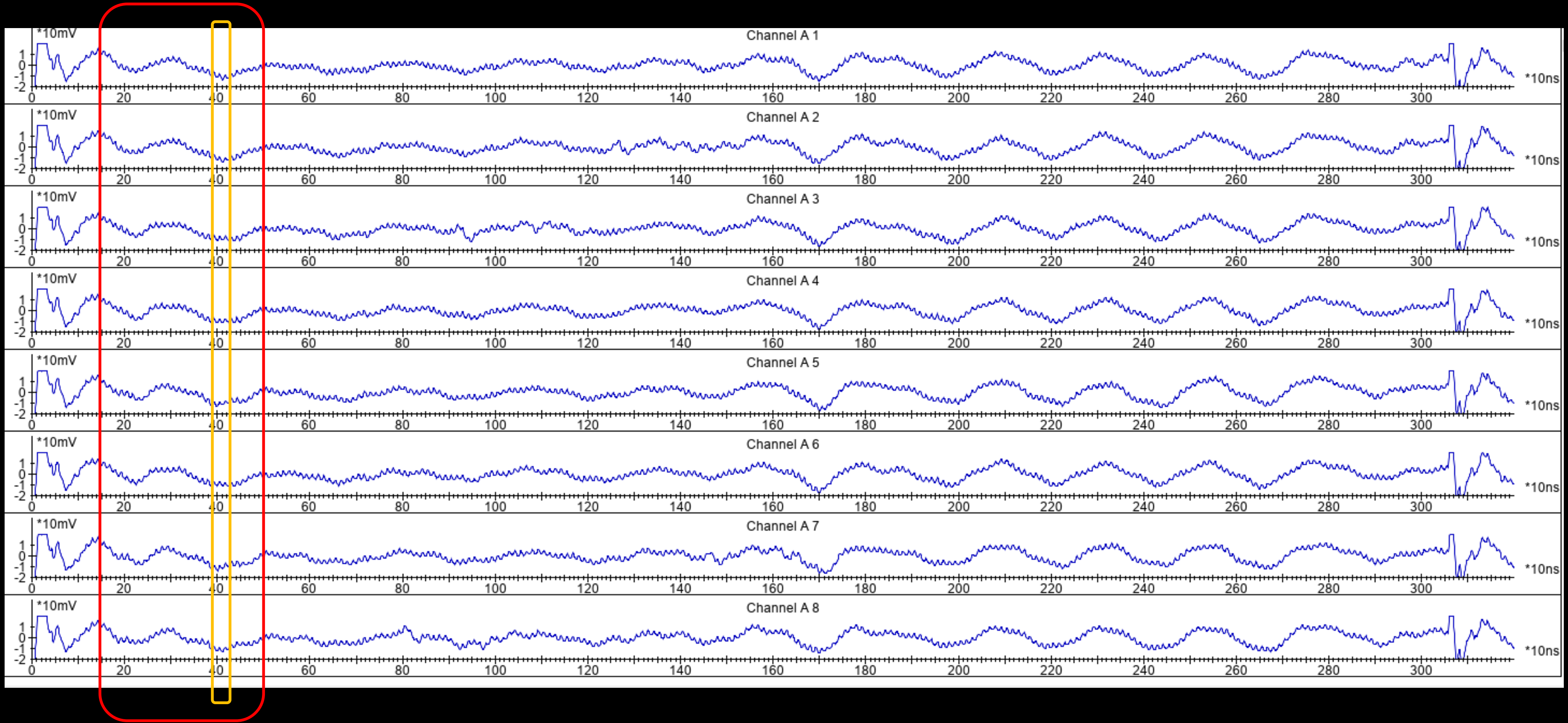
- Select a key candidate
- Compute all intermediates for the candidate key:
  - For each input value
- Look for a key whose intermediates can “match” all power profiles for all input values
- In practice:
  - Compute correlation (Pearson) between the intermediates and traces values (at each sample)
  - Correct key should exhibit highest correlation

The idea: Matching

Key A matches!

Key candidate A
R1SubBytes[0][0]

7b
ca
17
96
A4
03
68
fe



R1 SuBytes operation

Key candidate B
R1SubBytes[0][0]

6e
74
17
a1
7b
56
68
8a

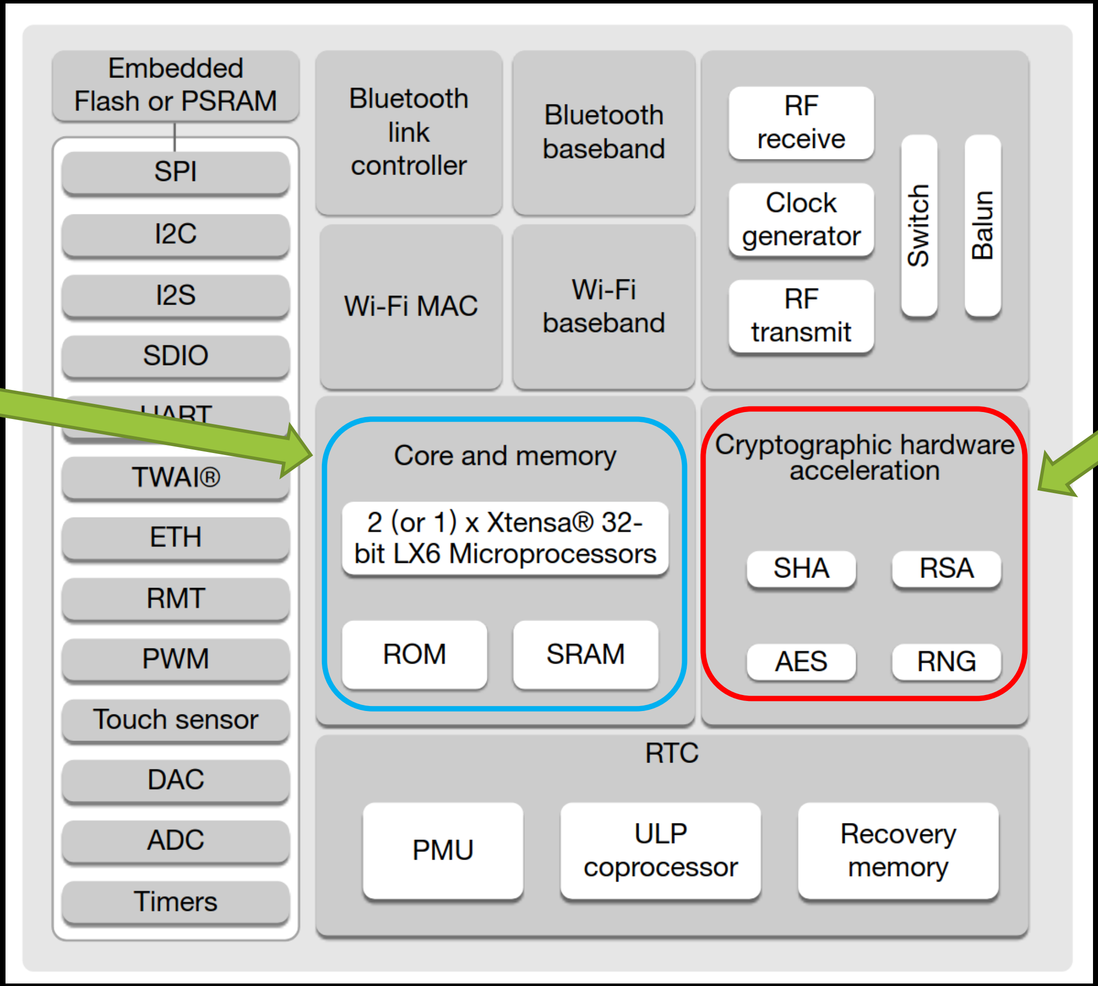
Our target: ESP32.

Espressif ESP32

ESP32-D0WDQ6

ESP32 SoC: diagram

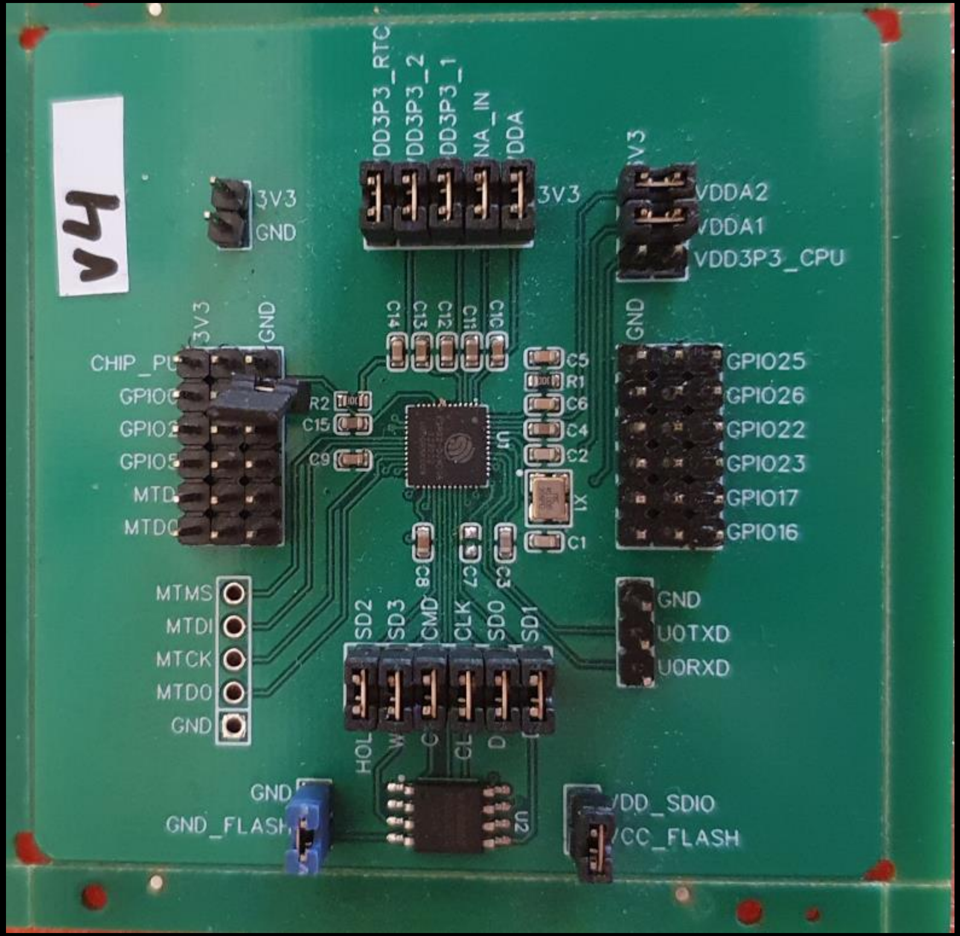
Execution units



Cryptographic hardware

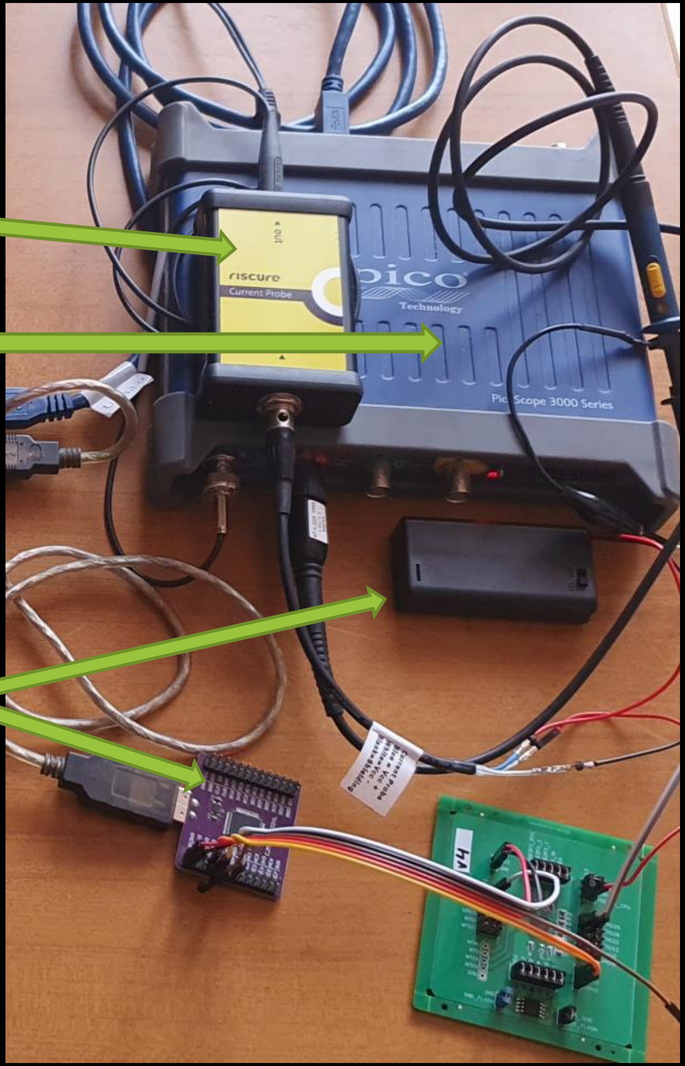
Our setup: Target

- Custom board:
  - Easier access to signals
  - Power CPU subsystem independently
- Application on target that can:
  - Set an arbitrary key
  - Operate the HW engine to perform encryptions/decryptions
  - Send a trigger to oscilloscope to start acquisition before encryption starts



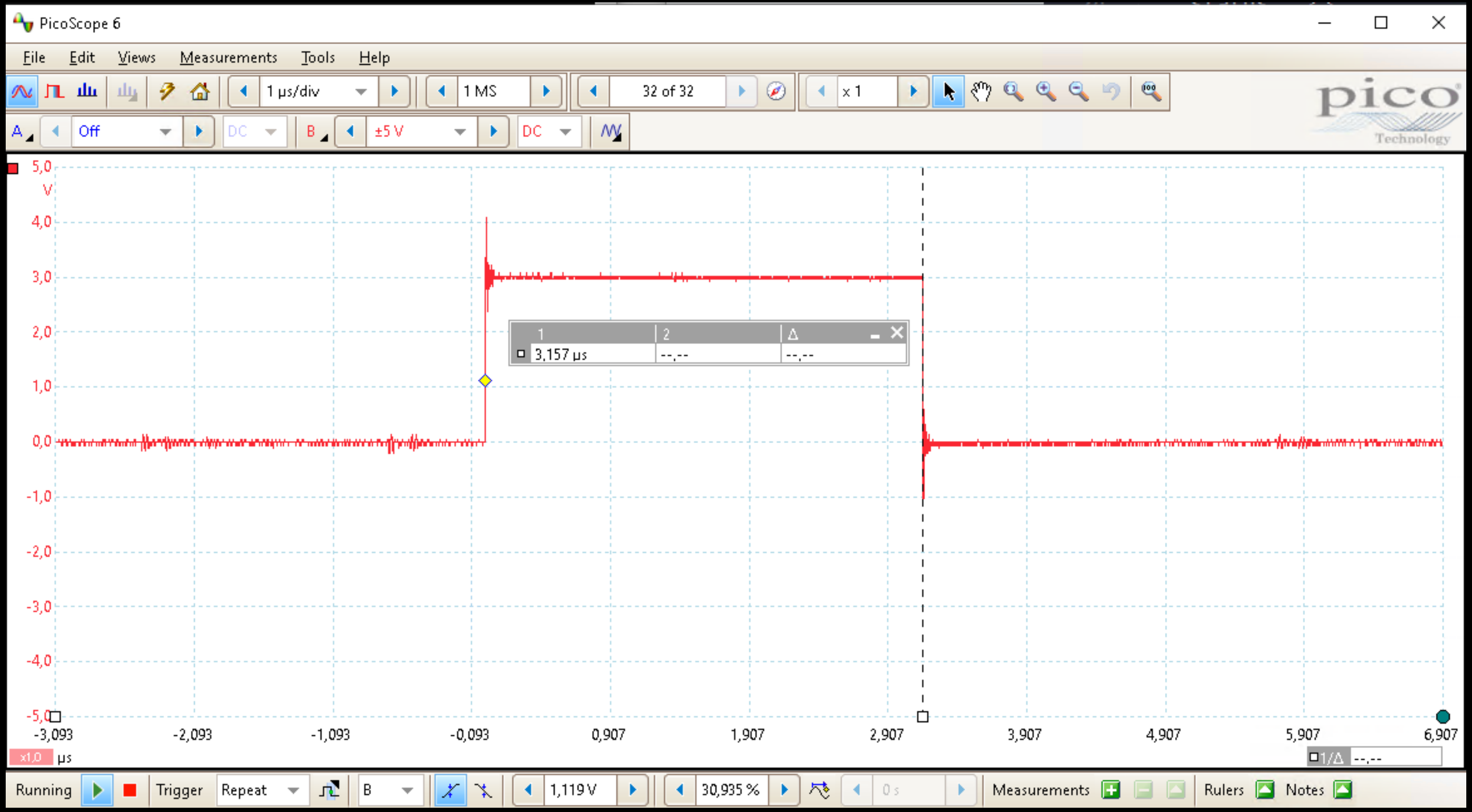
Our setup: Acquisition

- Riscure Current Probe:
  - For power measurements
- Picoscope 3406D:
  - Love that scope!
- FTDI 2232H:
  - Serial communications
    - For sending plaintext and receiving cypher text
  - Power the target: 3.3V
- A separate 3.3V battery package:
  - Cleaner measurements



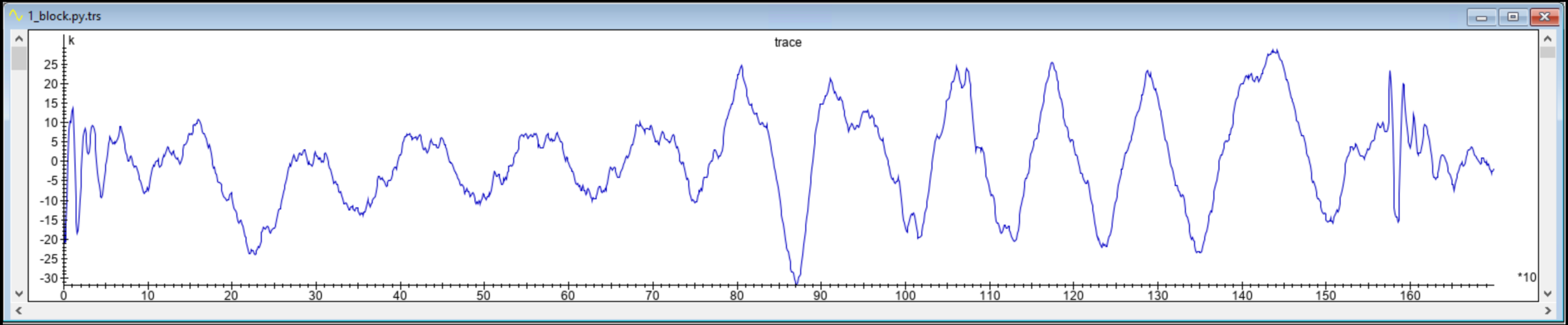
Reconnaissance and acquisition.

AES128 Encryption: Duration



- Time for one single encryption: 3.157 us

AES128 Encryption: Power profile



- Power profile does not show evident AES128 artifacts:
  - E.g. 10 repeated patterns (rounds)

Let’s collect more traces!

- 20000 traces
- 1700 samples at 500 Mbit/s:
  - i.e. We acquires 3.4us
- Acquisition time: 3m 02s
- Acquisition speed: ~9.47 Million traces/day

```
iterations = 1
nr_of_traces = 20000
delay = 0
nr_of_samples = 1700
```

```
Speed per day = 9.47E+06 (time: 0:03:02; traces: 19995)
Speed per day = 9.47E+06 (time: 0:03:02; traces: 19996)
Speed per day = 9.47E+06 (time: 0:03:02; traces: 19997)
Speed per day = 9.47E+06 (time: 0:03:02; traces: 19998)
Speed per day = 9.47E+06 (time: 0:03:02; traces: 19999)
Speed per day = 9.47E+06 (time: 0:03:02; traces: 20000)
20000
[Finished in 185.3s]
```

Can we get the key?

Considerations: Speed

- HW cryptographic engines can be fast:
  - Lower number of samples required (w.r.t SW implementations)
  - Operation completes in a shorter time (w.r.t SW implementations)

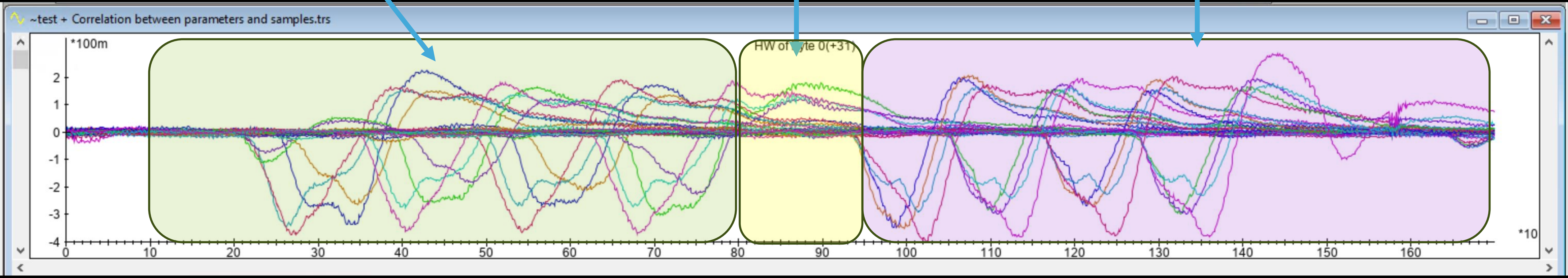
- Acquiring Millions of traces/day is not uncommon
  - Even with very simple setups

Input/Output correlation

Input correlation

Algorithm uses key here

Output correlation



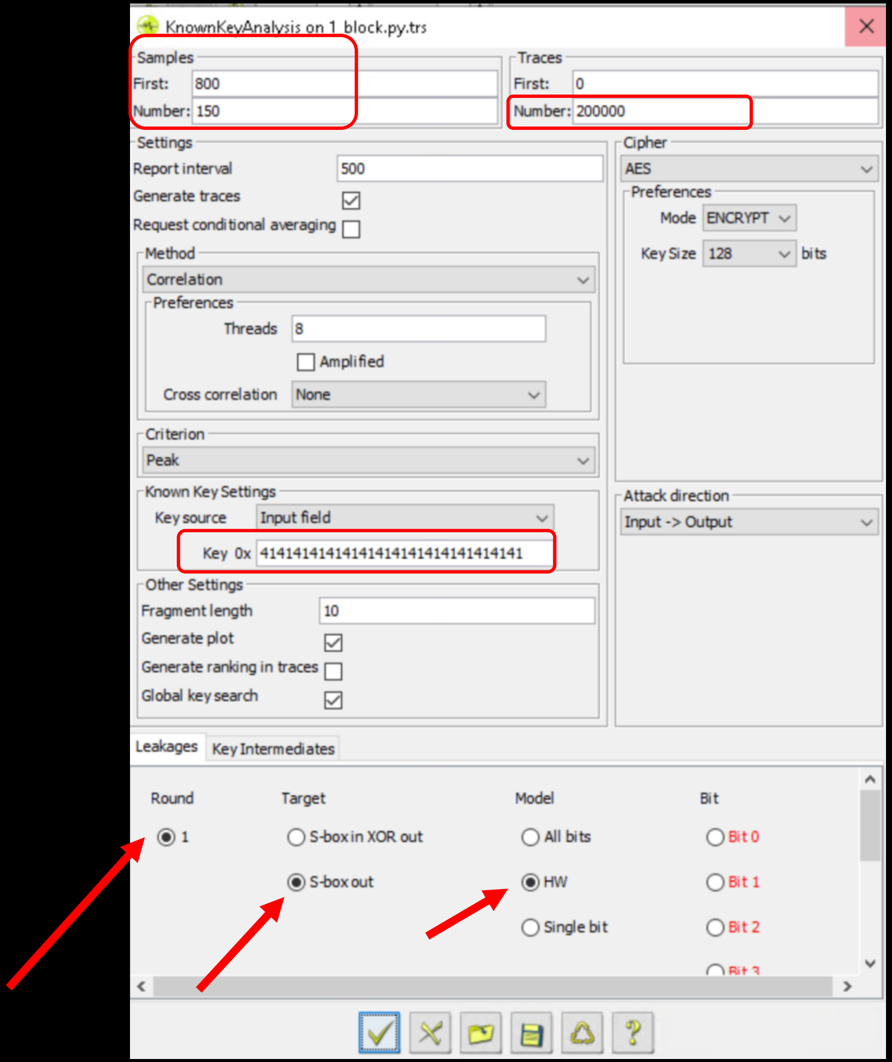
- Correlation with input (plaintext) and output (ciphertext) bytes:
  - Shows where such bytes are being "used"
- Key used:
  - after input is received
  - before output is generated
  - between samples 800 and 950

Known key analysis

- Take another device identical/similar to your target:
  - Same SoC
    Configuration as close as possible
  - You must control it (i.e. be able to set your key)
- Set your own key → You can computed intermediates (for every input)
- Perform correlation analysis with power profiles
- You will get:
  - If the SoC leaks information
  - Where information leakage happens

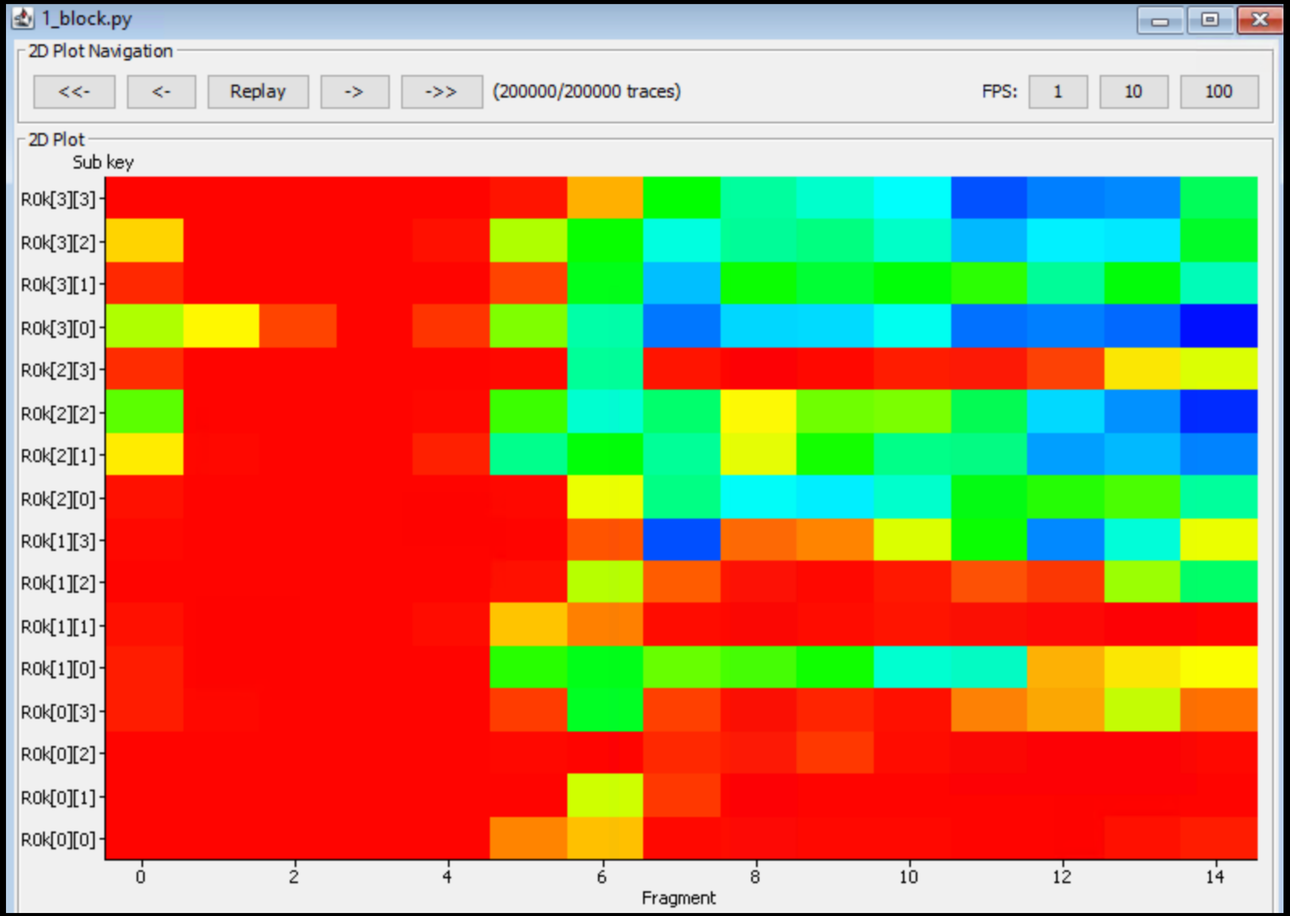
Known key analysis: Settings

- Performed on 200k traces:
  - Acquired in 30-40m
- Focus only between samples 800 and 950
- Leakage model:
  - Hamming weight on S-box output



Known key analysis: Results

- Leakage for all key bytes:
  - Samples: 820 → 840



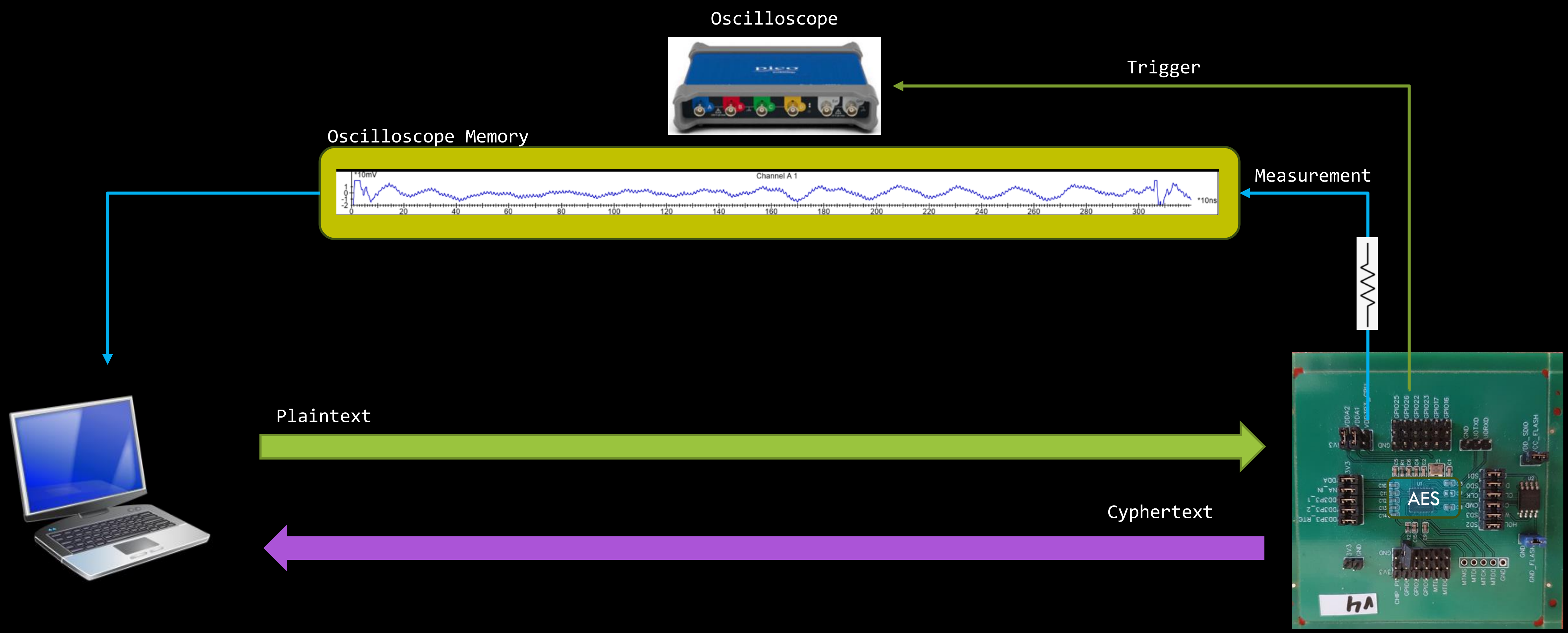
Success!

```
Correct key found: 41414141414141414141414141414141
```

- Key can be retrieved in ~40m:
  - 200k traces
  - Acquisition time: ~30m
  - Acquisition speed: ~9.4M traces/day

Can we go faster?
Segmented memory.

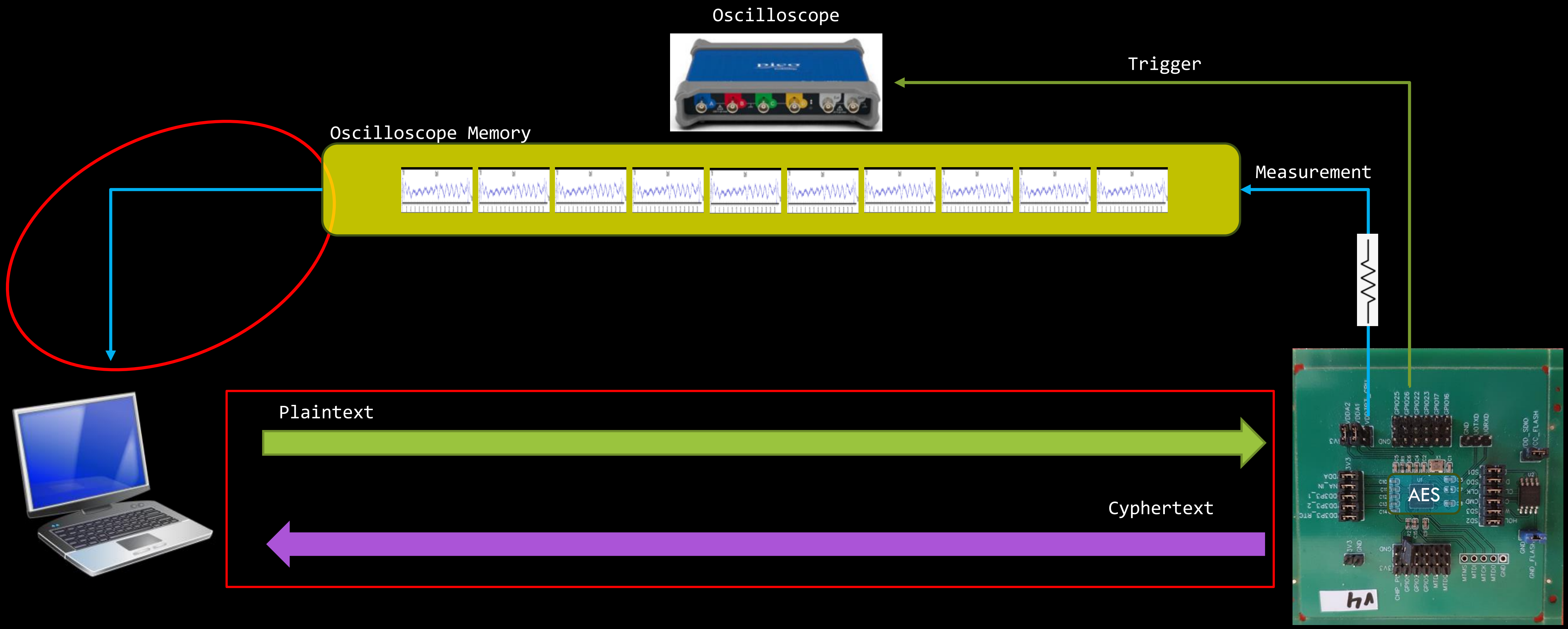
Acquisition cycle



Segmented Memory

- Feature available on many modern oscilloscopes
- Scope internal memory can be "segmented" to store multiple traces
  - Number limited by scope memory size
- Acquired traces are sent to PC in one single bundle
- Typical usage:
  - Perform multiple measurements with the same input
  - Traces can be averaged to reduce noise

Acquisition: Segmented memory



Segmented memory + averaged traces

- 20000 traces:
  - 100 traces in segmented memory (iterations)
  - 2000 input provided
- 150 samples
- Acquisition time: 25s
- Acquisition speed: ~675 Million traces/day

```
iterations = 100
nr_of_traces = 2000
delay = 800
nr_of_samples = 150
```

```
Speed per day = 6.75E+08 (time: 0:00:25; traces: 199200)
Speed per day = 6.75E+08 (time: 0:00:25; traces: 199300)
Speed per day = 6.75E+08 (time: 0:00:25; traces: 199400)
Speed per day = 6.75E+08 (time: 0:00:25; traces: 199500)
Speed per day = 6.75E+08 (time: 0:00:25; traces: 199600)
Speed per day = 6.75E+08 (time: 0:00:25; traces: 199700)
Speed per day = 6.75E+08 (time: 0:00:25; traces: 199800)
Speed per day = 6.75E+08 (time: 0:00:25; traces: 199900)
Speed per day = 6.75E+08 (time: 0:00:25; traces: 200000)
Traces in traceset: 2000
[Finished in 28.5s]
```

Can we get the key?

Nope.

- Input not sufficiently diversified
  - Only 2000 plaintexts

- No sufficient leakage to reveal key:
  - On this specific target

- How can we:
  - Have sufficiently diversified input AND
  - Minimize communication overhead with target

?

Speedy Gonzalez: On-target input generation.

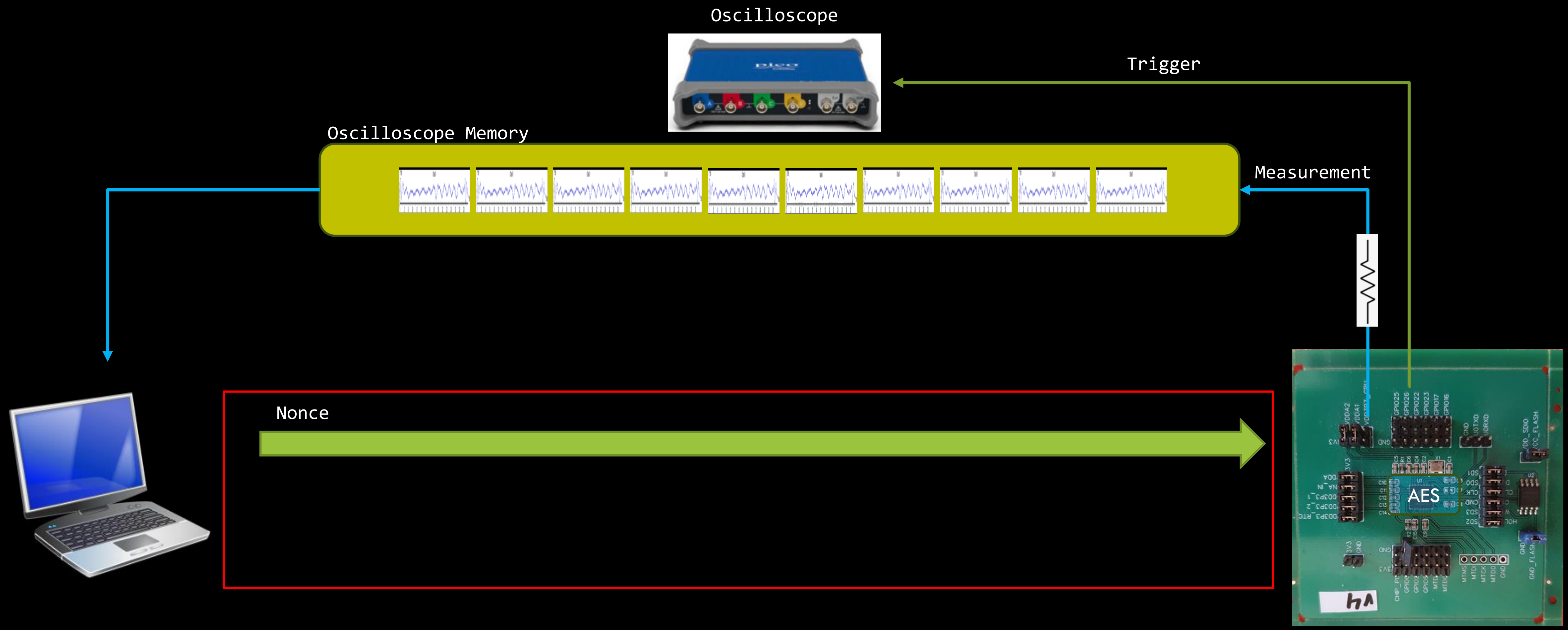
Generating input on target

- No need to provide input from PC
  - It's only sufficient to KNOW the input to the AES engine for each encryption operation
  - Output not needed
    - We are attacking encryption → AES round 1

- General idea:
  - Only send an initial nonce
  - Use a cryptographic function to generate next input
  - Apply the same function on the PC side to compute the same input

- Examples:
  - Recursively apply a Hash function to nonce
  - Use AES engine output as input for next operation

Acquisition: On-target input generation



How fast can we go?

- 20000 traces:
  - 50000 traces in segmented memory (iterations)
  - 4 bulk transfers to PC
- 1 sample: (nr. 820)
- Acquisition time: 12s
- Acquisition speed: ~2 Billion traces/day

```
iterations = 50000
nr_of_traces = 4
delay = 820
nr_of_samples = 1
```

```
Speed per day = 1.78E+09 (time: 0:00:02; traces: 50000)
Speed per day = 1.92E+09 (time: 0:00:04; traces: 100000)
Speed per day = 1.97E+09 (time: 0:00:06; traces: 150000)
Speed per day = 2.00E+09 (time: 0:00:08; traces: 200000)
Traces in traceset: 200000
[Finished in 11.9s]
```

Can we get the key now?

Yes!

```
Correct key found: 41414141414141414141414141414141
```

- Key can be retrieved in ~1.5m:
  - 200k traces
  - Acquisition time: ~12s
  - Acquisition speed: ~2B traces/day

Notes

- Even faster acquisition speed may be possible with further tuning

- We can now retrieve a key from ESP32:
  - Used by the HW crypto engine
  - By means of power analysis
  - Using segmented memory
  - Generating input on target
  - Using Jlsca for analysis
- In less than 25s.

Demo.

Technique is known and used!

- Research:
  - Leakage Assessment Methodology - a clear roadmap for side-channel evaluations - Schneider et. Al
  - A flexible leakage trace collection setup for arbitrary cryptographic IP cores - Moschos et. al.
  - Apple vs. EMA: Electromagnetic Side Channel Attacks on Apple CoreCrypto - Haas et. al.
  - Using a magic wand to break the iPhone's last security barrier – tihmstar
- Also security/evaluation labs are (likely) using it ☺

Back to base: Conclusions.

Summary

- Acquisition speed of millions of traces/day are common
- Billions of traces/day can be achieved:
  - Under specific conditions
  - Some degree of target control is required
- Technique is known, described in literature and actively used

Implications

- Claims of “resistance to SCA” should consider acquisition speeds in the billions of traces/day:
  - Evaluation/security labs
  - Certification schemes

- Very fast attacks may be possible in some specific scenarios:
  - E.g. when access to target is time constrained

Thank you!

Niek Timmers
niek@raelize.com
@tieknimmers

Cristofaro Mune
cristofaro@raelize.com
@pulsoid