Using a magic wand to break the iPhone's last security barrier

tihmstar

Target device

- iPhone 4 (A4 CPU)
- Released June 2010
- Vulnerable to limera1n exploit
  (gives highest possible privileges)
- No dedicated security co-processors
- Hardware AES engine
  - Hardware fused keys (allows setting own keys)
  - Oracle access to enc/dec in AES CBC mode

Motivation

- Extract model specific GID key
  - Allows decrypting firmware without physical device
    - No benefits for devices with BootRom exploit
    - Very useful if BootRom code-exec was achieved on modern devices though glitching!

Motivation

- Extract device specific UID key
  - Allow scalable offline cracking of passcode
    - Passcode decrypts userdata
    - Attack benefits outweighs effort for passcode which are more complex than 8 digit numeric
      (Otherwise you might as well crack on-device)

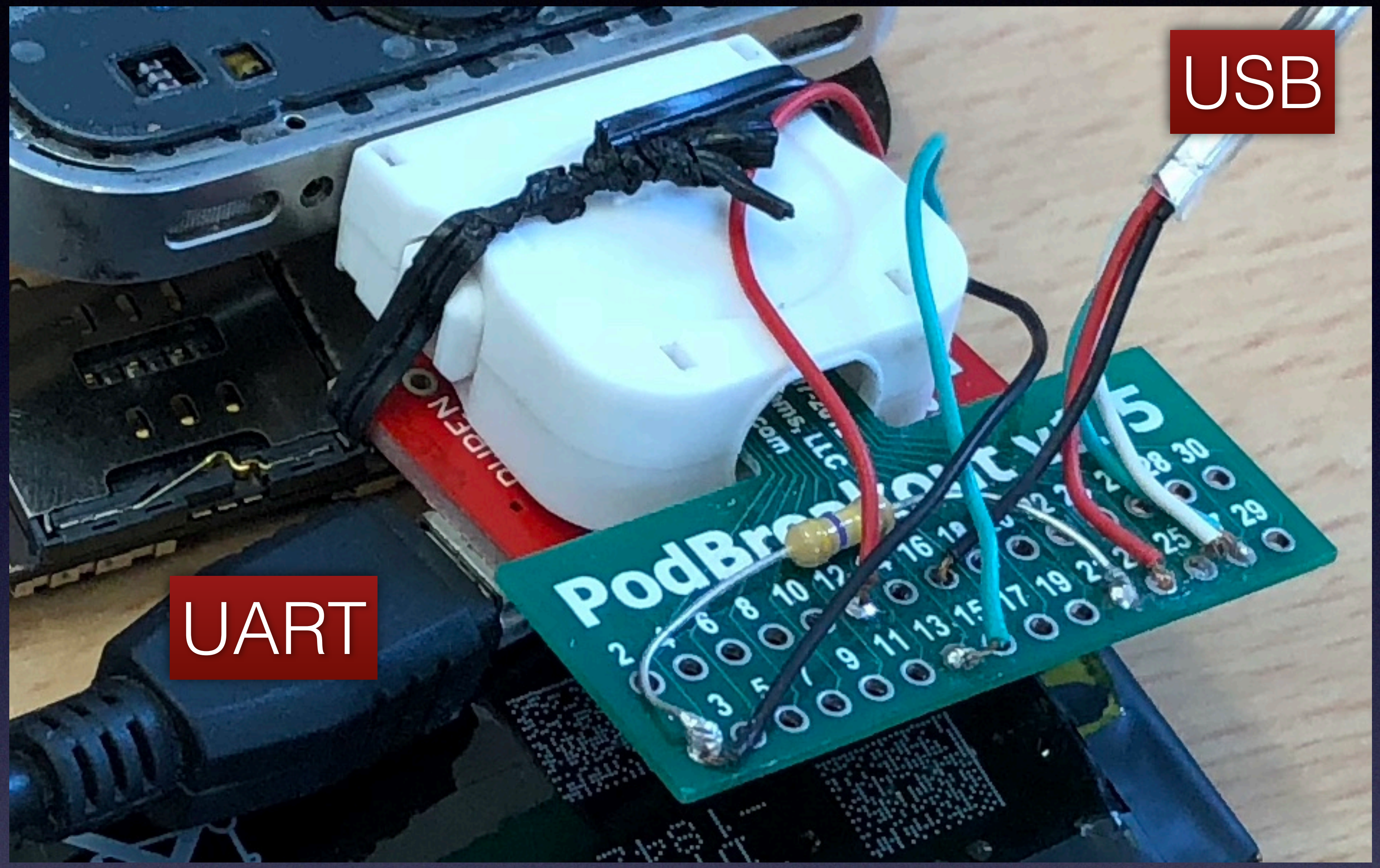
Requirements for EM-attack

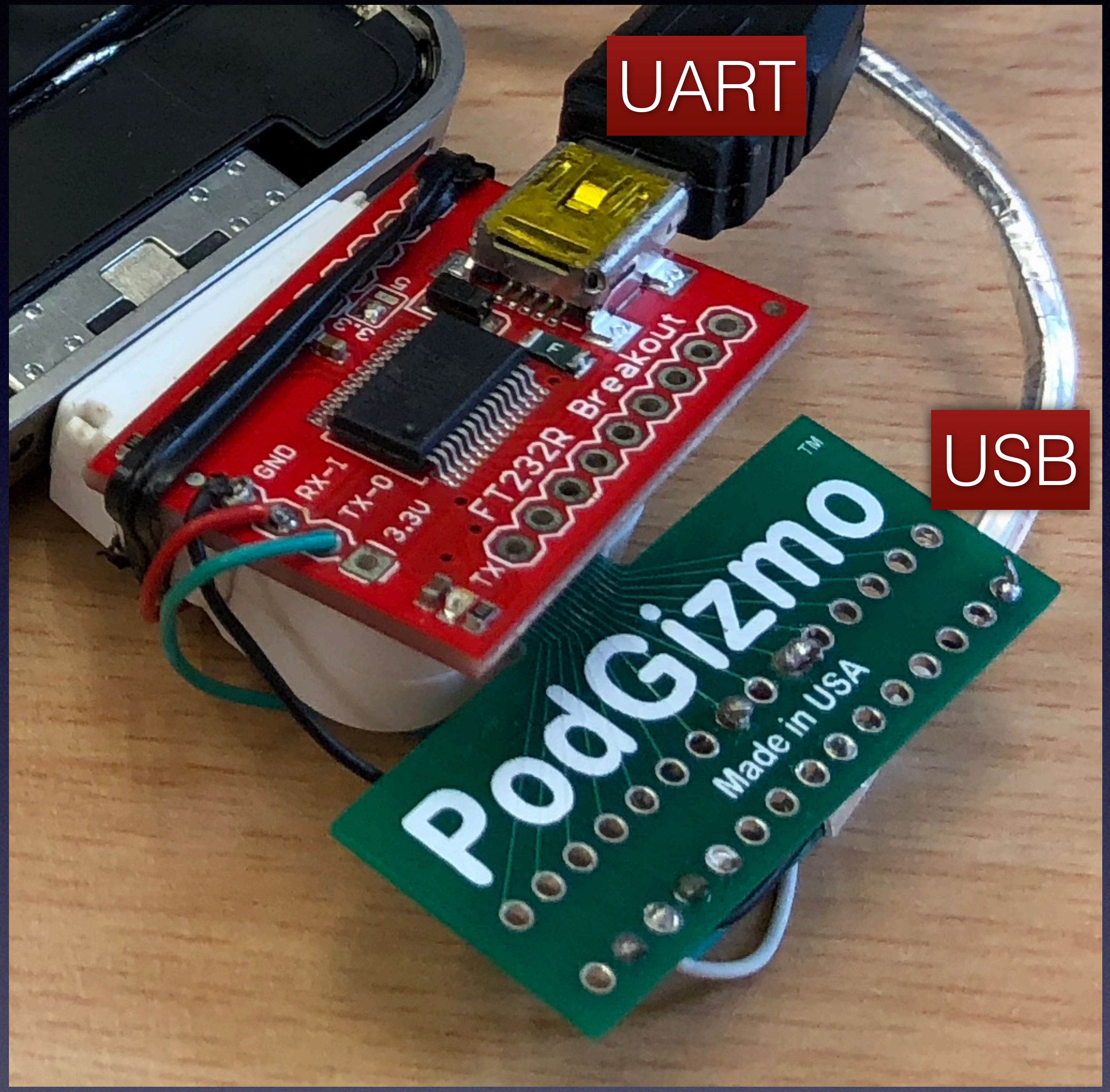
- Code execution on target
  - Oracle access to AES engine
    - With target key
    - With known key (Not required, but you really want this!)
- Low latency trigger for the Oscilloscope measurements
- EM-Probe, Oscilloscope, Computational resources ...

Target setup (software)

- limera1n BootRom exploit gives early code execution
  - Boot patched 1st stage Bootloader (low level initialisation eg. DRAM)
  - Boot patched 2nd stage Bootloader (for a more convenient environment)
  - Deploy custom payload in 2nd stage Bootloader (provides custom shell)
    (easy "bare metal" interface with target hardware)
- Interface via: Proprietary USB interface & UART
- Gives full access to RAM / MMIO
- Best noise-free environment you can get!
  1 active CPU core, cooperative scheduler, no unexpected interrupts...

Target setup (hardware)





- Interface:
  - USB + UART requires custom cable (using FT232R board for UART-to-USB)

UART as trigger? No!

- Universal **Asynchronous** Receiver Transmitter
- Possibly buffered (on sender **and** receiver), not real-time
- Using (standard) bitrate of 115200 bit/s (8,68 µs / bit)
  1 byte transfer takes 1 byte ~ 11*8,68µs = 95.47µs
- Immediate start-stop signal ~ 190.94µs
  (Assuming no buffering in between)
- SPOILER: Our target AES is **much** faster

Peripherals

- ARM peripherals are connected over GPIO
  - Buttons
  - Modules
    Wifi, Bluetooth, GPS, Gyro, Compass
  - Control signals
    LCD, Cam ...

Peripherals

ARM peripherals are connected over GPIO

- **Buttons**
- Modules
  Wifi, Bluetooth, GPS, Gyro, Compass
- Control signals
  LCD, Cam ...

Easily accessible from outside!

GPIO

- Buttons are GPIO input
- But **G**eneral**P**urpose**I**nput**O**utput pins are configured at boot
- We have code execution, so reconfiguration is possible!
  - Reconfigure Volume Button GPIO pin to be output instead of input
- (GPIO is **M**emory**M**apped**IO**)
  - Write to address "in RAM" to set pin high/low
- Much faster than UART and synchronous!

GPIO button output

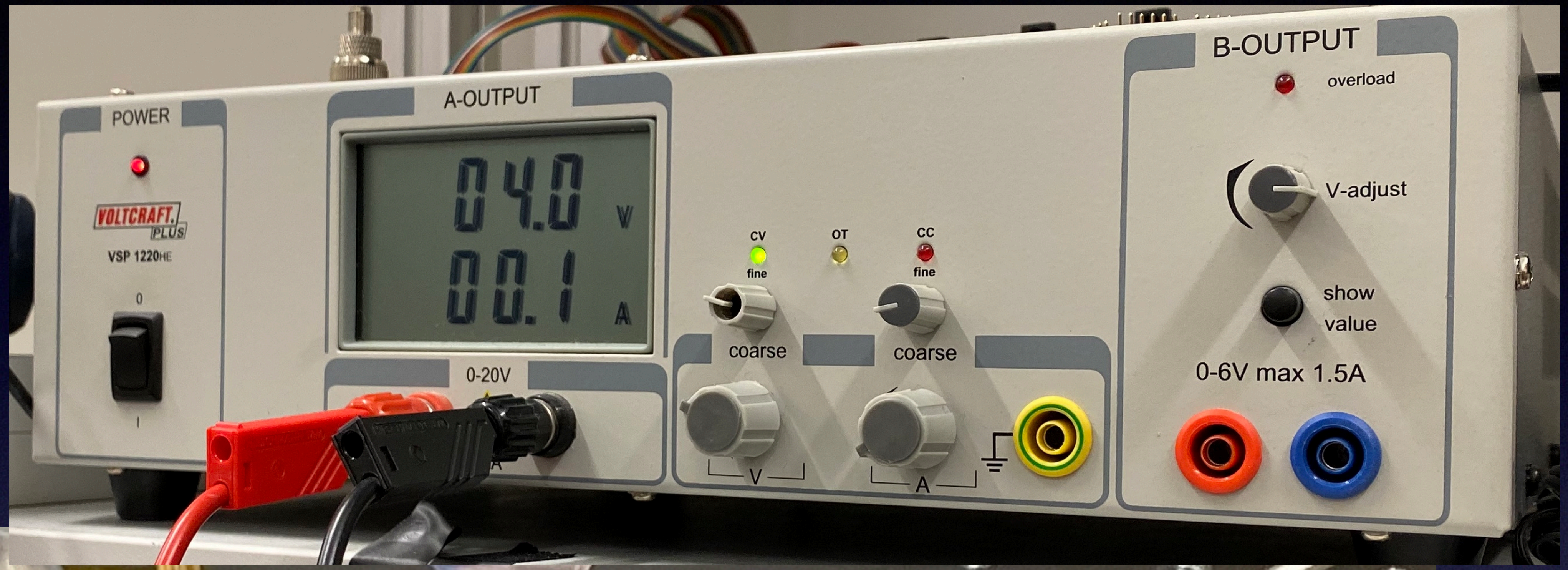
Payload

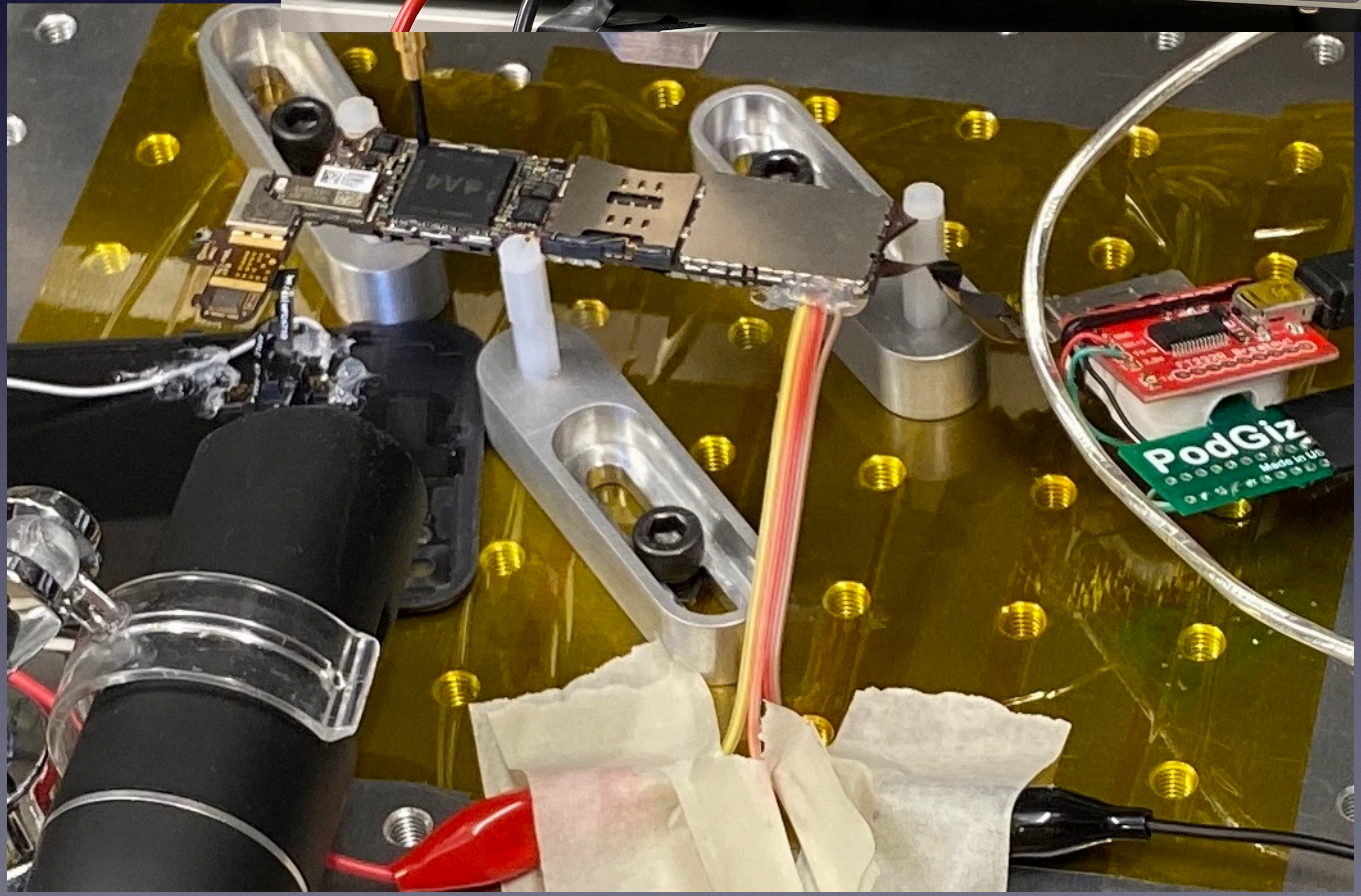
- Trigger high
- AES start
- Check AES done
- Trigger low

```
asm(
    //prepare GPIO write
    "movs r4, #0xc\n\t"
    "movt r4, #0xbfa0\n\t"
    "mov  r0, #0x92\n\t"
    "mov  r1, #0x93\n\t"


    //prepare DMA access
    "movw r5, #0x2000 \t\n" //RX (r1)
    "movt r5, #0x8700 \t\n"

    "ldr r7, [r5]\t\n" //RX_val
    "orr r7, r7, #0x1\t\n" //RX_val
    "str r7, [r5]\t\n" //store RX  first!


    "movw r6, #0x1000 \t\n"
    "movt r6, #0x8700 \t\n" //TX (r0)

    "ldr r7, [r6]\t\n" //TX_val
    "orr r7, r7, #0x1\t\n" //TX_val

   //about to enter time critical section!
   "strb r1, [r4]\n\t"  //set GPIO high (AES_START)
   //START_CRITICAL_SECTION

    "str r7, [r6]\t\n" //store TX (AES_START)

    "w1:\t\n"
    "ldr r7, [r5]\t\n" //RX_val
    "and r7, r7, #0x30000\t\n" //load RX_val and check for idle
    "cmp r7, #0x10000\t\n"
    "beq w1\t\n"
     //-- END_CRITICAL_SECTION
     "strb r0, [r4]\n\t" //set GPIO low
    );
}
```

Measurment Setup

- Running AES in a loop uses more power than USB provides
  - Dischargees battery
  - Limited measuring time :(
- Use custom power supply!
  - Remove battery
  - Connect lab power supply





Oscilloscope

- LeCroy WaveRunner 8254M oscilloscope
  - 40*GS/s* sampling rate
- Langer EMV-Technik RF-B 0,3-3
  - 2mm diameter
  - 30MHz to 3GHz frequency range
- Langer EMV-Technik PA 303 SMA
  - 100kHz to 3GHz by 30dB

Analysis

Gather information

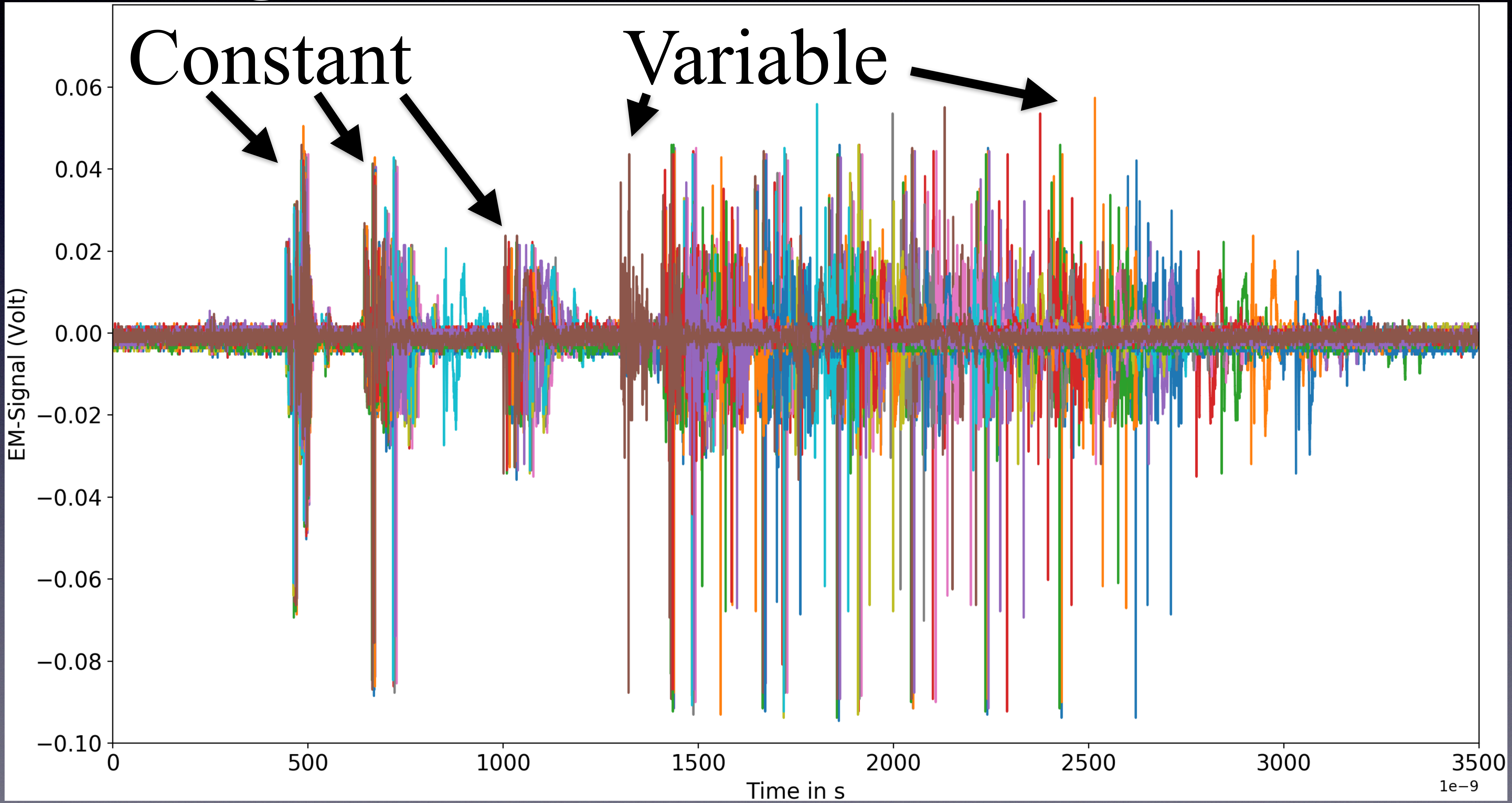
- Search online + manual reverse engineering
- AES 128/192/256
- UID / GID / user - key
- CBC / ECB mode

Timing analysis

- We have **start** - **end** GPIO signals
- Study EM-traces
- Run AES on multiple blocks (CBC)

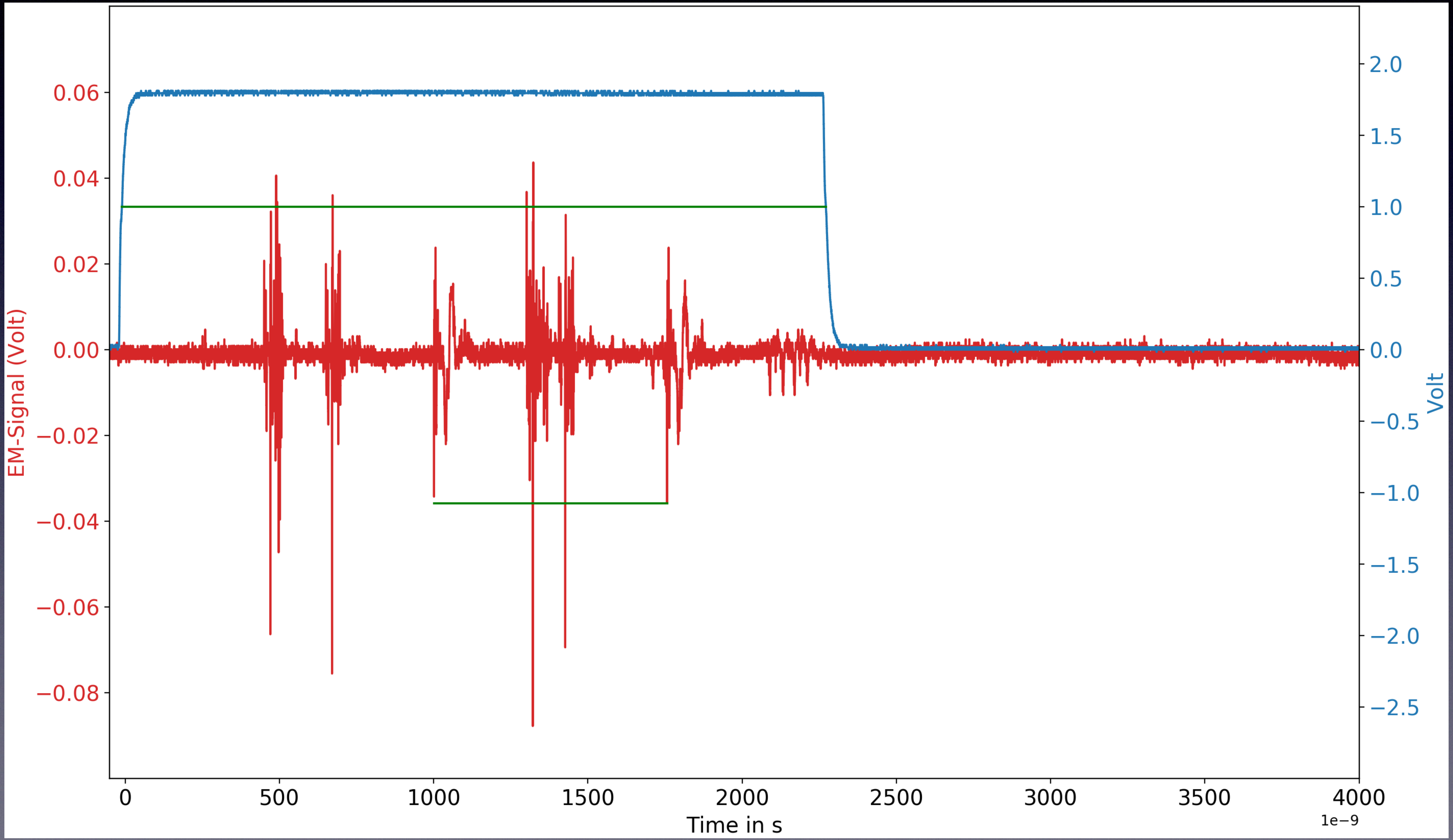
"constant"
"constant"
2 Blocks
4 Blocks
8 Blocks
16 Blocks
EM-Signal (Volt)
Volt
Time in s
1e−9

Timing Analysis: Multiple Blocks



Timing Analysis

- 1 Block
  - GPIO: 2000ns
  - Peak: 755.58ns
- 16 Blocks (per block)
  - GPIO: 205.82ns
  - Peak: 124.78ns
- AES < 124ns



Leakage assessment

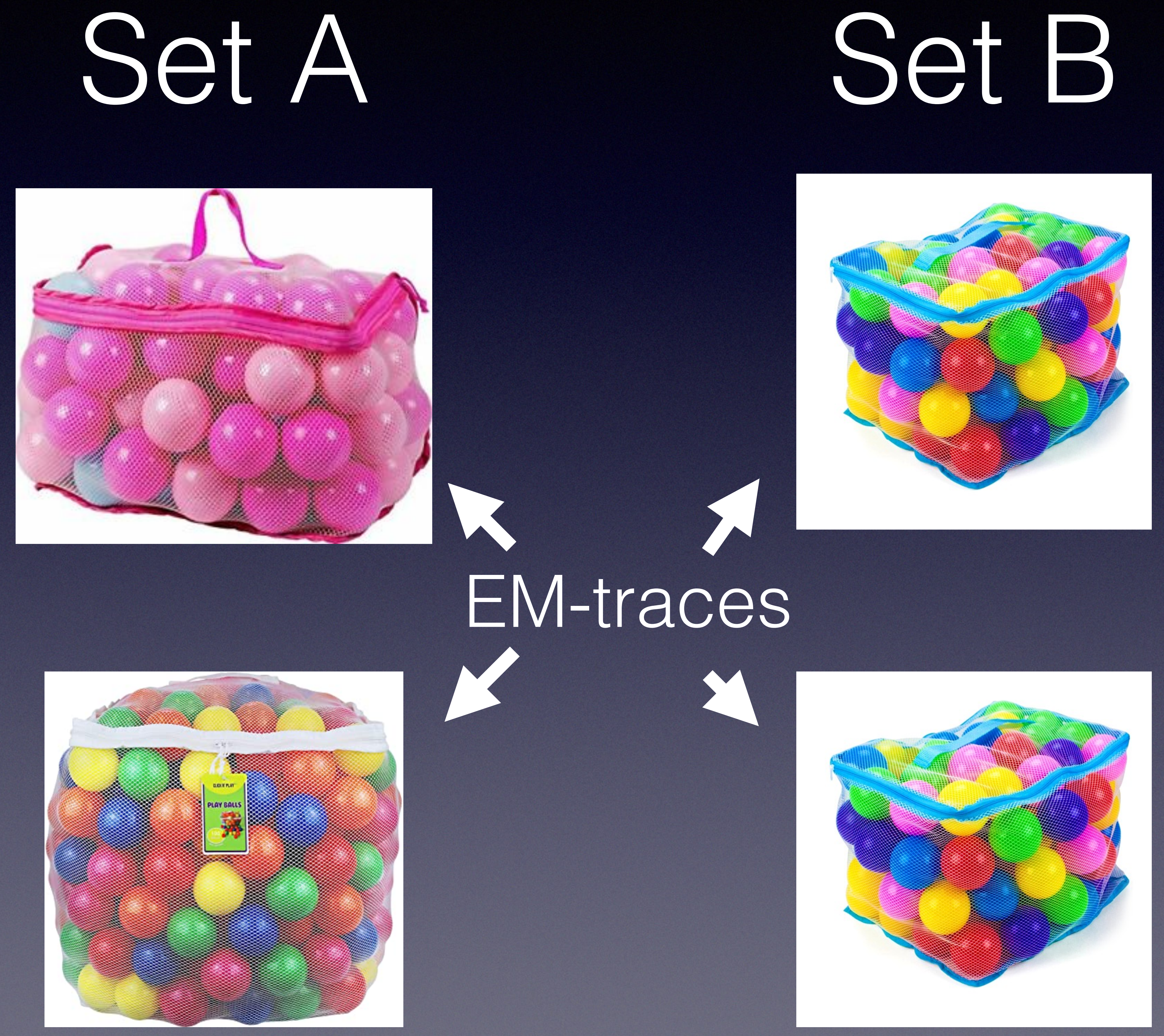
- Non-specific t-test
  (Process fixed vs random AES input)
- Same intermediate values for fixed input
  (for non-masked implementations)
- If groups distinguishable by EM-traces, attack might be possible
- When unsuccessful, try higher order t-test

t-test (explained)

- statistical hypothesis test (confirm or reject)
- null hypothesis
  = Assumes 2 Sets are drawn from same distribution
  rather than from 2 separate distributions
- "Statistical value for determining likelihood that 2 given sets are drawn from two distinct distributions"

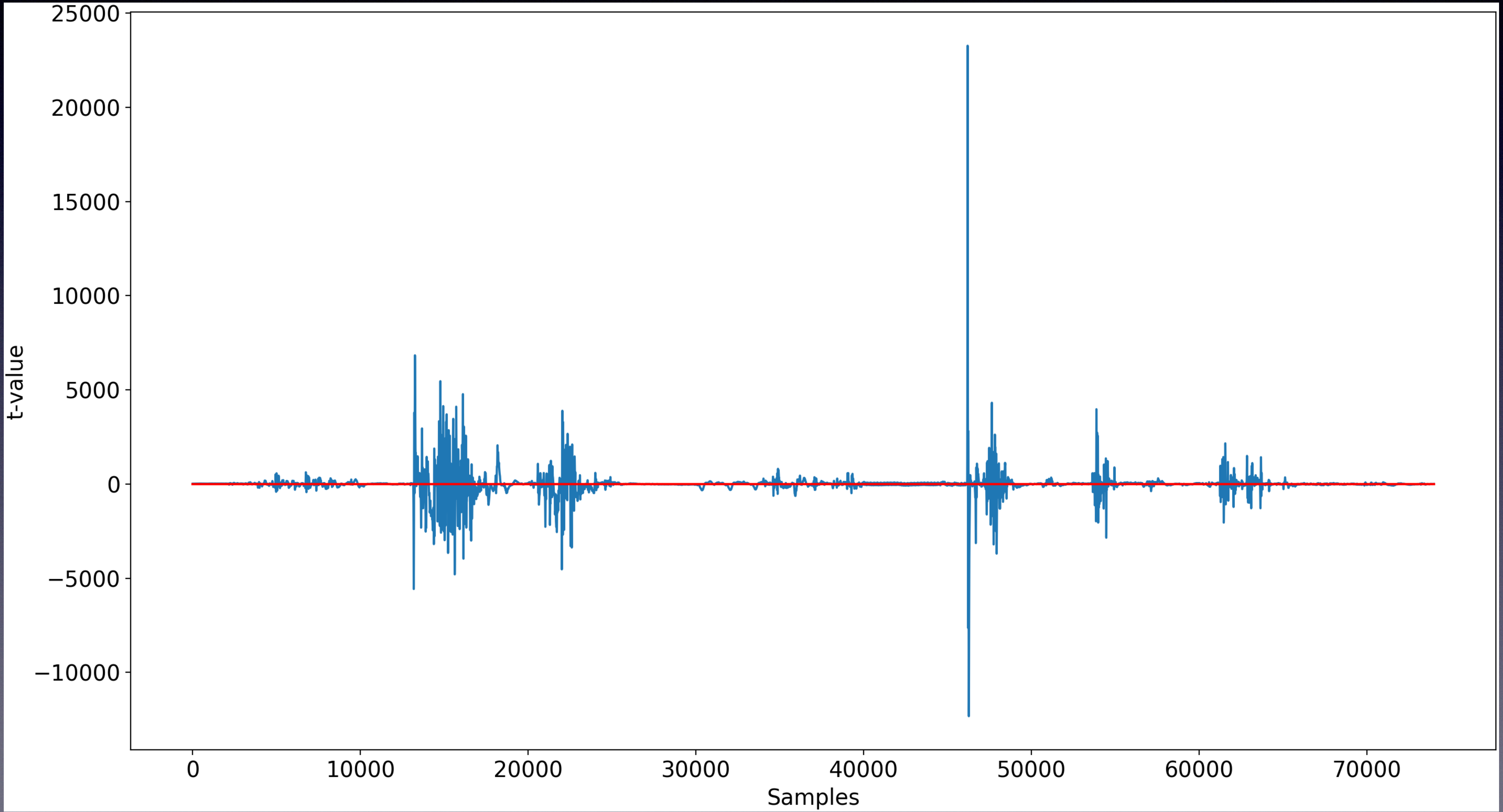
t-test (explained)

- AES fixed vs. random (high absolute t-vale)
- AES random vs. random (t-value close to 0)



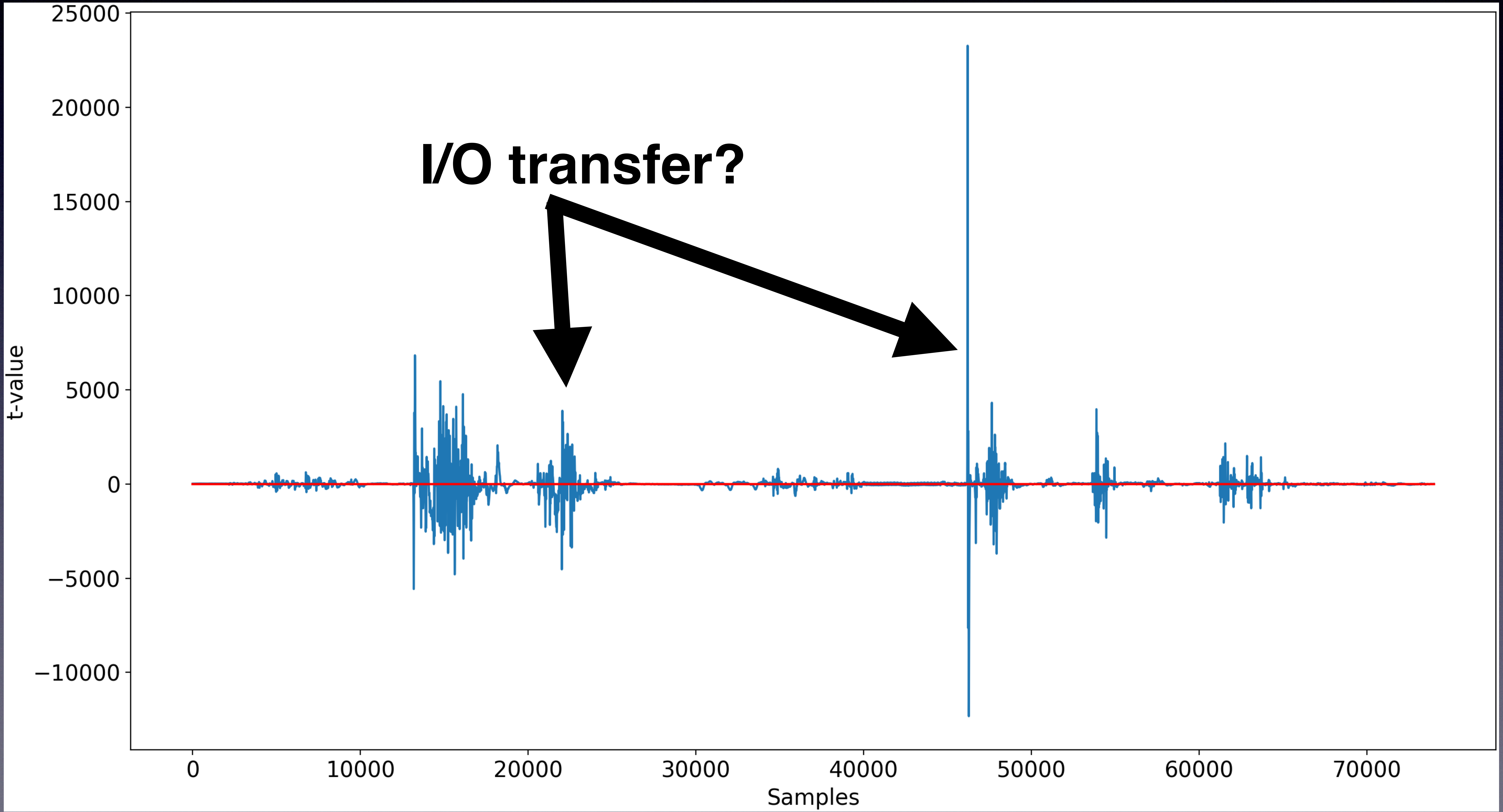
t-Test

- 10.000.000 Traces
- Way above 4.5/-4.5 threshold!

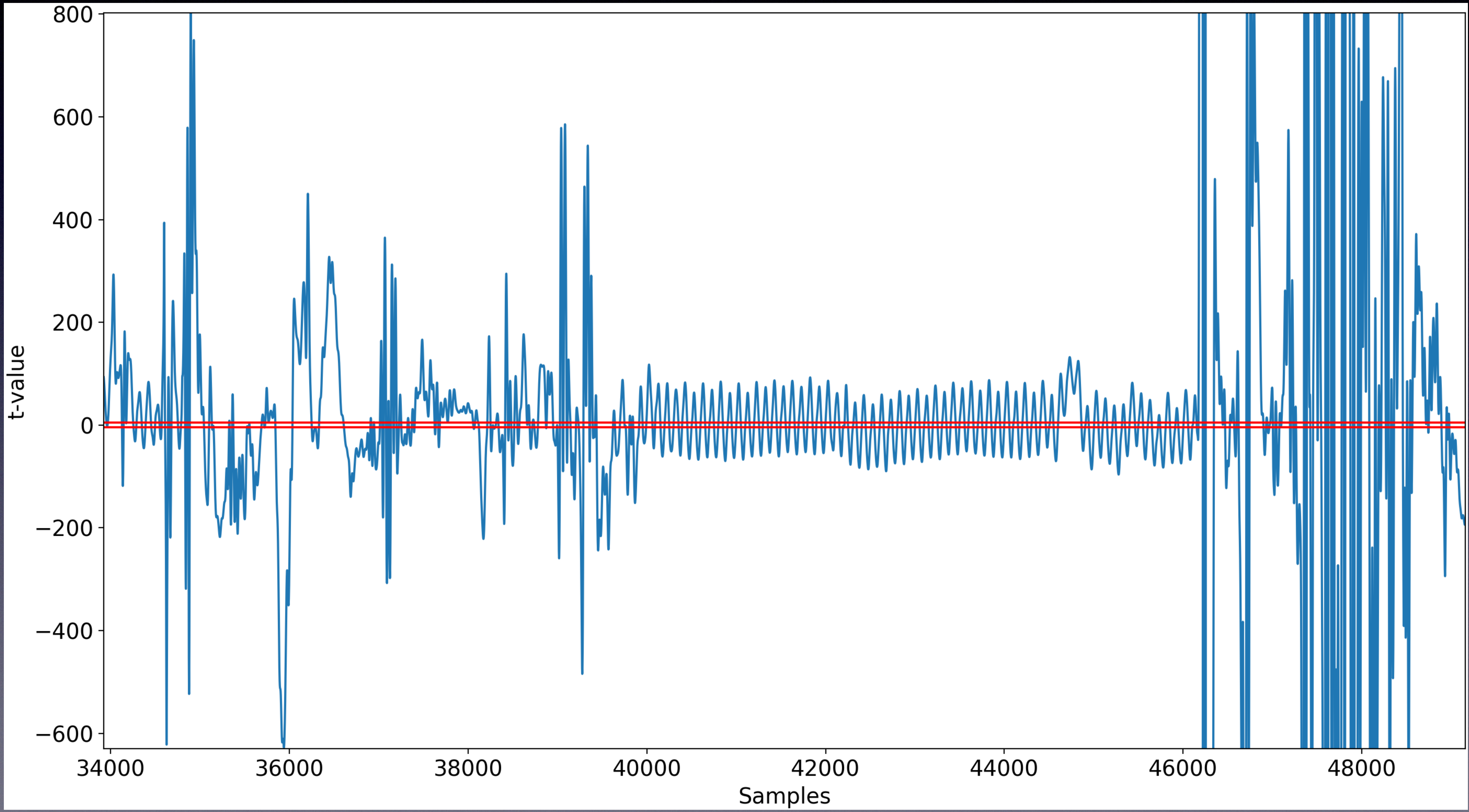


t-Test

- 10.000.000 Traces
- Way above 4.5/-4.5 threshold!

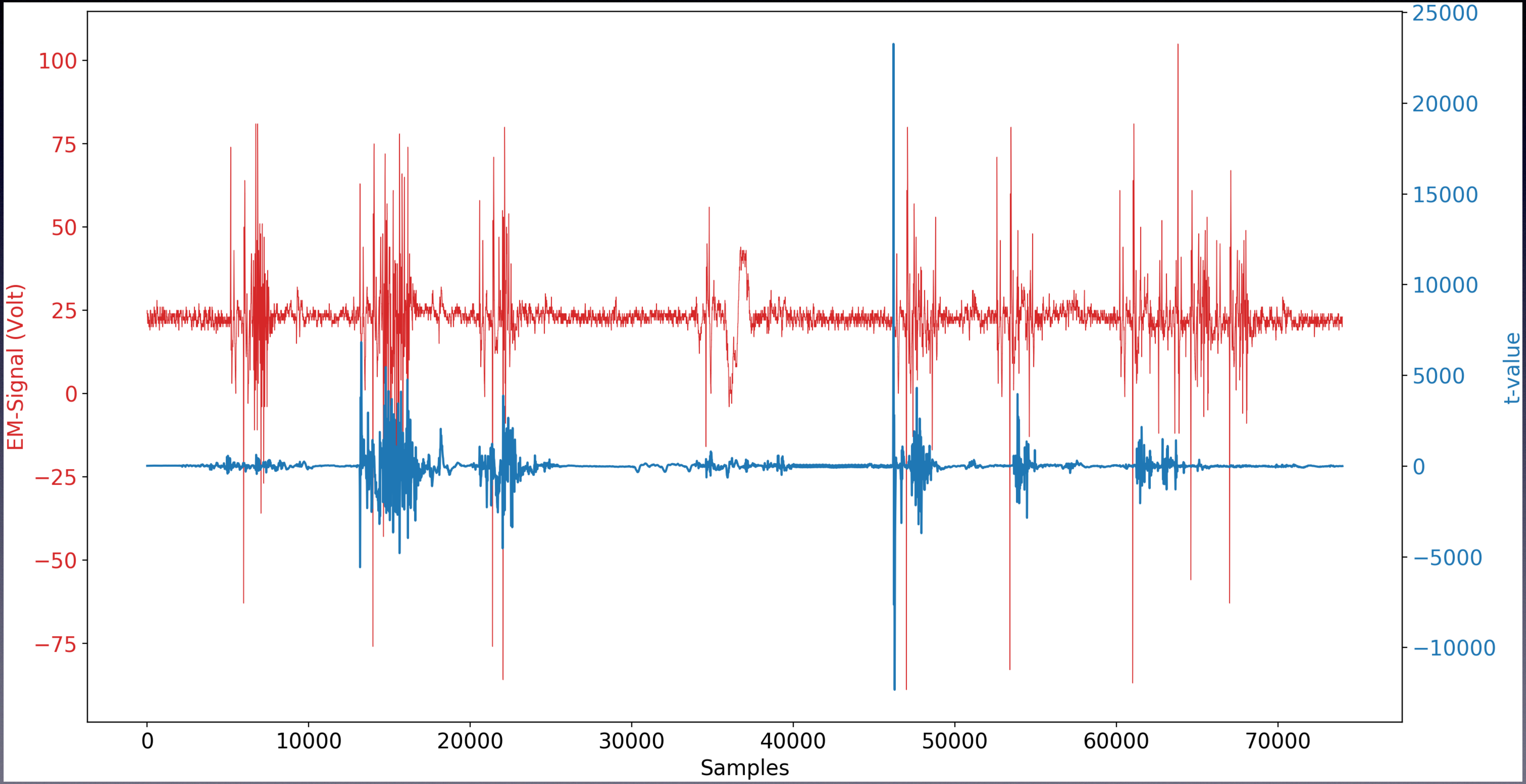


t-Test



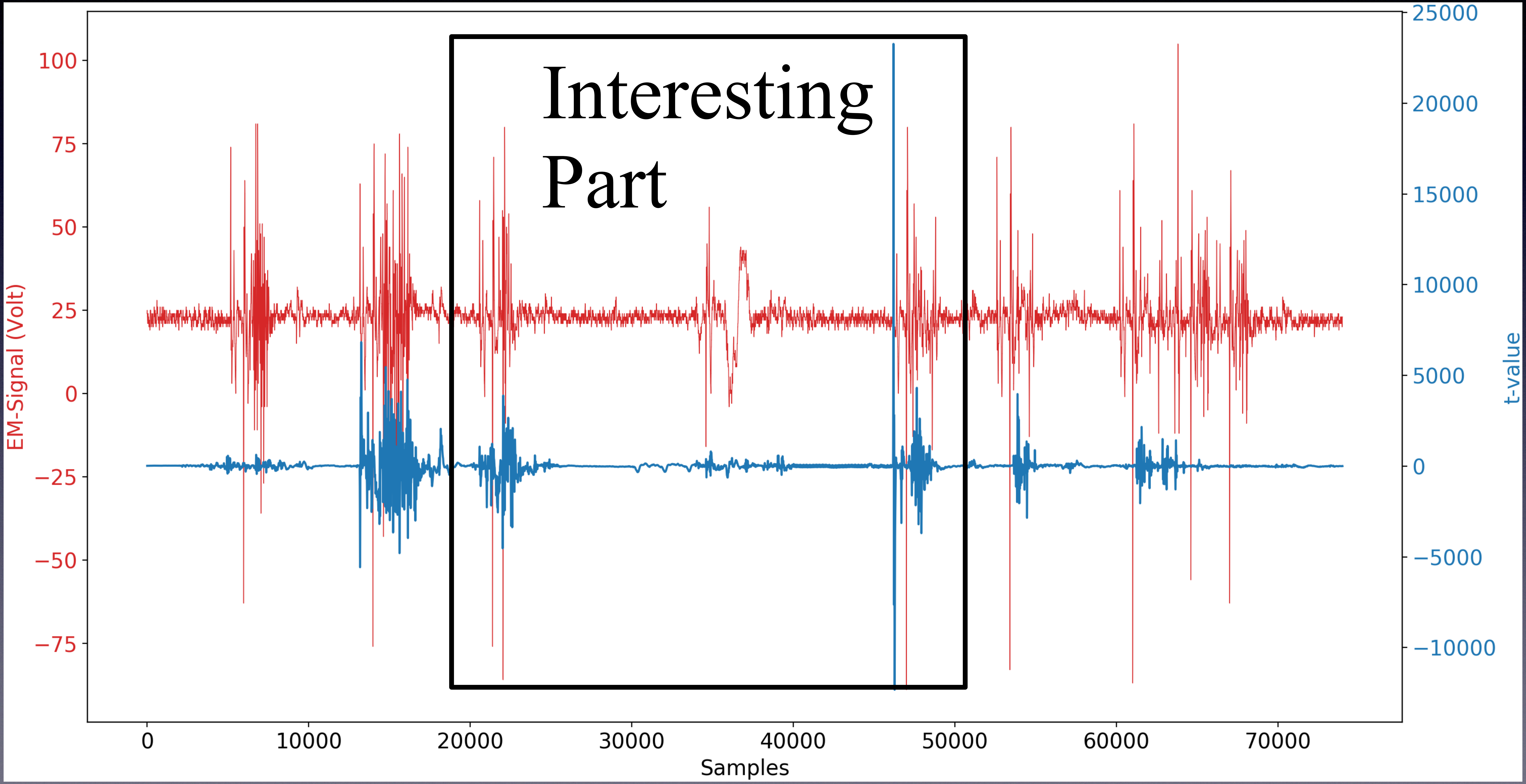
- Way above 4.5/-4.5 threshold!
- *Something* is leaking!

t-Test



EM-Signal (Volt)
t-value
Samples
100
75
50
25
0
−25
−50
−75
25000
20000
15000
10000
5000
0
−5000
−10000
0
10000
20000
30000
40000
50000
60000
70000

t-Test


Interesting
Part
EM-Signal (Volt)
t-value
Samples
100
75
50
25
0
−25
−50
−75
25000
20000
15000
10000
5000
0
−5000
−10000
0
10000
20000
30000
40000
50000
60000
70000

Signal-to-Noise Ratio (explained)

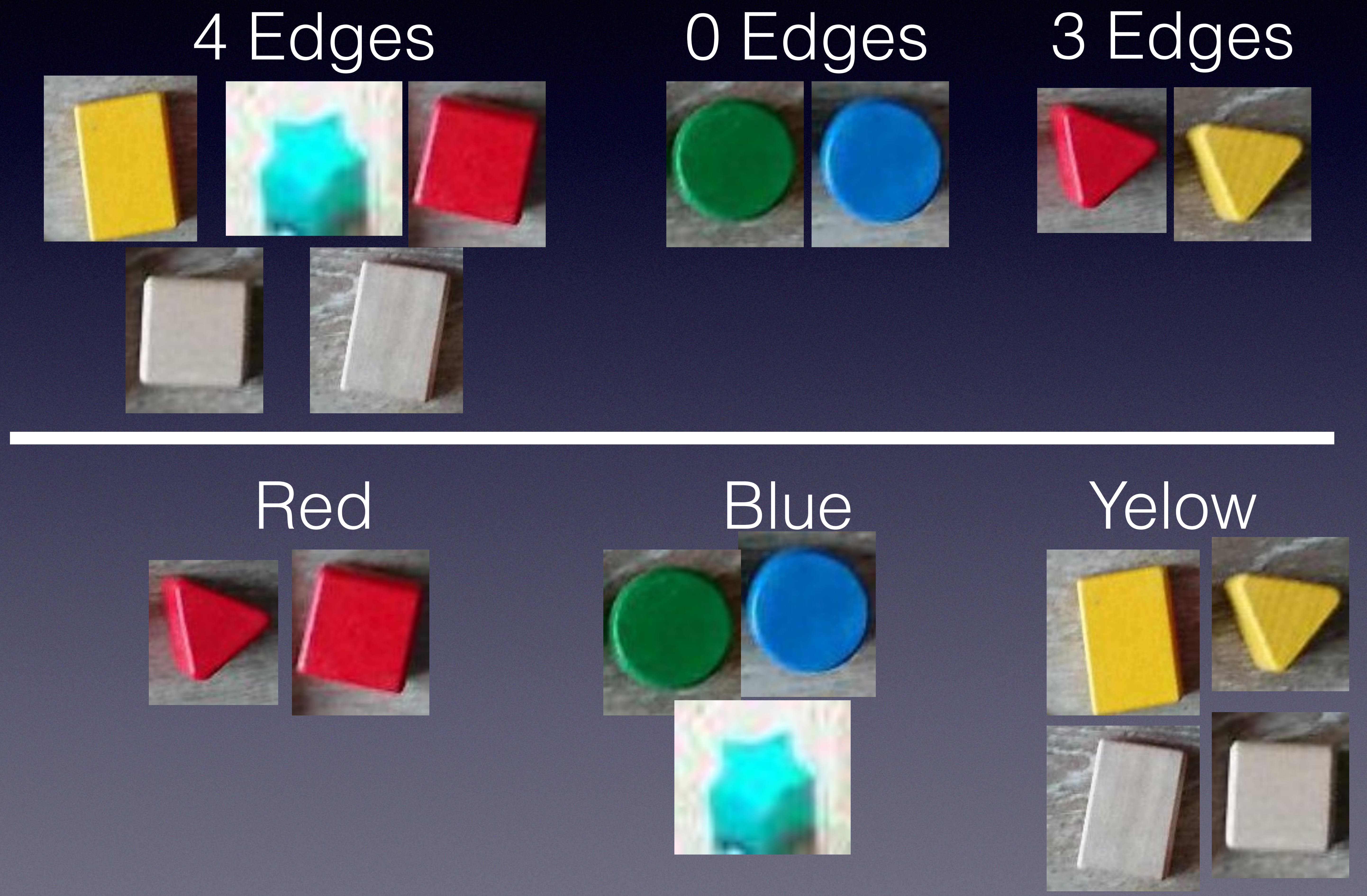
- 1) Use **powermodel** to **sort** traces into **groups** based on a **value**
- Accurate powermodel will have distinct groups
- Other powermodels will have *random* groups
- 2) Gets value on *how different* groups are from each other
  - Info about *accuracy* of power model (use for attack)
  - Info about where the model *fits* (point in time in the trace)

Signal-to-Noise Ratio (explained)

- Sort by *model*
- Compute SNR for "How good are groups sorted judging by edges"

Sort by Edges
SNR=0.92

Sort by Color
SNR=0.42

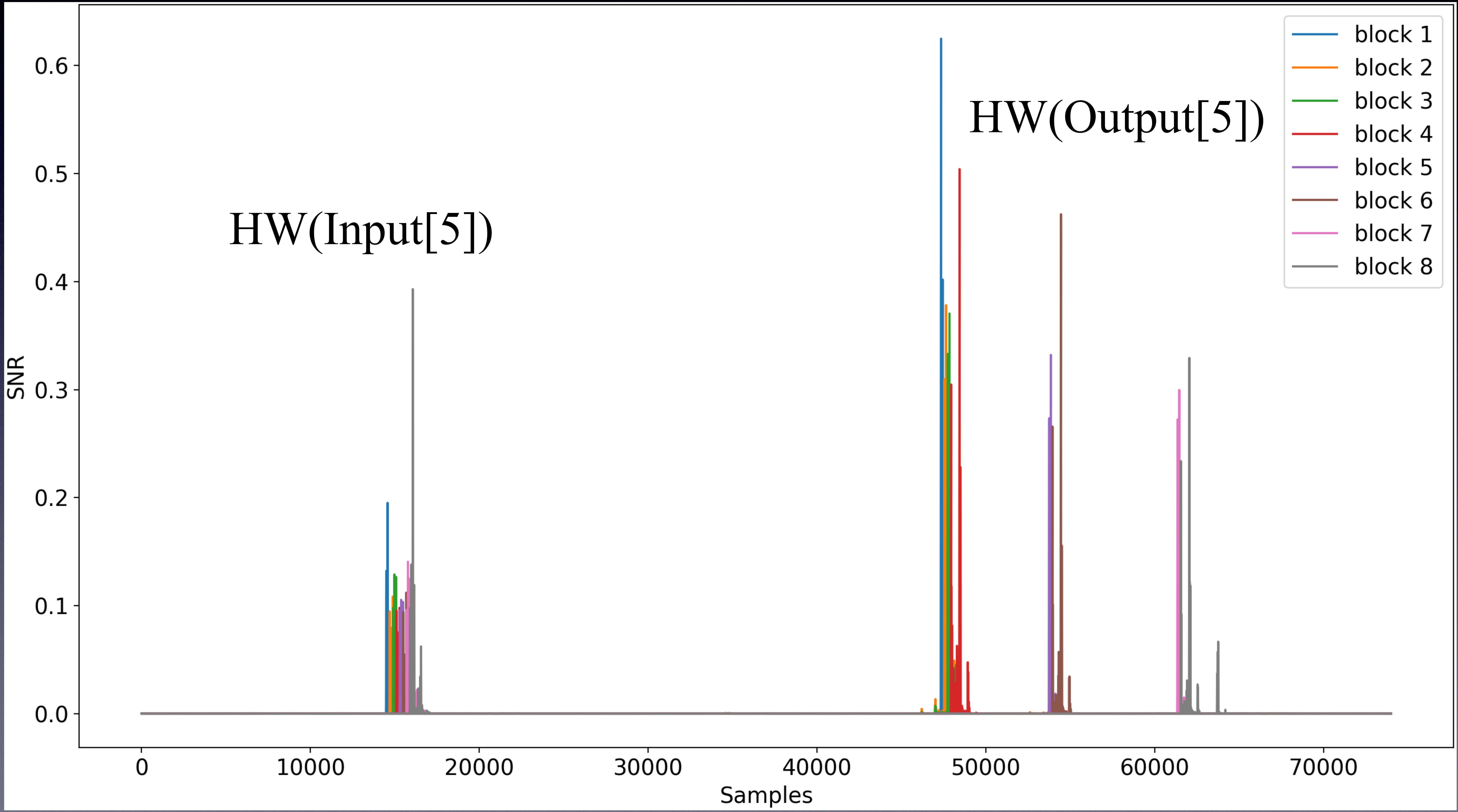


Signal-to-Noise Ratio

- Use Input/Output (HW8) as model
- Check if/where signal can be seen

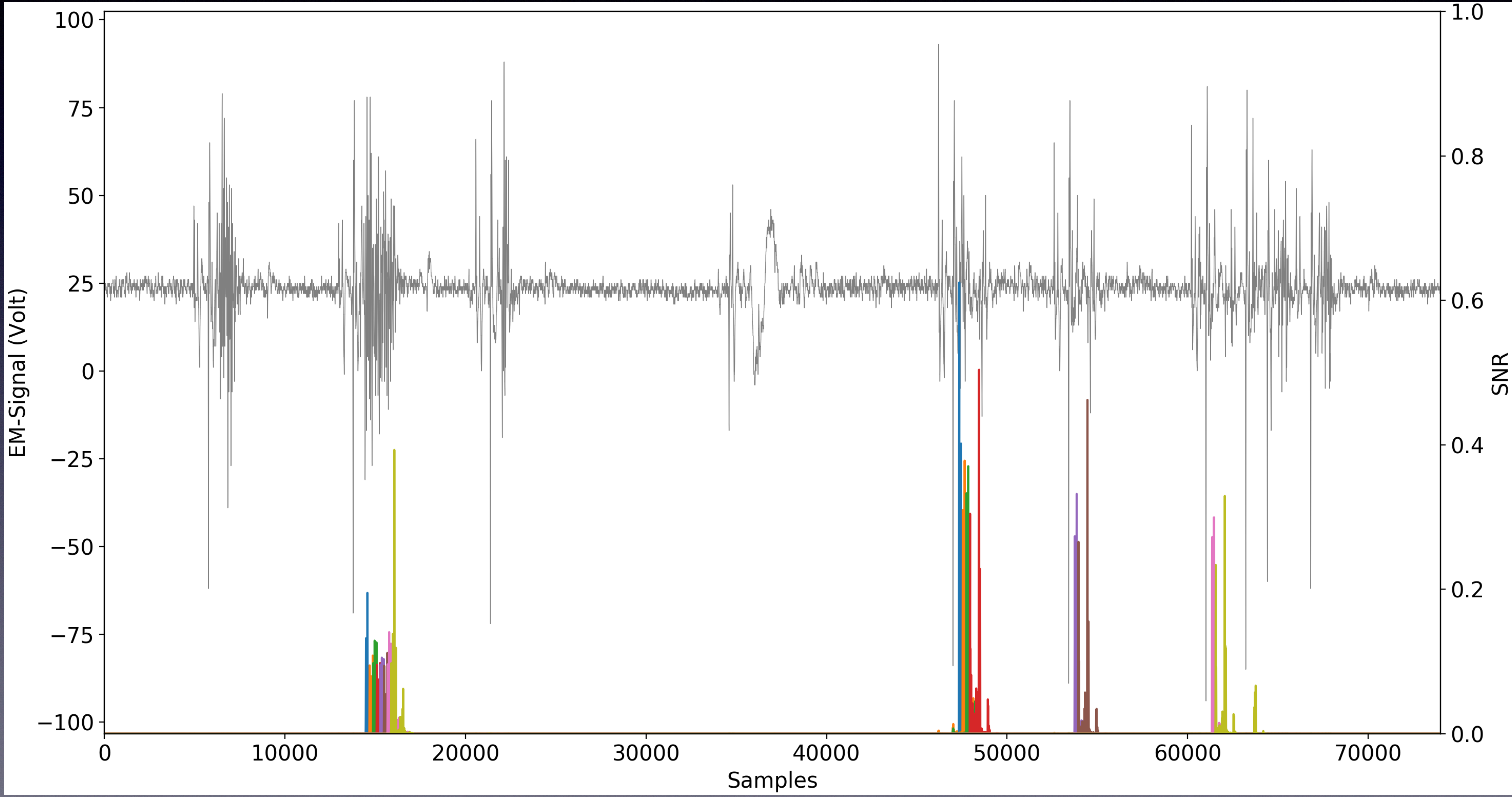
Signal-to-Noise Ratio

- 10.000.000 Trace
- Other bytes yield similar SNRs

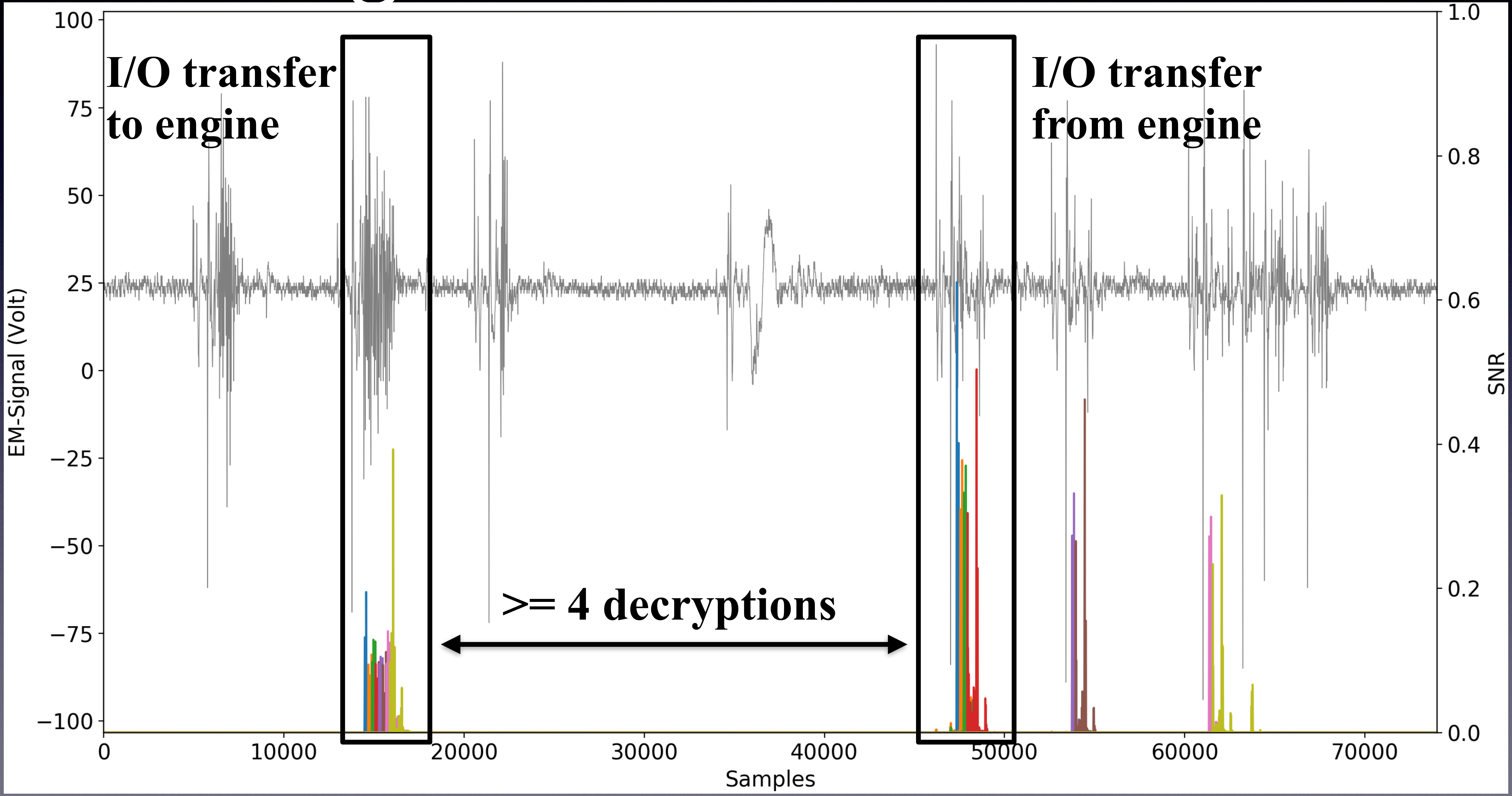


Signal-to-Noise Ratio

- SNR overlaps with *noisy* part of trace
- Likely IO transfers not AES itself



Signal-to-Noise Ratio



EM-probe placement (textbook)

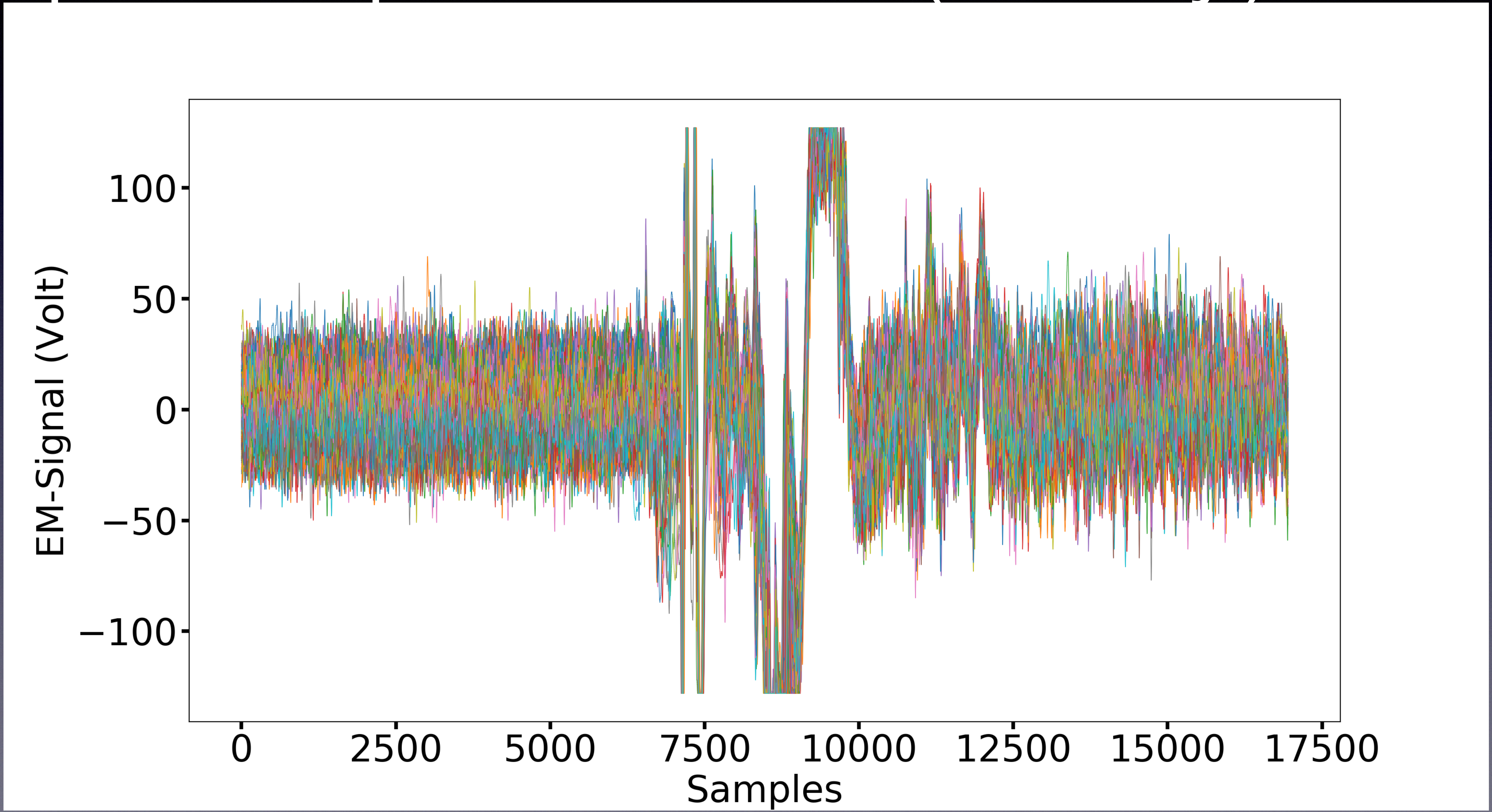
- Move probe around chip
- Look at EM traces
- Try identifying target signal (between start-stop signal)
- Find position with strongest signal

EM-probe placement (reality)

- No visible signal
- Lots of jitter
- Staring too much at traces makes you crazy
  - Don't *over-interpret* noise / random peaks!

EM-probe placement (reality)

- No visible signal
- Spoiler: There are 5 AES in this trace



EM-probe placement (reality)

- Best (initial) strategy: Educated guessing

Search leaking power model

- Model AES intermediate state
- Tested for various implementations
  - T-Table, round based, registers after certain steps ...
- Tried many models with SNR
  - For different bytes/rounds
  - For enc/dec
- No success :(

$V(B_i)$
$HW_8(B_i)$
$HW_{32}(B_i \| B_{i+1} \| B_{i+2} \| B_{i+3})$
$V(S'(C_i \oplus K_i))$
$HW_8(S'(C_i \oplus K_i))$
$Z(C_i \oplus K_i)$
$HW_{32}(T'_0(C_4 \oplus K_4))$
$HW_{32}(T'_1(C_{13} \oplus K_{13}))$
$HW_{32}(T'_2(C_{10} \oplus K_{10}))$
$HW_{32}(T'_3(C_7 \oplus K_7))$
$HW_{32}(T'_0(C_4 \oplus K_4) \oplus T'_1(C_{13} \oplus K_{13}))$
$HW_{32}(T'_0(C_4 \oplus K_4) \oplus T'_1(C_{13} \oplus K_{13}) \oplus T'_2(C_{10} \oplus K_{10}))$
$HW_{32}(T'_0(C_4 \oplus K_4) \oplus T'_1(C_{13} \oplus K_{13}) \oplus T'_2(C_{10} \oplus K_{10}) \oplus T'_3(C_7 \oplus K_7))$
$HW_{32}(T'_0(C_4 \oplus K_4) \oplus (C_0 \| C_1 \| C_2 \| C_3))$
$HW_{32}(T'_0(C_4 \oplus K_4) \oplus (C_0 \| C_1 \| C_2 \| C_3) \oplus (K_0 \| K_1 \| K_2 \| K_3))$
$V(S_{1,i})$
$HW_8(S_{1,i})$
$HW_{32}(S_{1,i} \| S_{1,i+1} \| S_{1,i+2} \| S_{1,i+3})$
$V(P'(S'(C_i \oplus K_i)) \oplus K_{i+16})$
$HW_8(P'(S'(C_i \oplus K_i)) \oplus K_{i+16})$

SNR model problems

- SNR divides traces into groups
- Resources are always tight, especially (V)RAM
  (We process more than 50.000.000 traces with 80000 points)
- More groups require more computing resources
  - HW8 -> 9 groups (ok)
  - HW128 -> 129 groups (too much for efficient implementation)

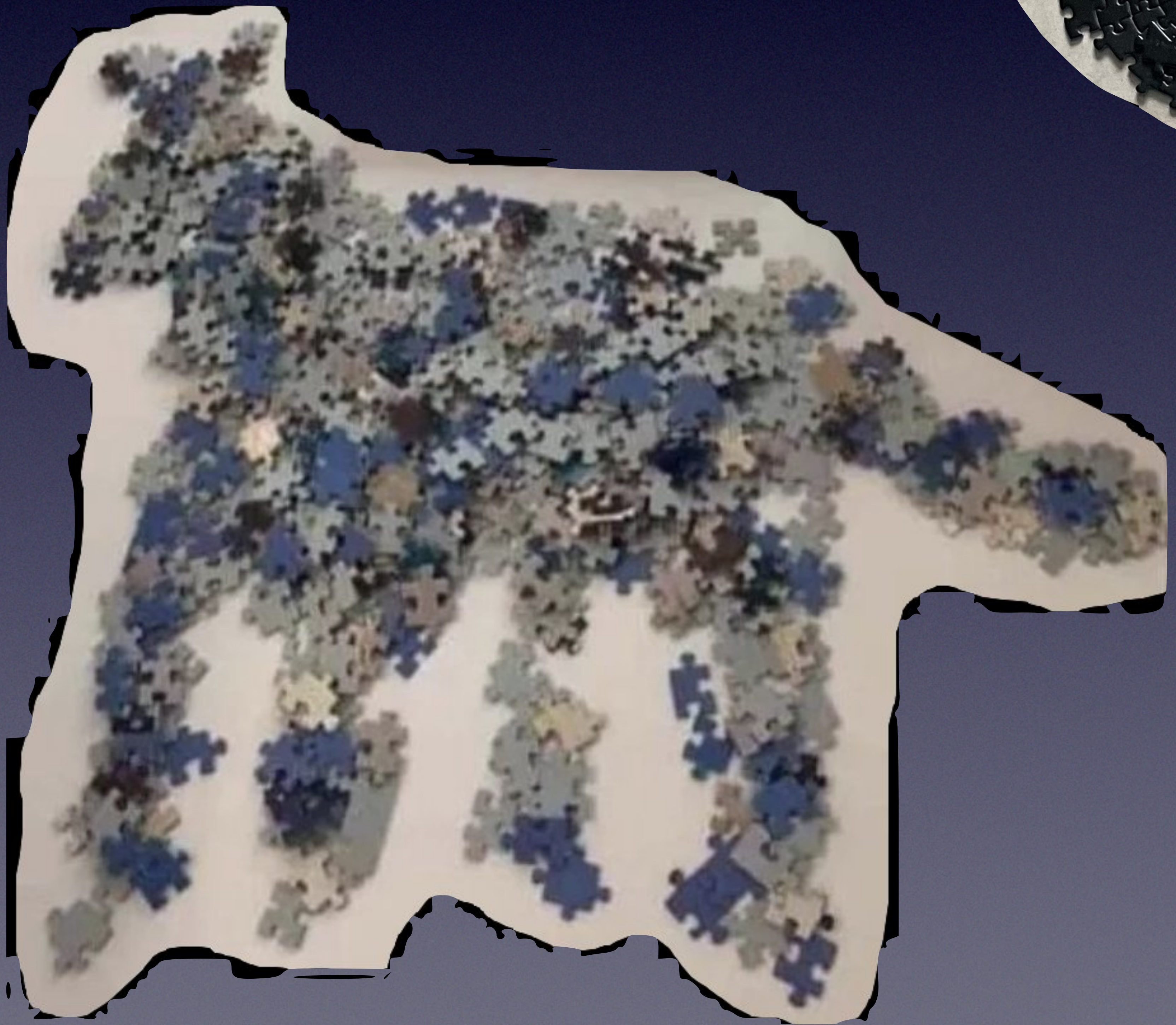
CPA (explained)

- Correlation Power Analysis
- 1) Use **powermodel** to create **model** based on a **value**
- 2) Gets value on *how similar* **model** is to **real** trace
- Lower memory footprint than SNR (1 single modeled trace vs. X groups of real traces)

CPA (explained)

Real Trace

Model A
corr=0.63



Model B
corr=0.27

CPA model testing

- Test various HW128 models for different rounds
- A full 128 bit register might be updated every round
  - (if so,) large model will leak better

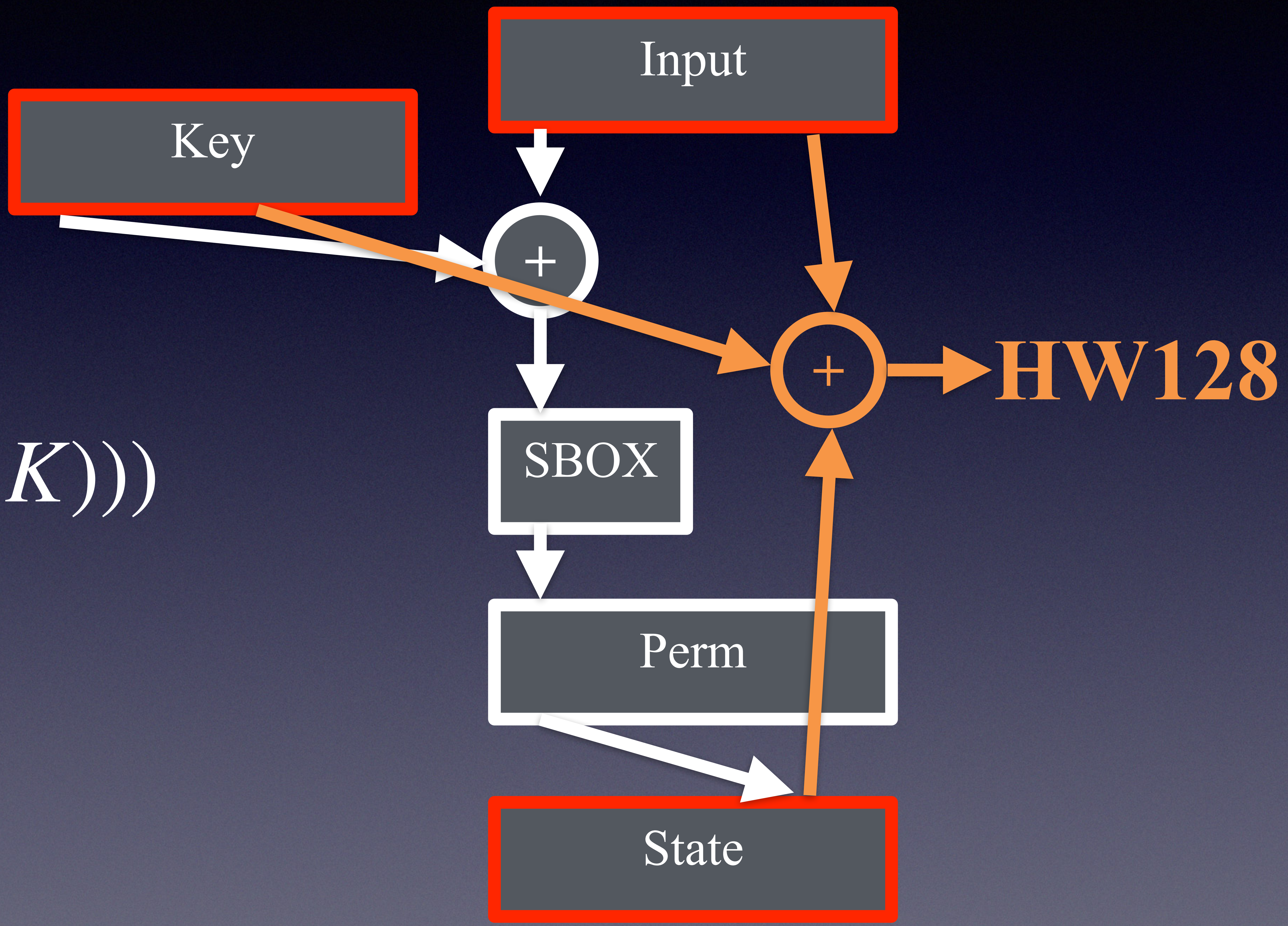
$HW_{128}(S_r)$
$HW_{128}(S'(P'(S_r)))$
$HW_{128}(S'(P'(S_r)) \oplus K_r)$
$HW_{128}(MC'(S'(P'(S_r)) \oplus K_r))$
$HW_{128}(S_r \oplus S_{r+1})$
$HW_{128}(P'(S_r) \oplus P'(S_{r+1}))$
$HW_{128}(S'(P'(S_r)) \oplus S'(P'(S_{r+1})))$
$HW_{128}(S'(P'(S_r)) \oplus K_r \oplus S'(P'(S_{r+1})) \oplus K_{r+1})$
$HW_{128}(MC'(S'(P'(S_r)) \oplus K_r) \oplus MC'(S'(P'(S_{r+1})) \oplus K_{r+1}))$

Leaking powermodel

- Leaking model:
  $HW_{128}(S_r \oplus S_{r+1})$

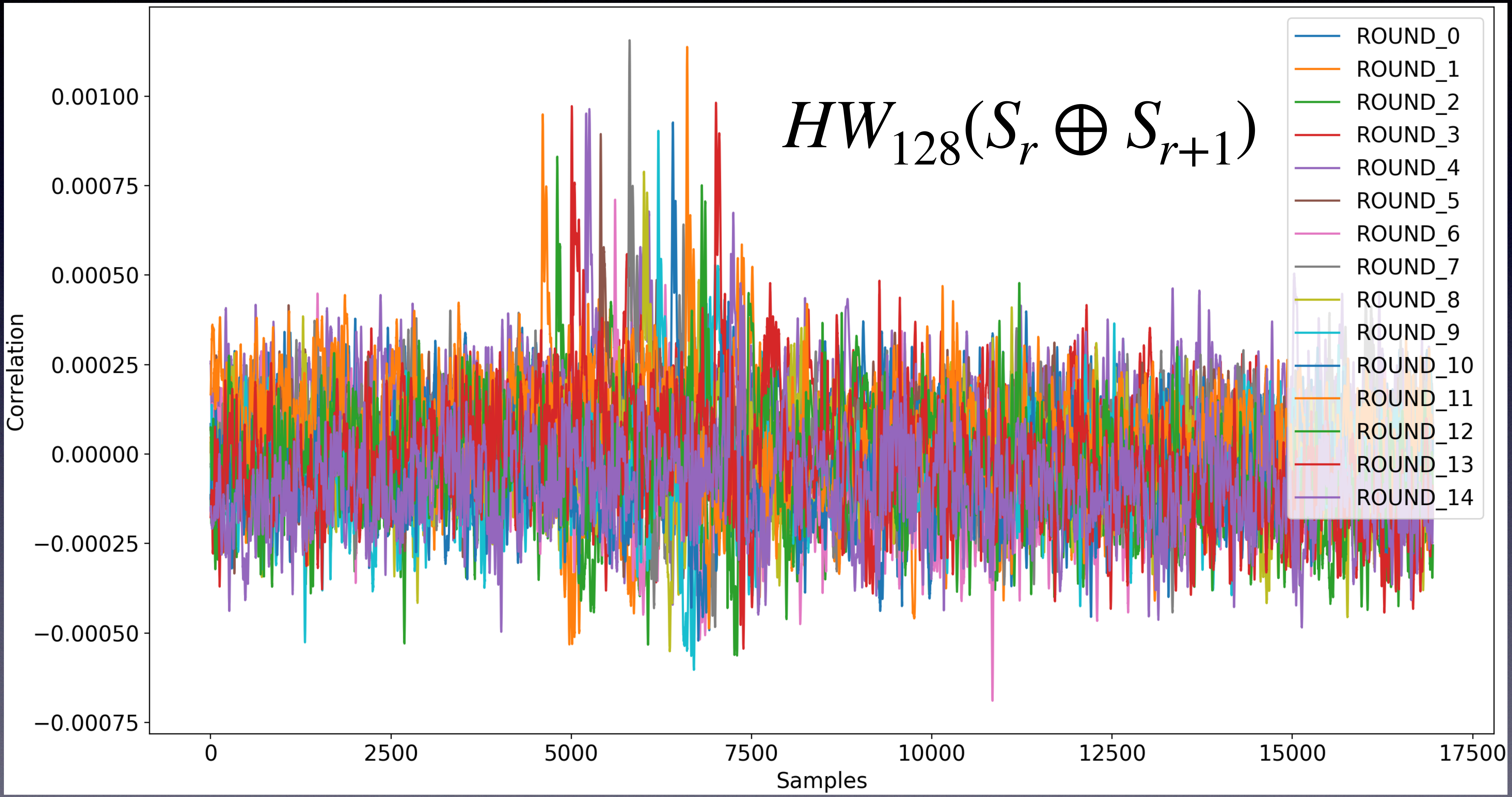
- Round 1:
  $HW_{128}((PT \oplus K) \oplus P(S(PT \oplus K)))$

- PT = plaintext
  K = key
  $P(\cdot)$ = Permutation layer
  $S(\cdot)$ = Substitution layer

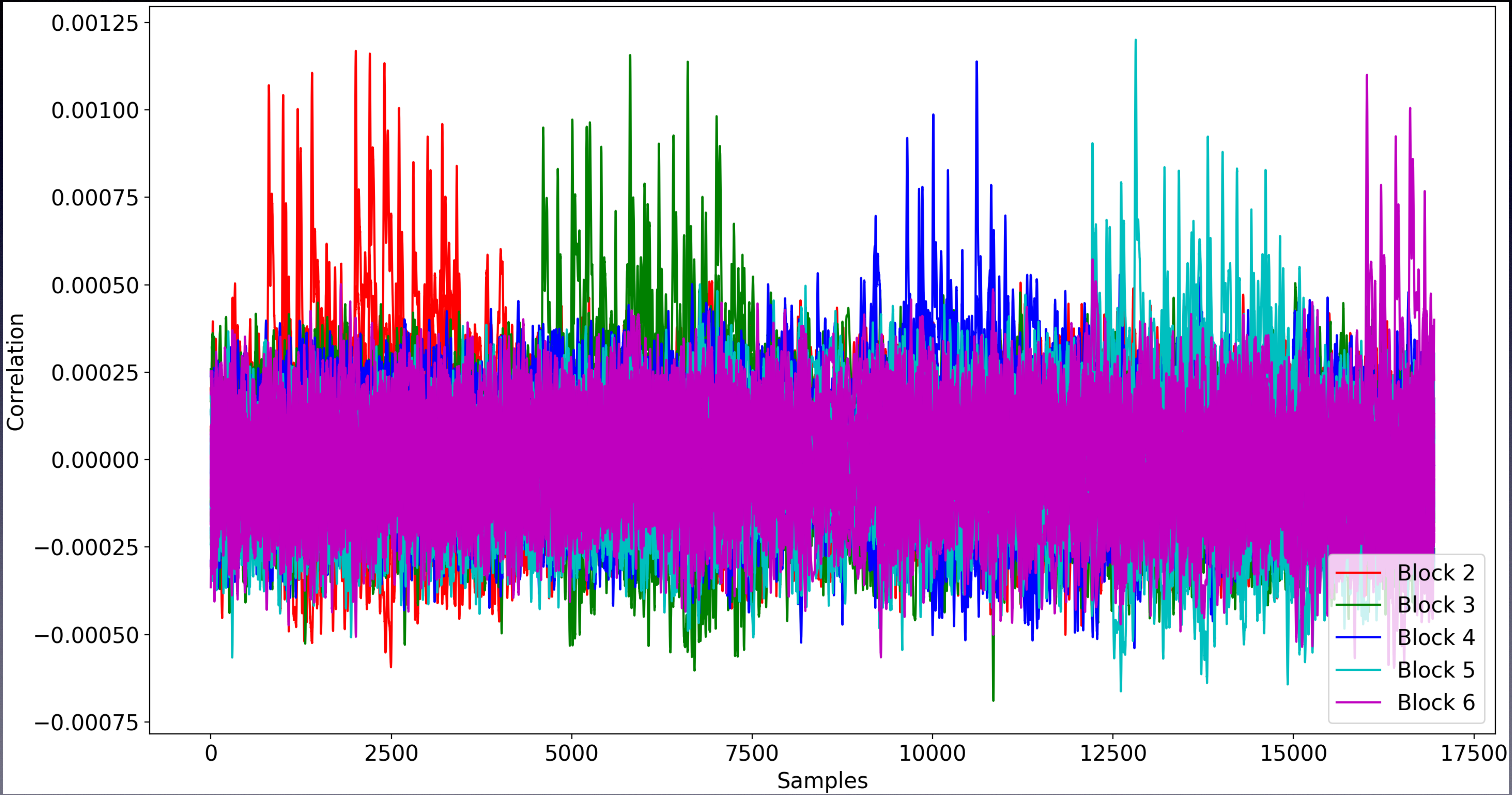


CPA

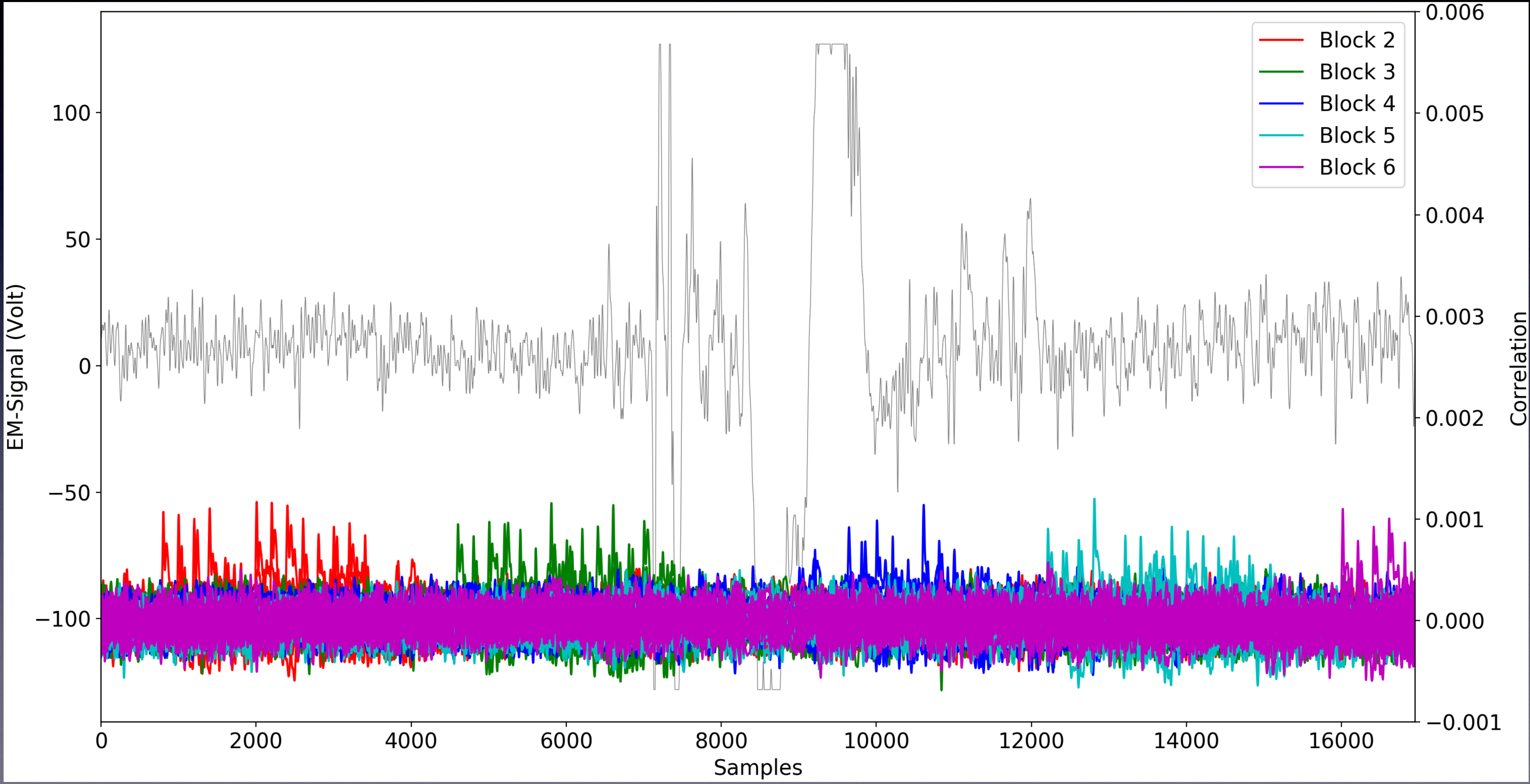
- One peak in each round distinguishes significantly
- 1 round = 5ns
- 14 rounds = 70ns
- 1 block = 95ns (including pre/post processing)
- Round base AES (200MHz clock)



CPA

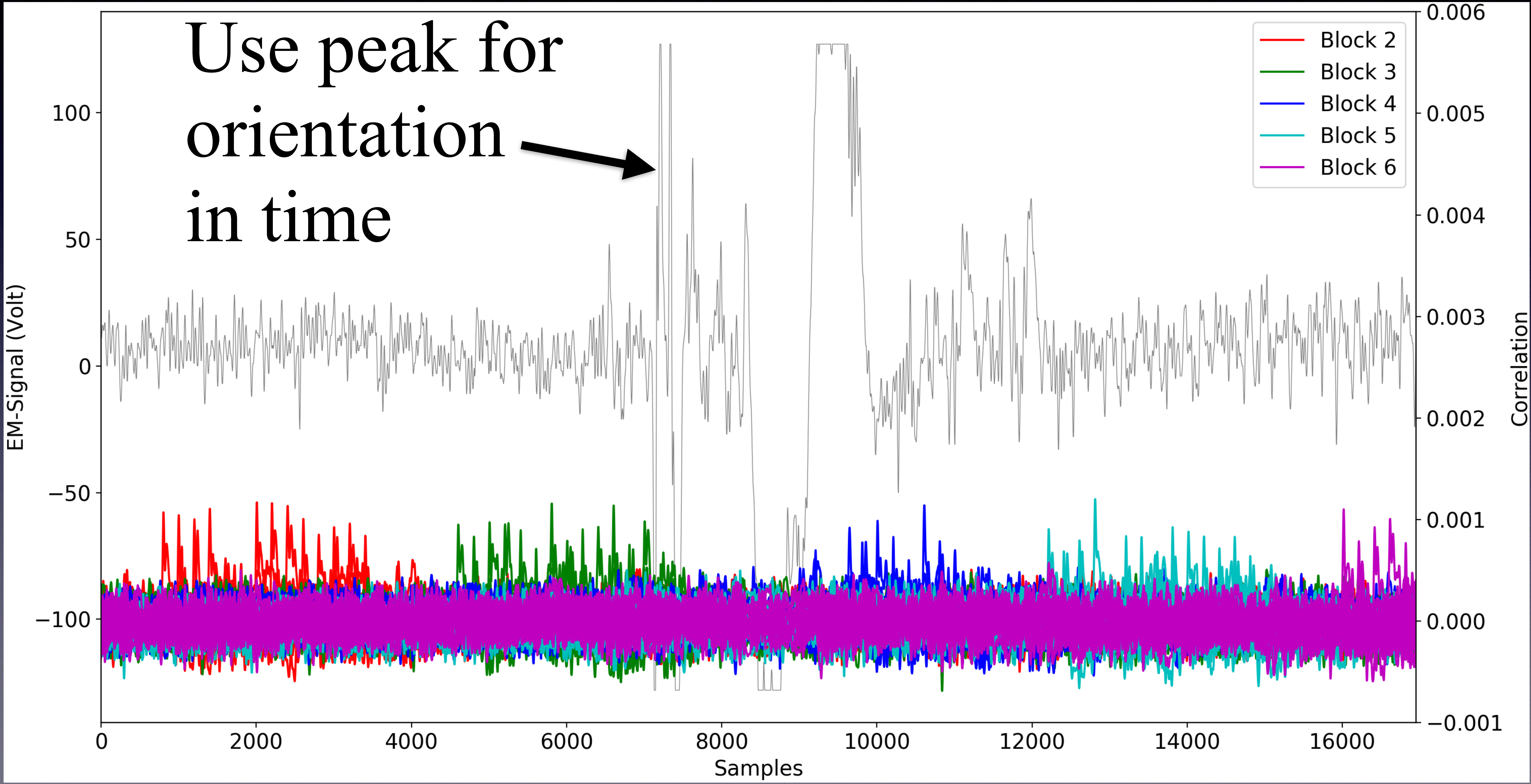

0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
−0.00025
−0.00050
−0.00075
Correlation
0
2500
5000
7500
10000
12500
15000
17500
Samples
Block 2
Block 3
Block 4
Block 5
Block 6

CPA

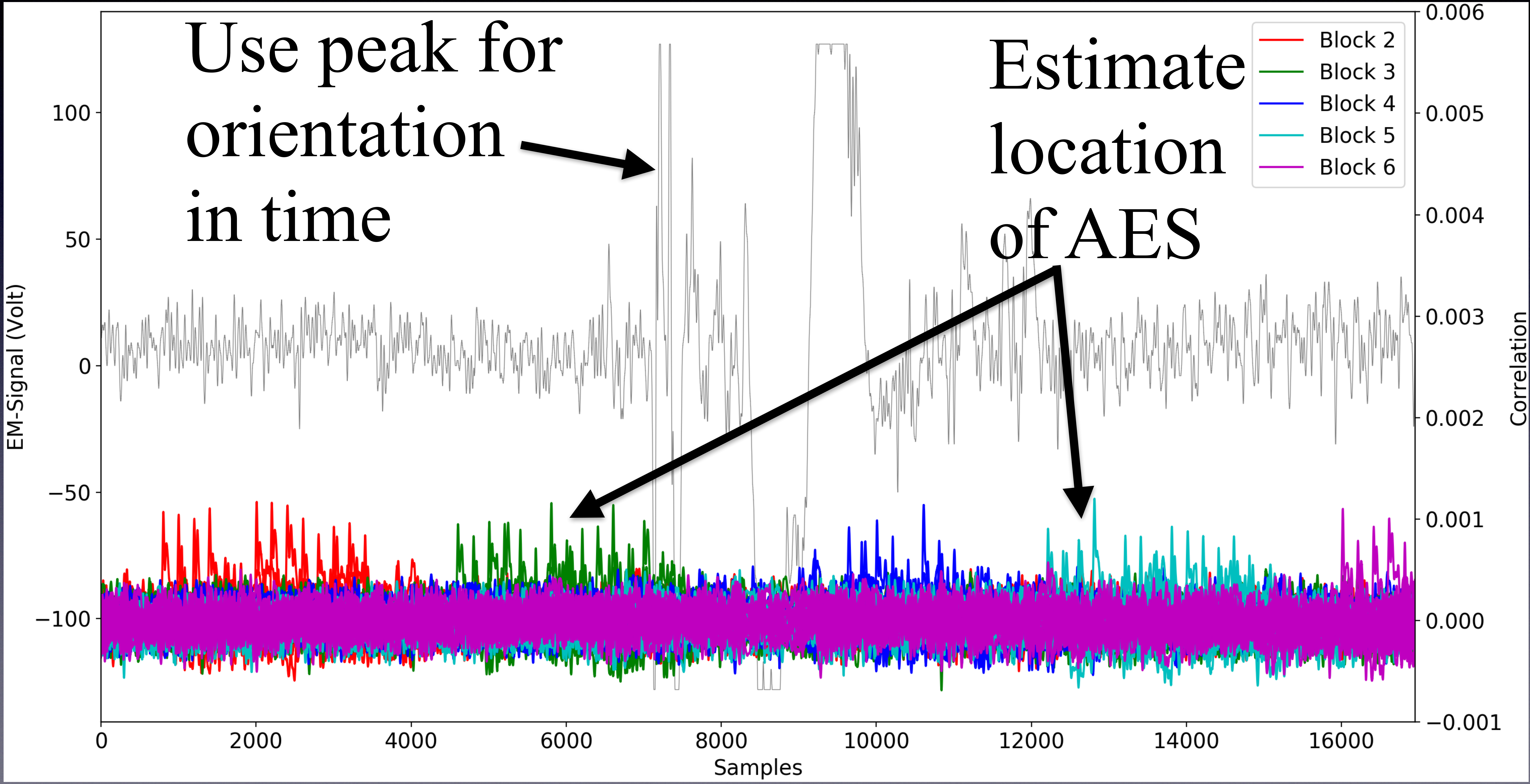


Block 2
Block 3
Block 4
Block 5
Block 6
EM-Signal (Volt)
100
50
0
−50
−100
Correlation
0.006
0.005
0.004
0.003
0.002
0.001
0.000
−0.001
0
2000
4000
6000
8000
10000
12000
14000
16000
Samples

CPA


Use peak for orientation in time
Block 2
Block 3
Block 4
Block 5
Block 6
EM-Signal (Volt)
Correlation
Samples
100
50
0
−50
−100
0.006
0.005
0.004
0.003
0.002
0.001
0.000
−0.001
0
2000
4000
6000
8000
10000
12000
14000
16000

CPA


Use peak for orientation in time
Estimate location of AES
Block 2
Block 3
Block 4
Block 5
Block 6
EM-Signal (Volt)
Correlation
Samples
100
50
0
−50
−100
0.006
0.005
0.004
0.003
0.002
0.001
0.000
−0.001
0
2000
4000
6000
8000
10000
12000
14000
16000

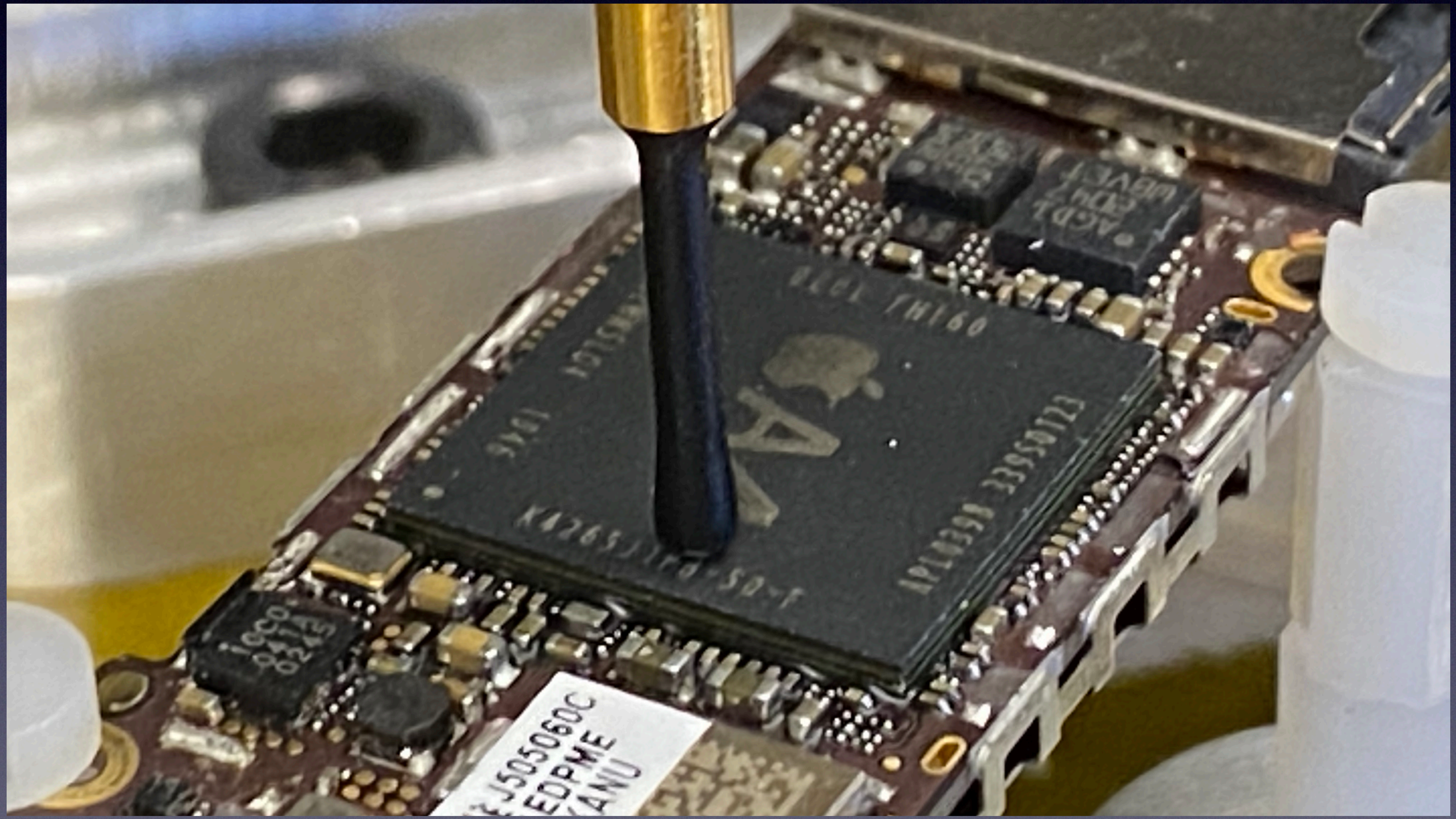
Full chip scan

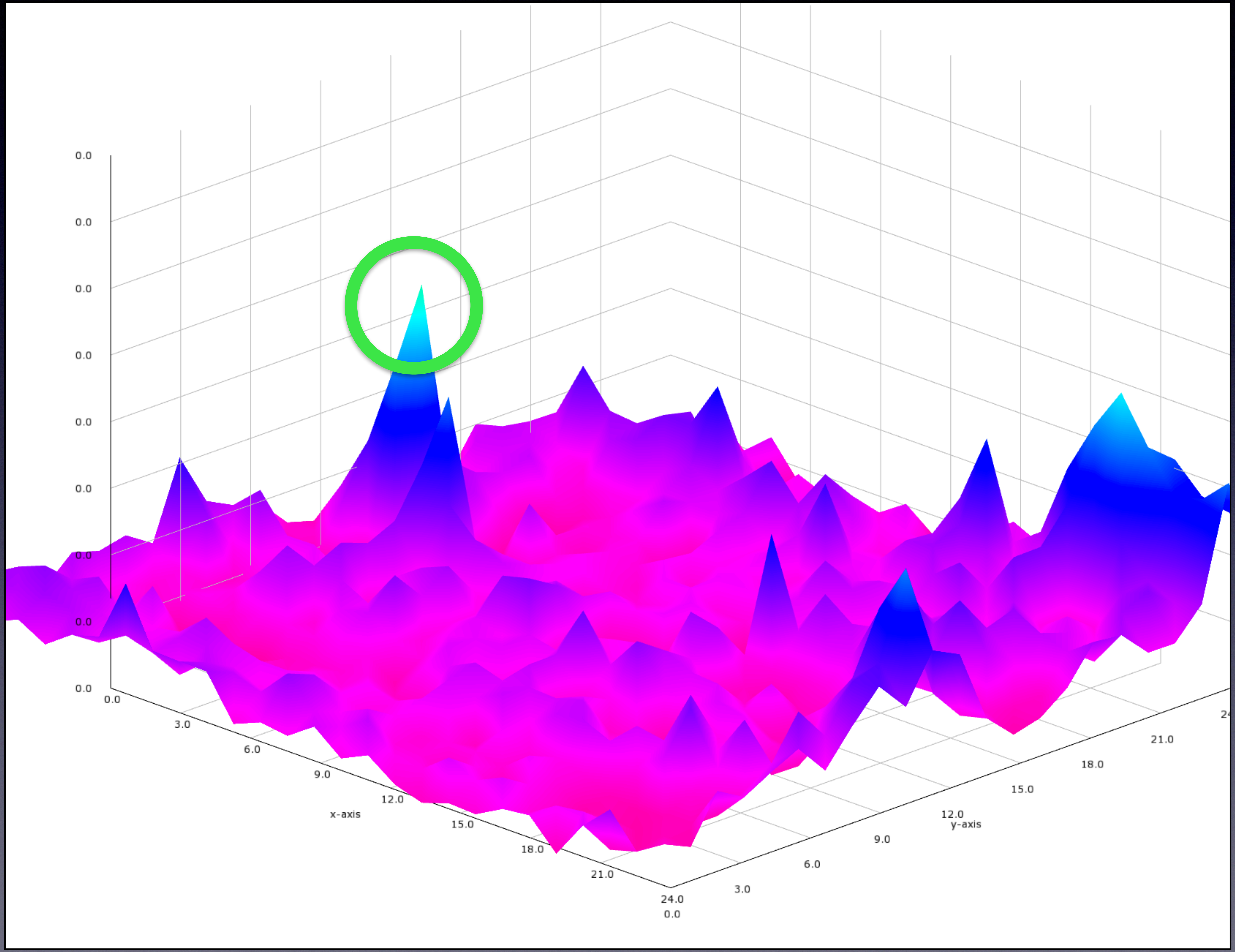
- Divide chip in grid of 24x24 squares
- Record traces on each position
- Correlate with 128bit model on each position (requires *working* power model)
- Find optimal attack spot

Full chip scan (problems)

- Very few traces can be recorded
  - Major limitation is disk space
- My constraints:
  - 6TB (compressed) traces
  - 150'000 traces - 40'000 points of 4 blocks AES
    (in contrast to: 100'000'000 traces - 80'000 points of 8 blocks AES)
- Not very reliable results - only vague orientation

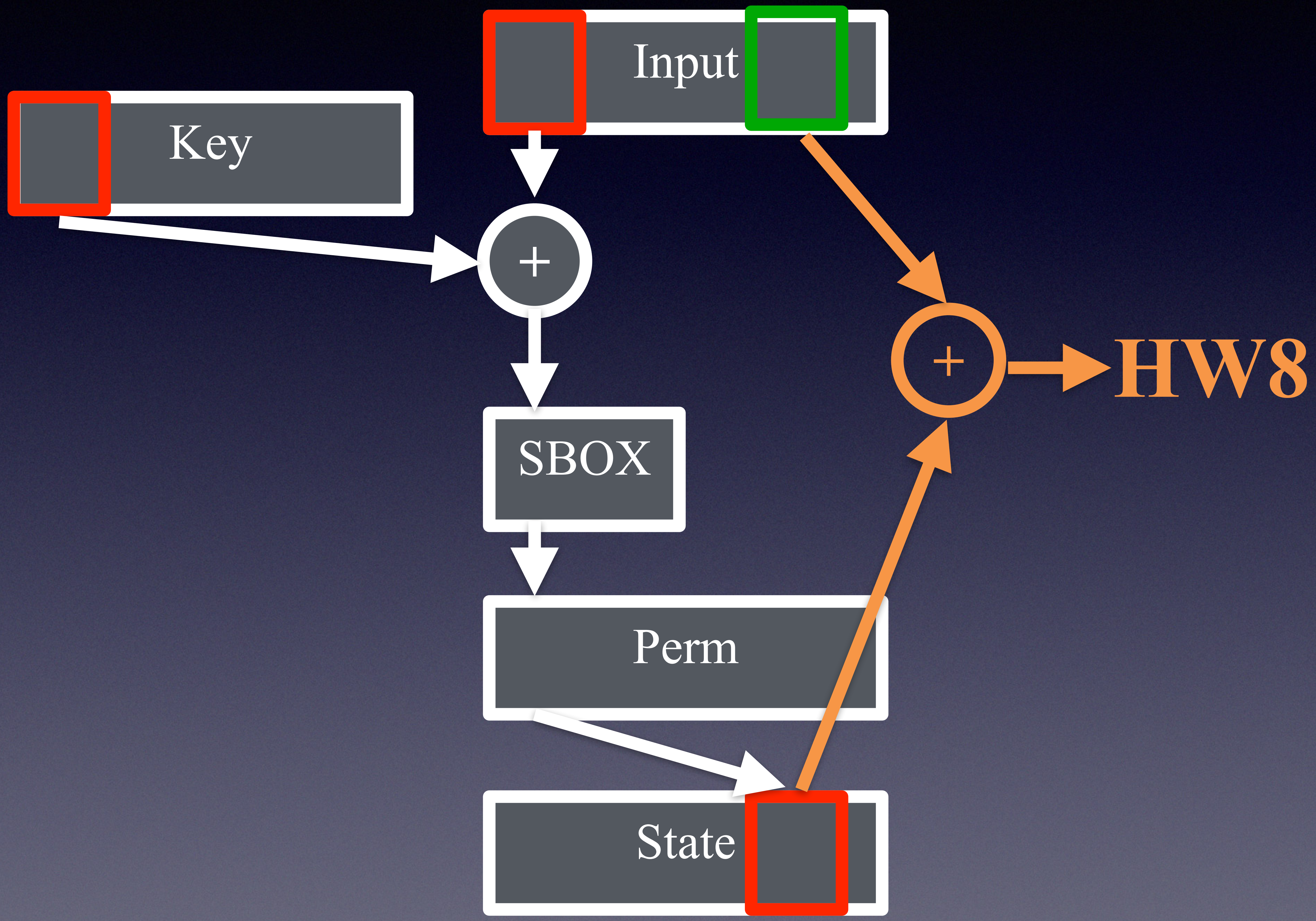
Full chip scan





Smaller powermodel

- HW128 model too large for practical attack
- Test HW8 powermodel
- Record AES decryption
- Target last round of decryption (here: modeled with encryption)

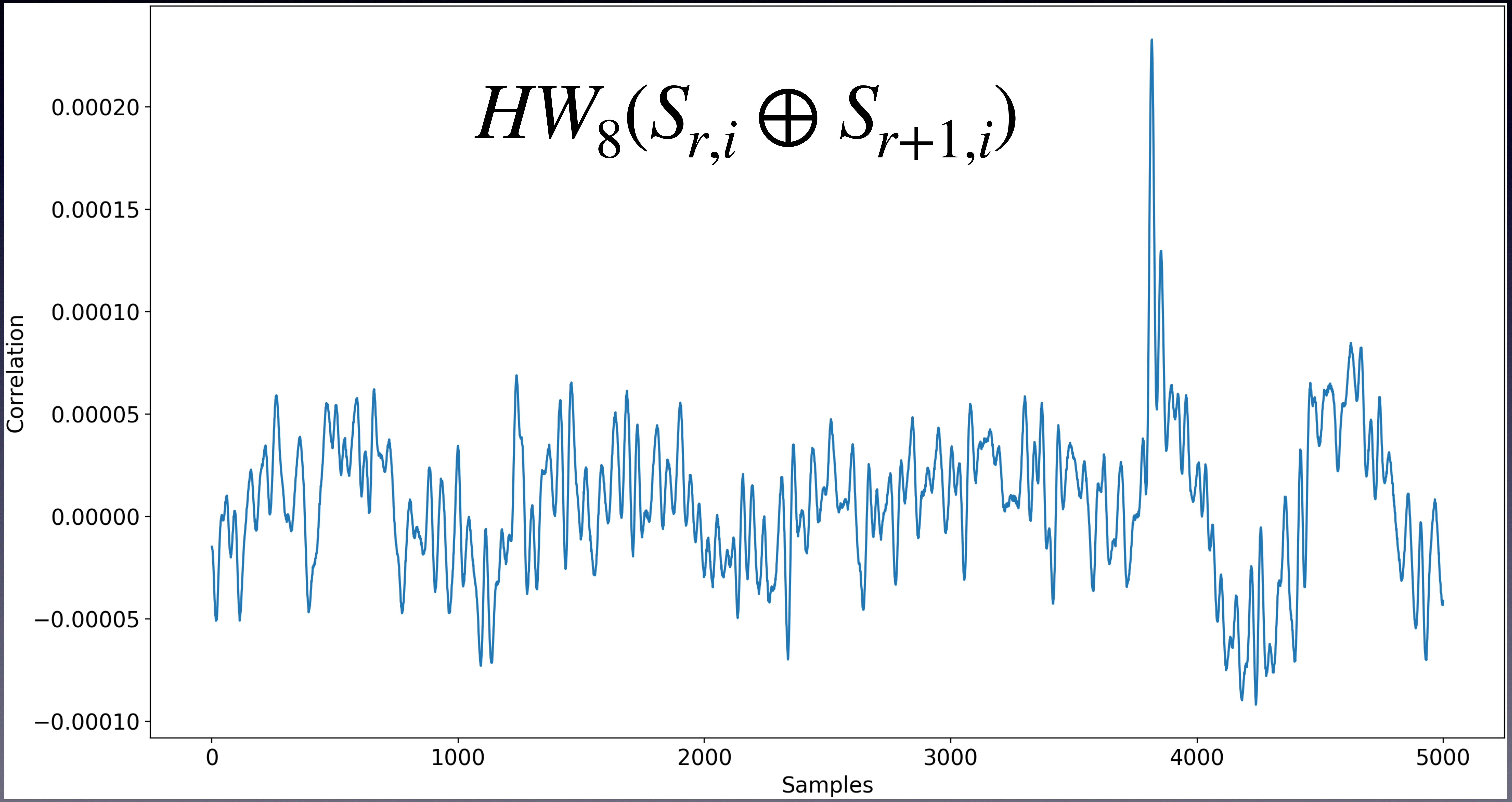


CPA attack

- Record 8 block AES256-CBC decryption with **random** ciphertext (500'000'000 traces -> ~1 Week)
  - UART transfer slow -> Send seed, generate ciphertext on device
  - Generate same ciphertext on PC
- Need plaintext for attack (last round of decryption)
  - Use USB to transfer data to device, decrypt, send back (~1 Week decryption)
- Run correlation on GPU (~1 Week)

Known key correlation (time)

- Test with known key
- Estimate leaking point-in-time for correct key byte
- Compare with wrong key bytes
- Then, correlate over keys...



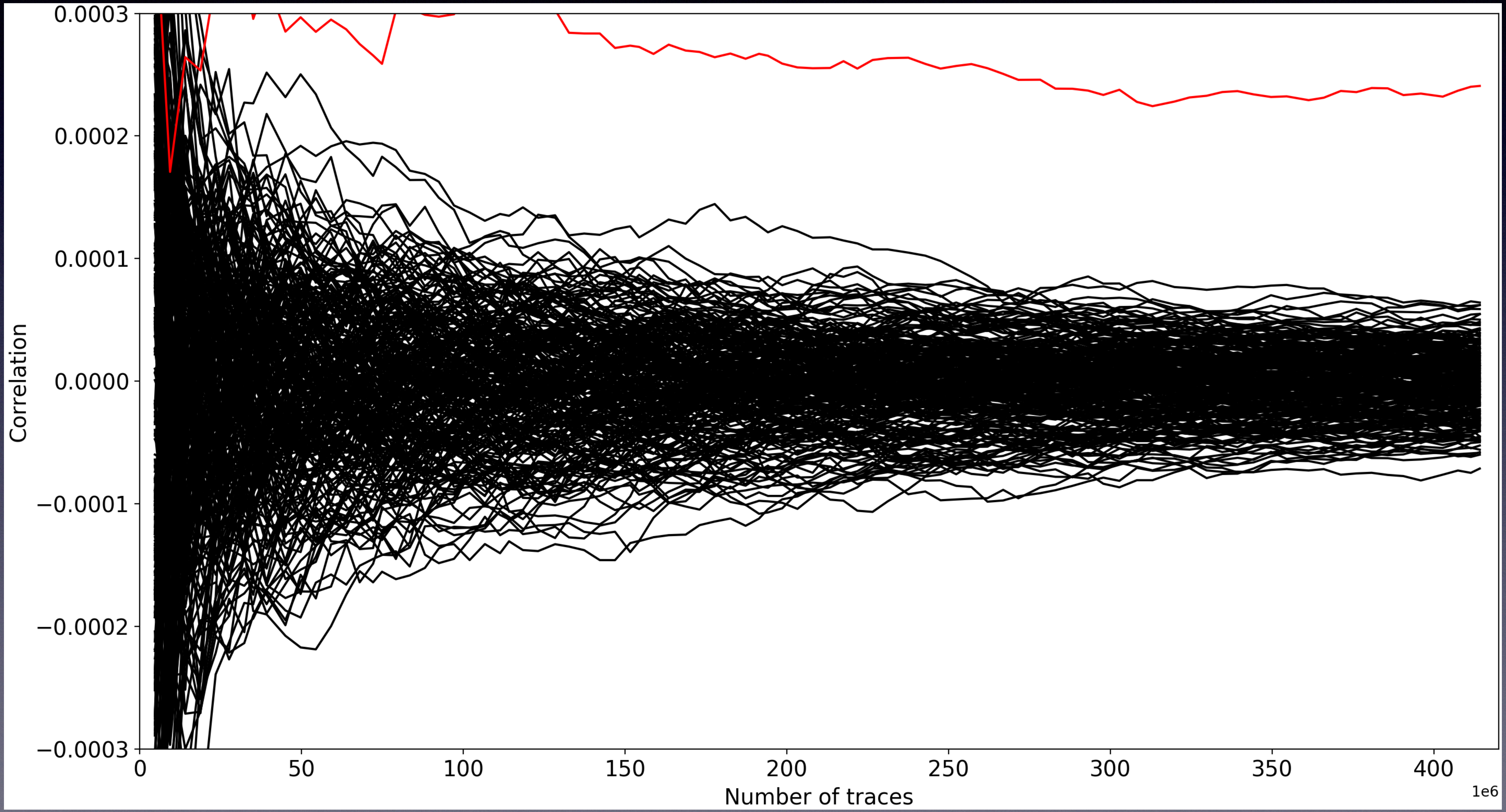

Correlation
Key
0.00015
0.00010
0.00005
0.00000
−0.00005
−0.00010
0.00025
0.00020
0
50
100
150
200
250


Target Key
clearly visible!
Correlation
Key

How many traces
are needed for an attack?

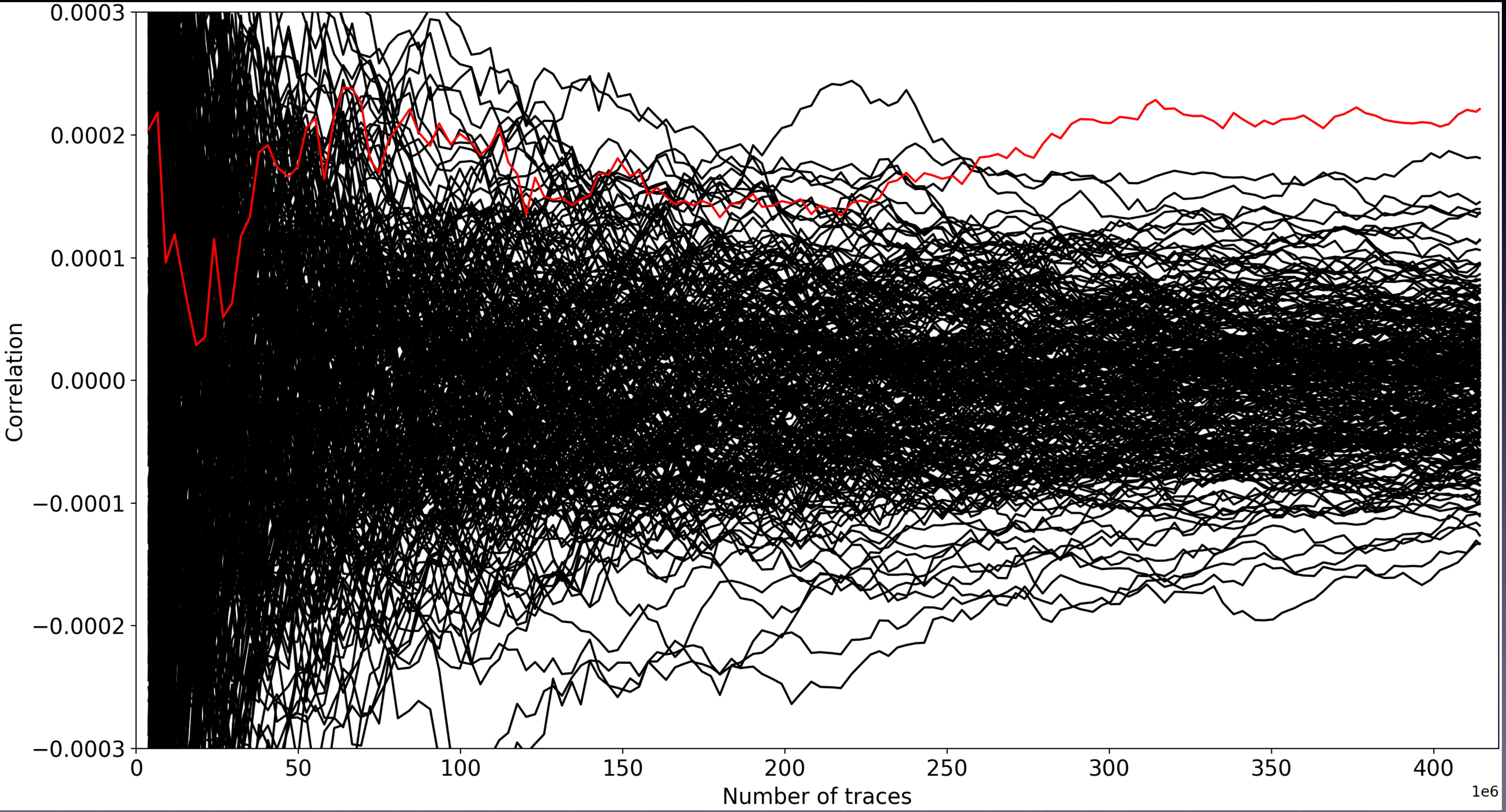
Number of Traces

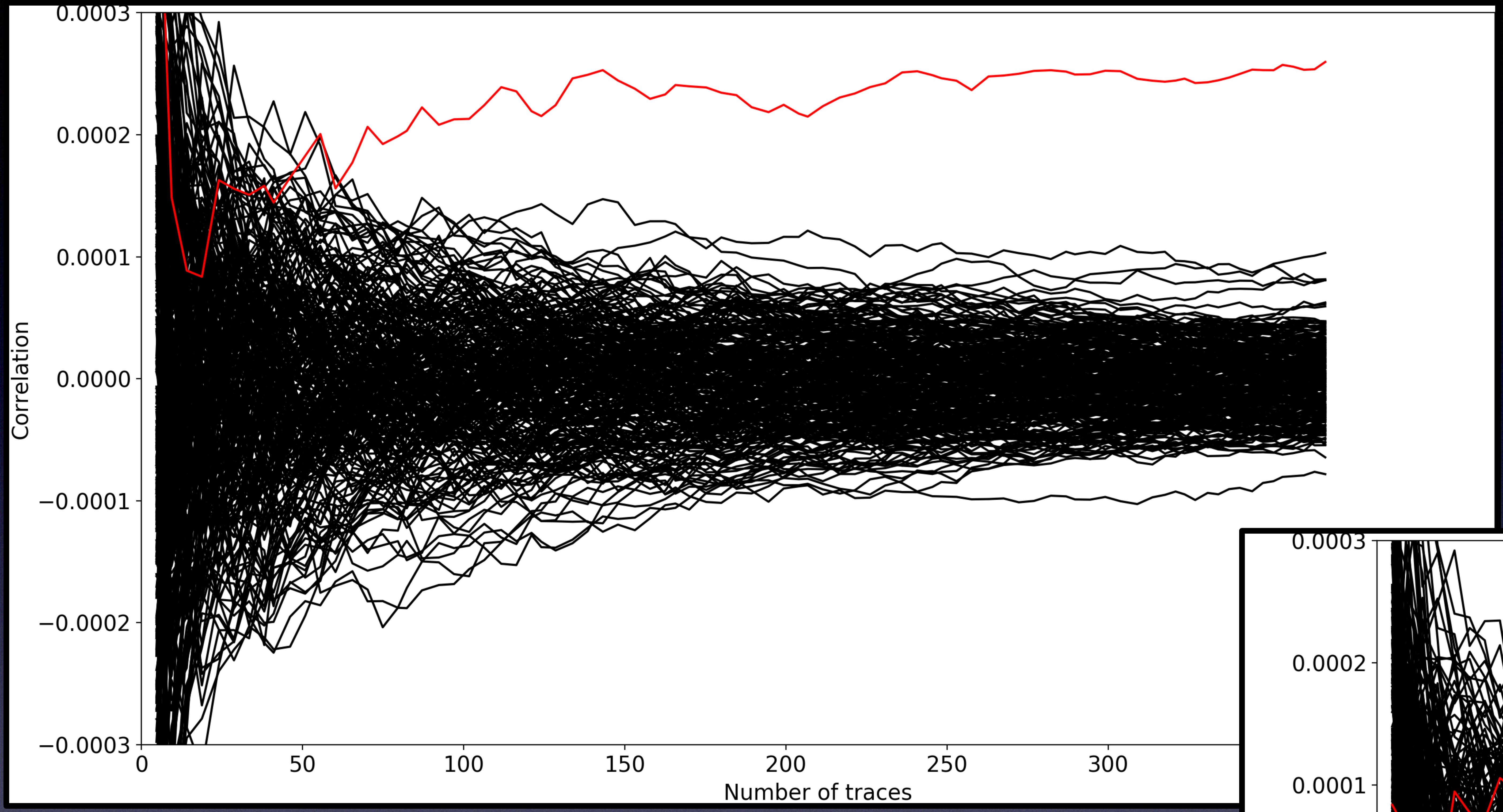
- Varies a lot!
- Some bytes need 30.000.000 Traces

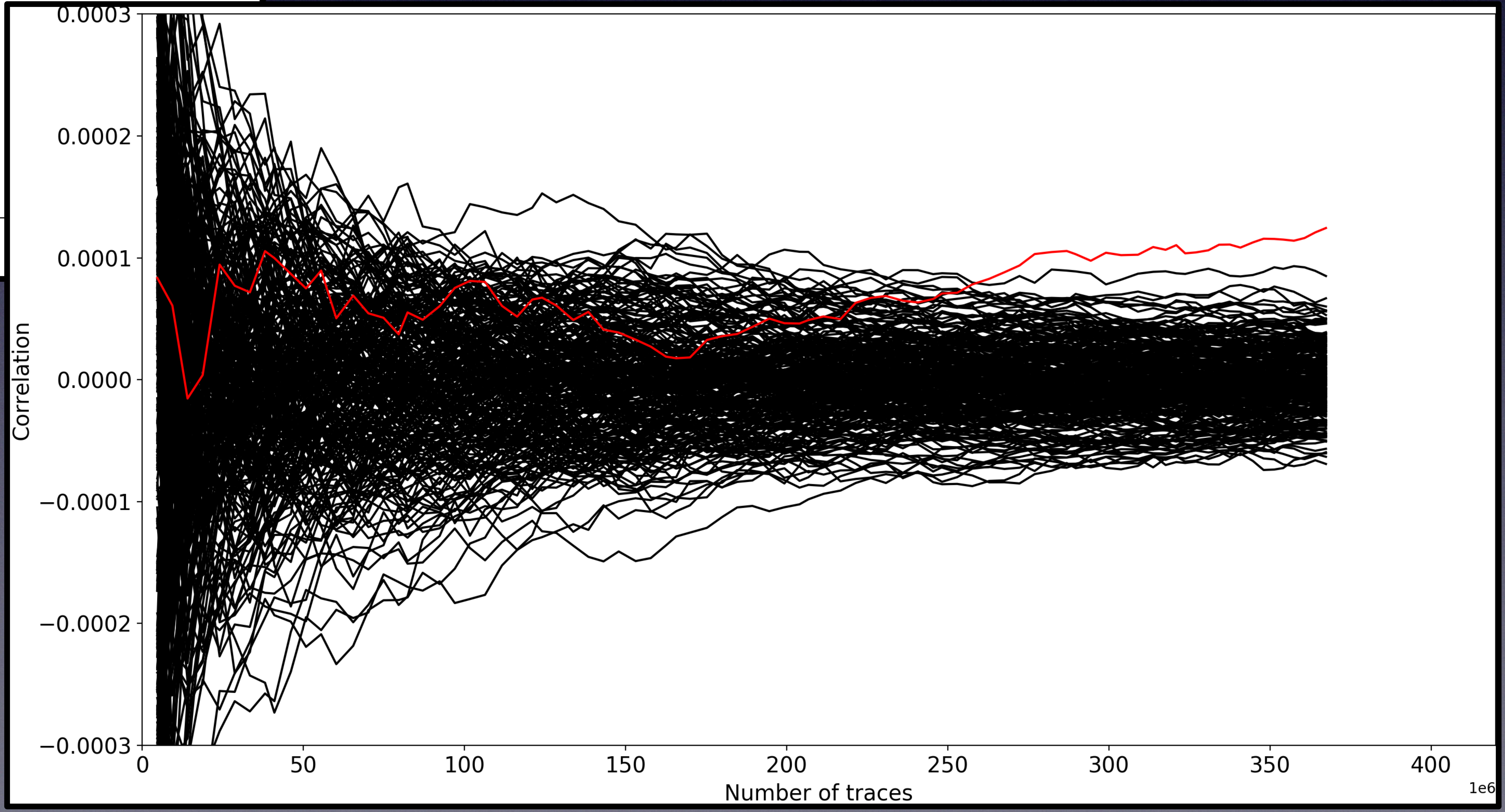


Number of Traces

- Varies a lot!
- Some bytes need 300.000.000 Traces

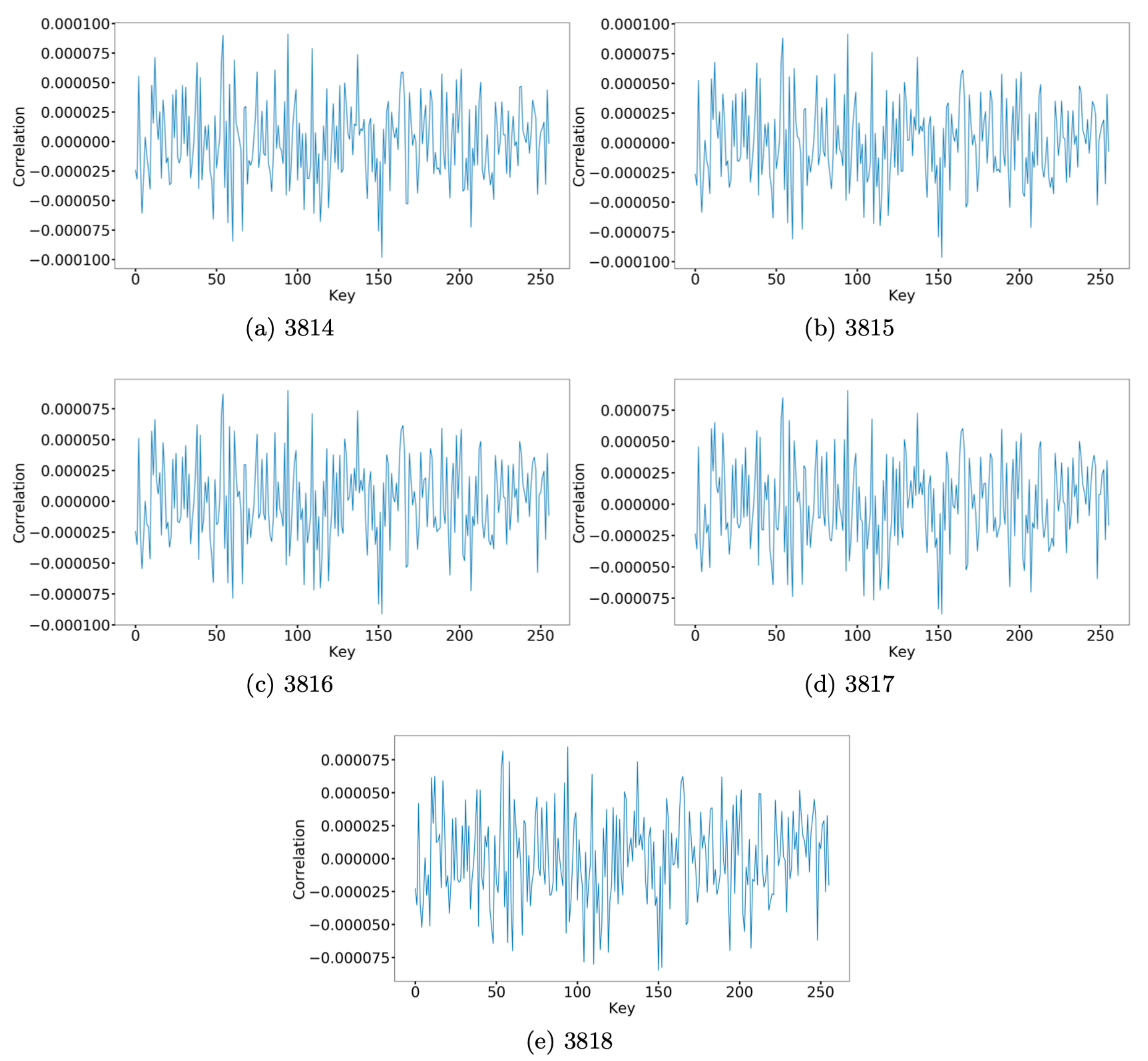


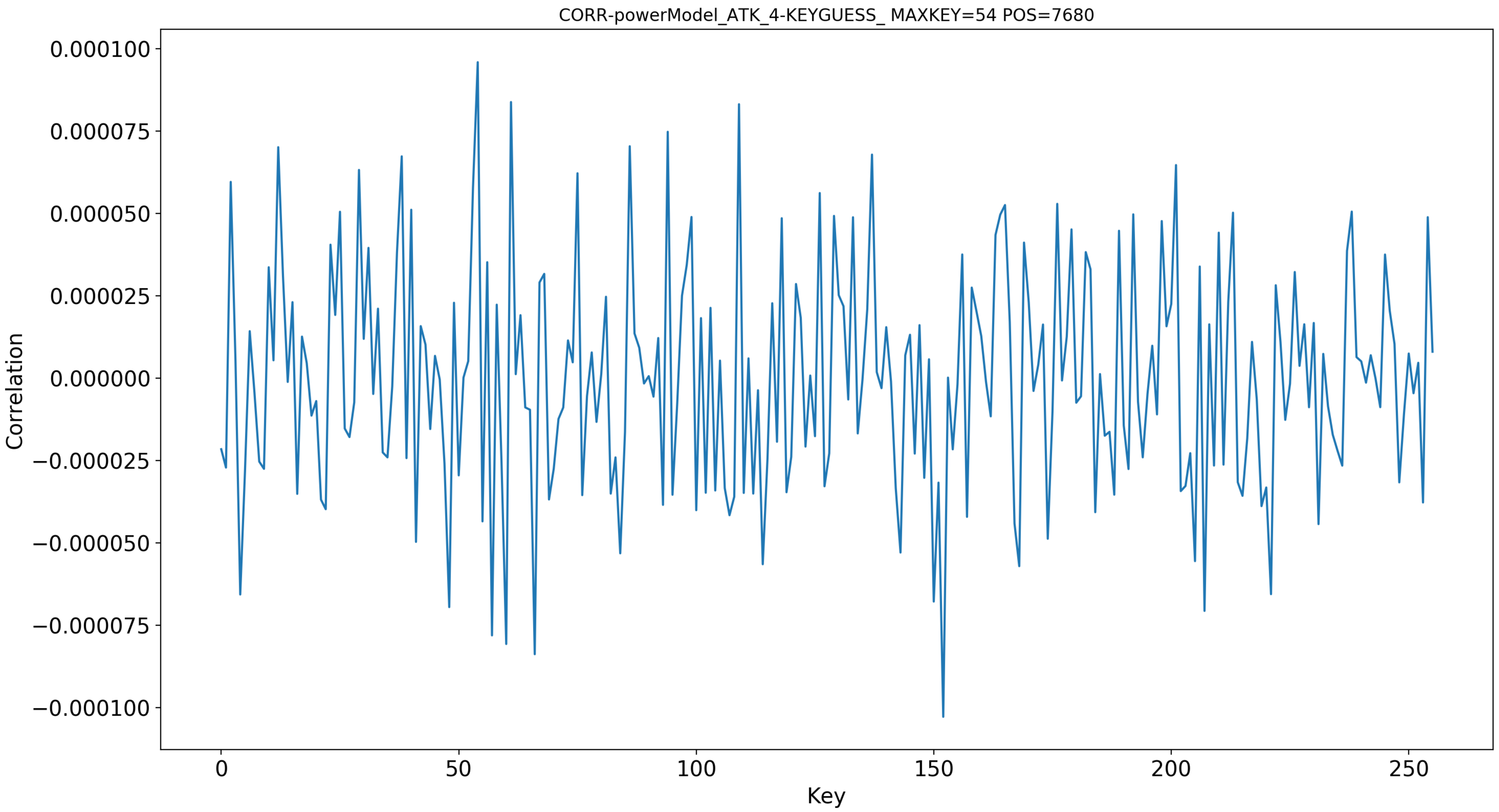

0.0003
0.0002
0.0001
0.0000
−0.0001
−0.0002
−0.0003
Correlation
0
50
100
150
200
250
300
Number of traces

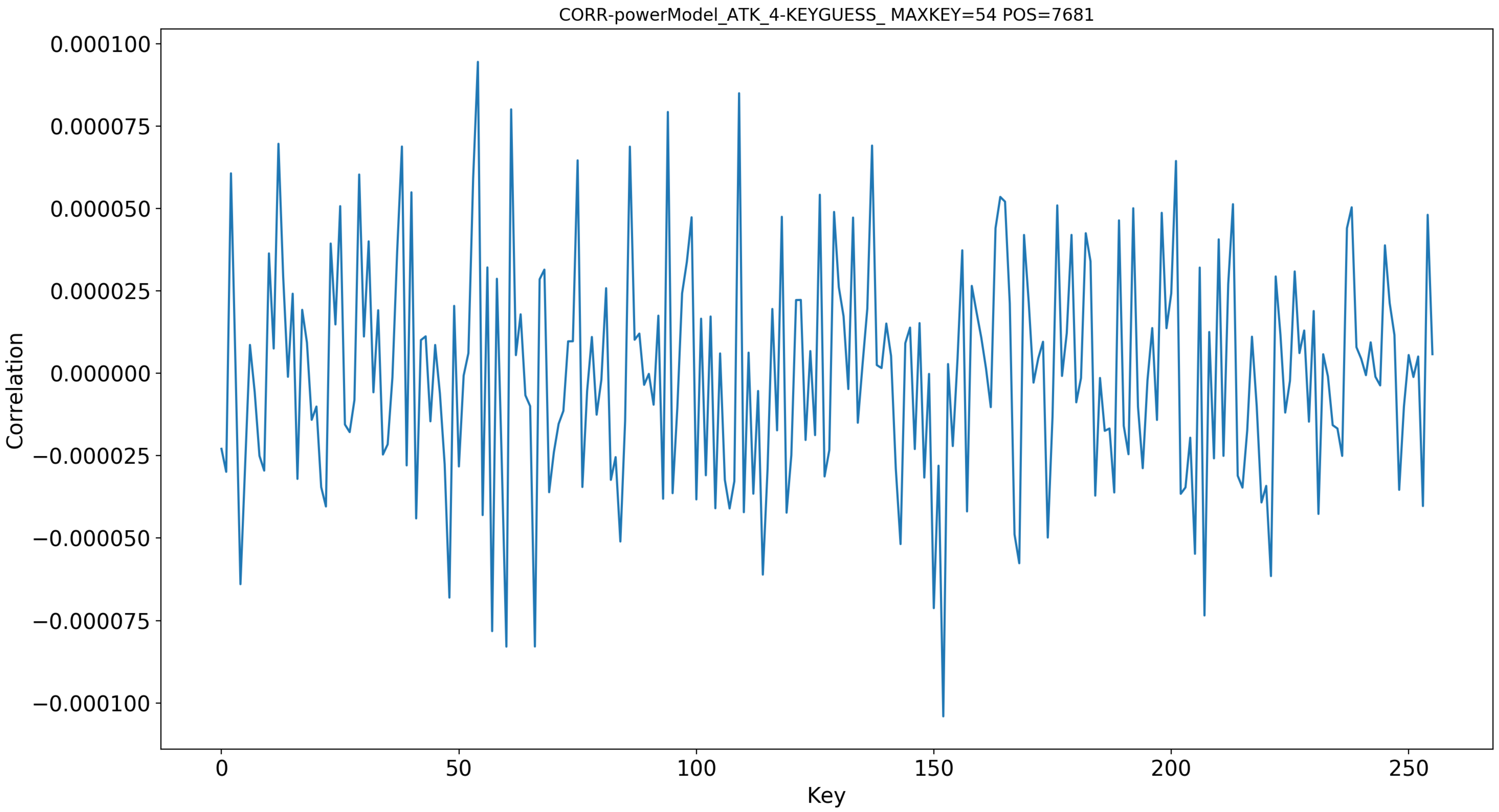

0.0003
0.0002
0.0001
0.0000
−0.0001
−0.0002
−0.0003
Correlation
0
50
100
150
200
250
300
350
400
1e6
Number of traces

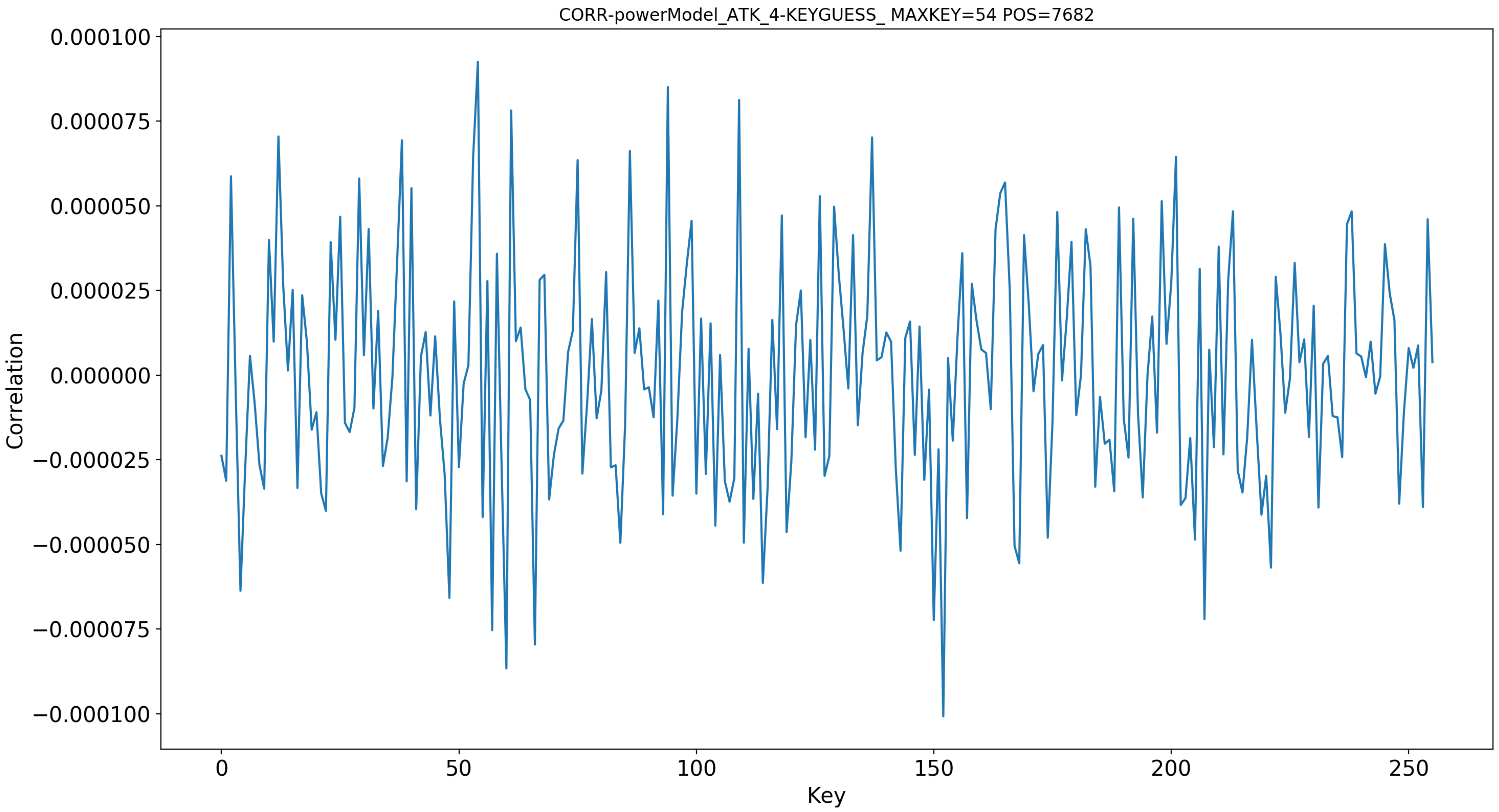
(noisy) Key recovery strategies

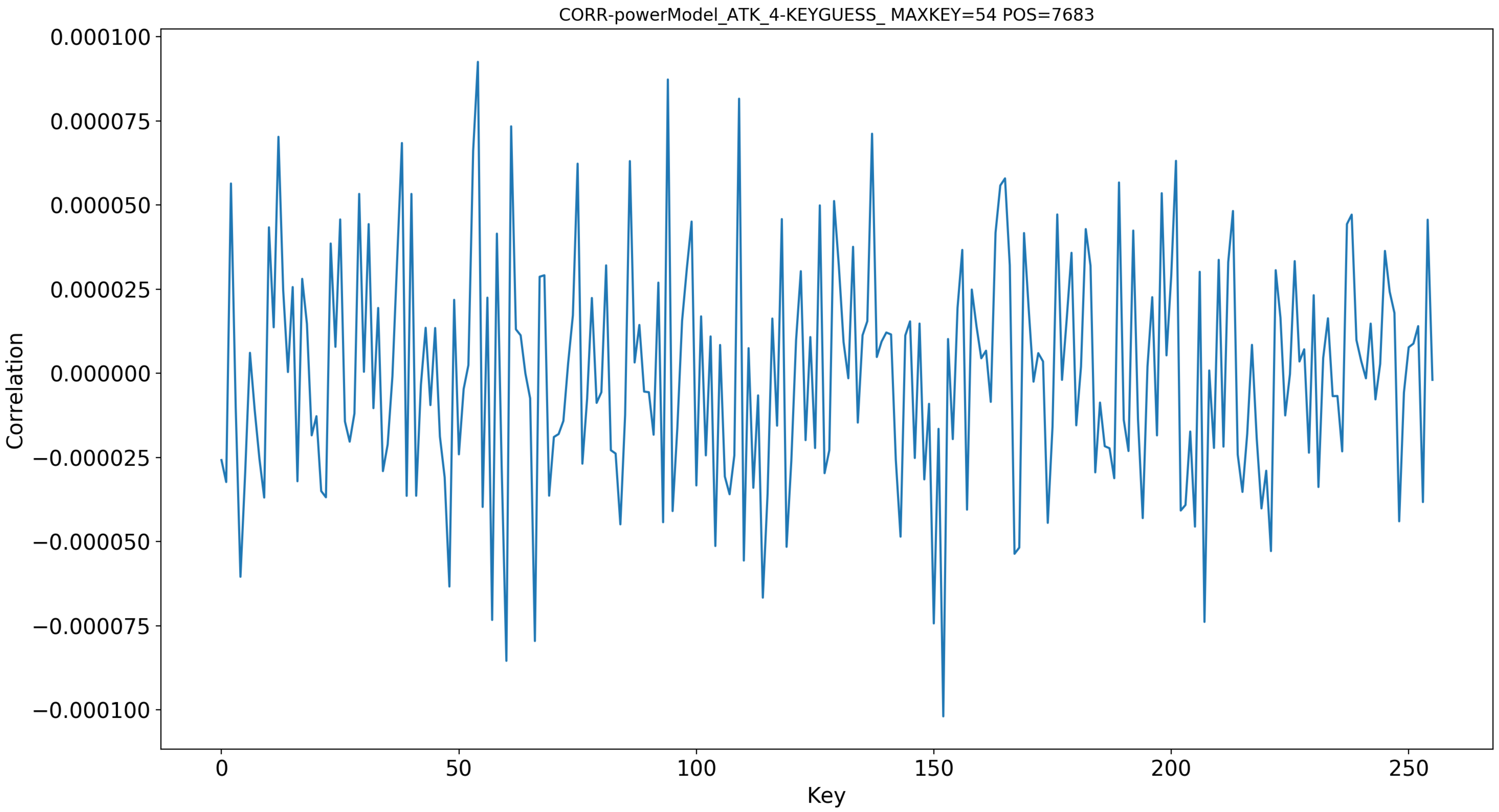
- Check nearby points in time

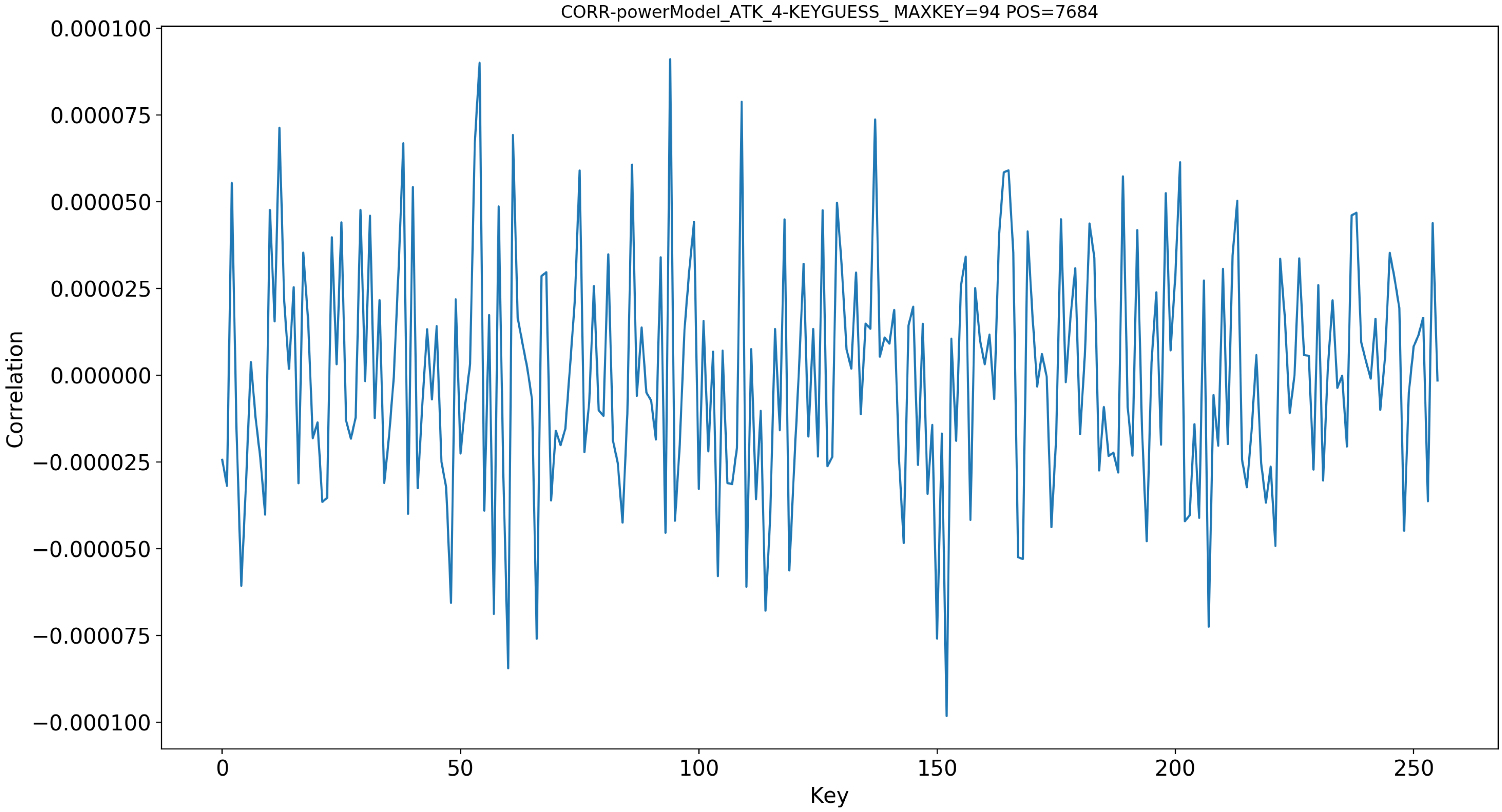


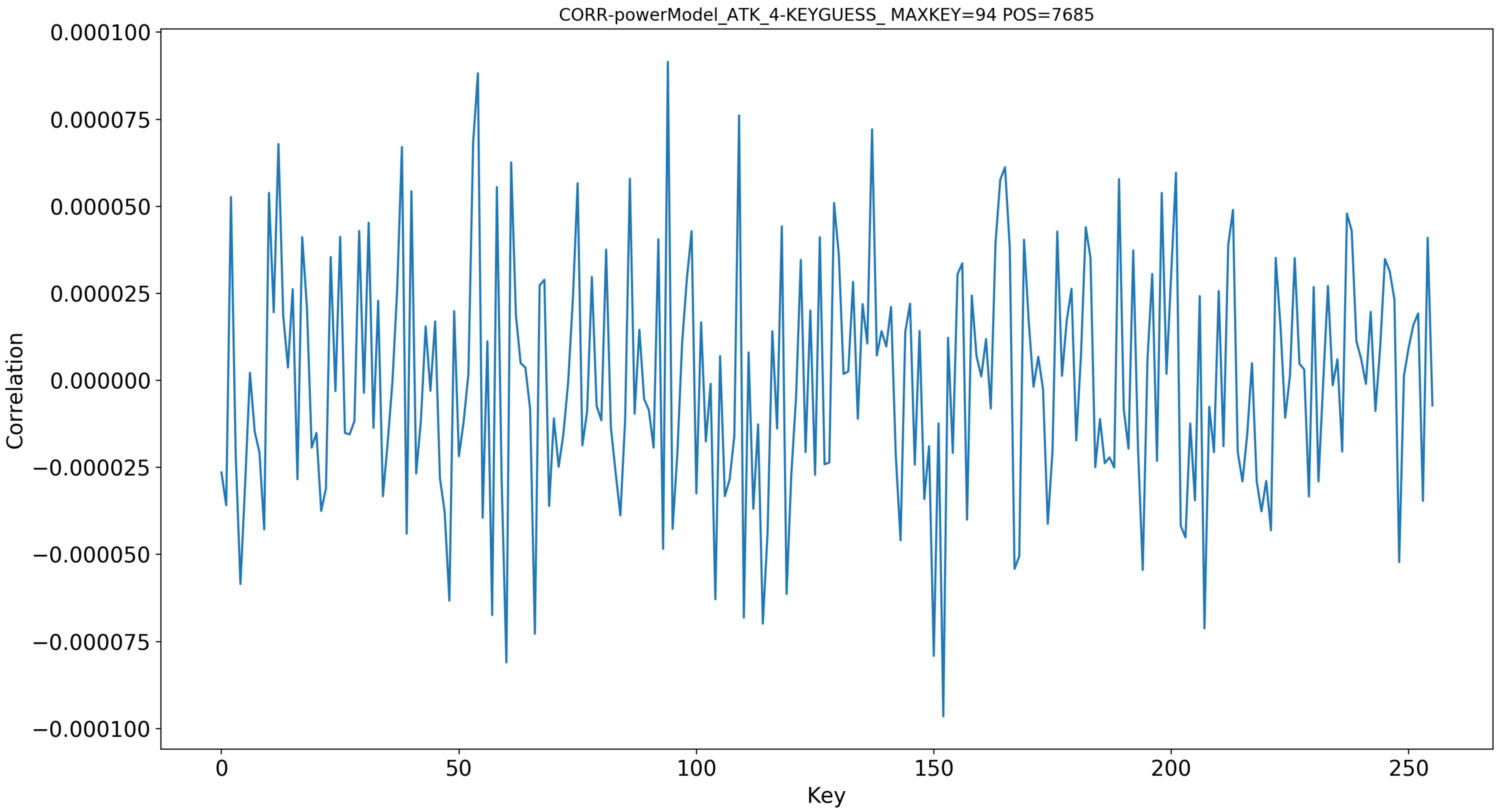

CORR-powerModel_ATK_4-KEYGUESS_ MAXKEY=54 POS=7680
0.000100
0.000075
0.000050
0.000025
0.000000
−0.000025
−0.000050
−0.000075
−0.000100
Correlation
0
50
100
150
200
250
Key

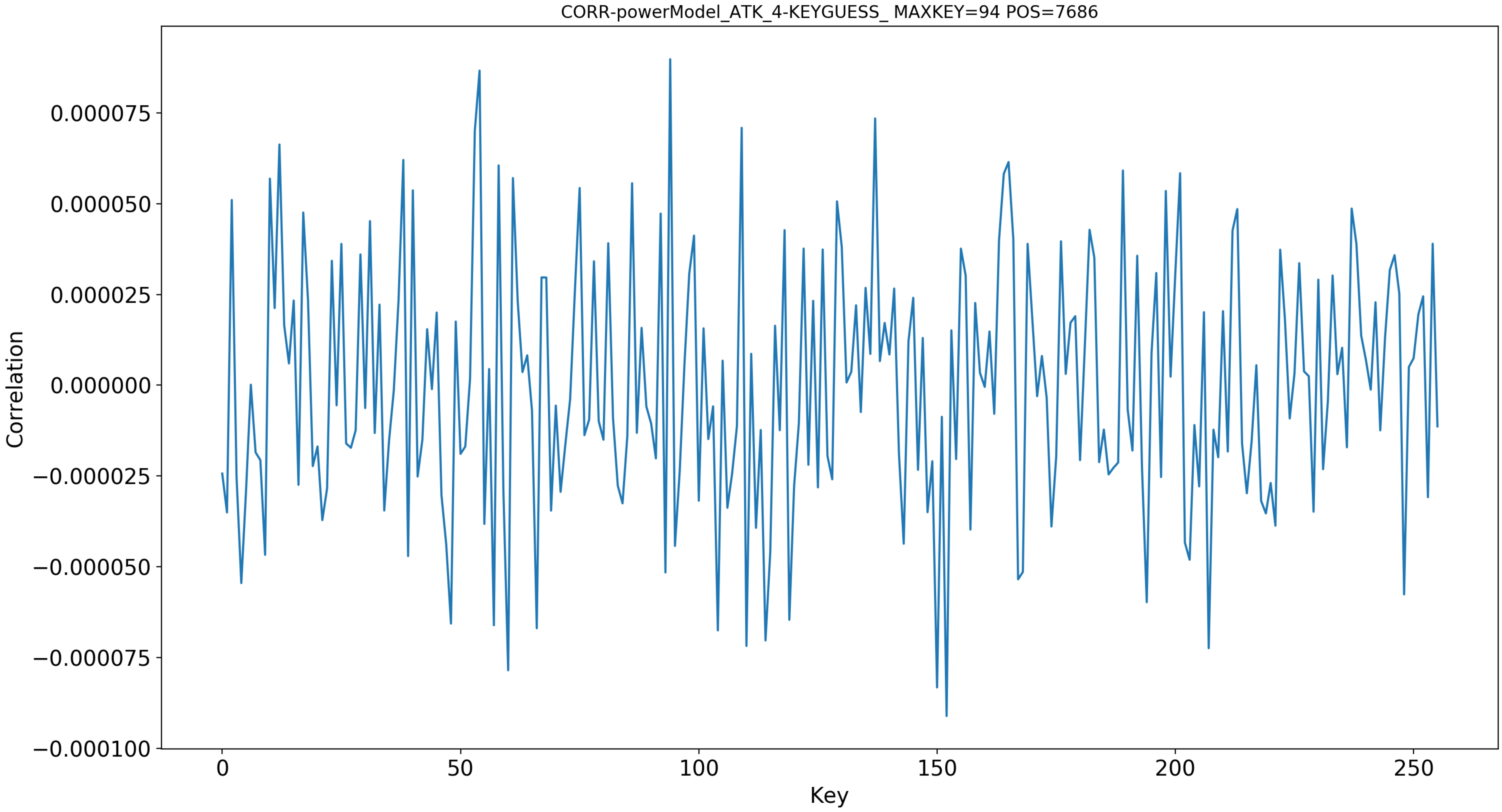

CORR-powerModel_ATK_4-KEYGUESS_ MAXKEY=54 POS=7681
Correlation
Key
0.000100
0.000075
0.000050
0.000025
0.000000
−0.000025
−0.000050
−0.000075
−0.000100
0
50
100
150
200
250

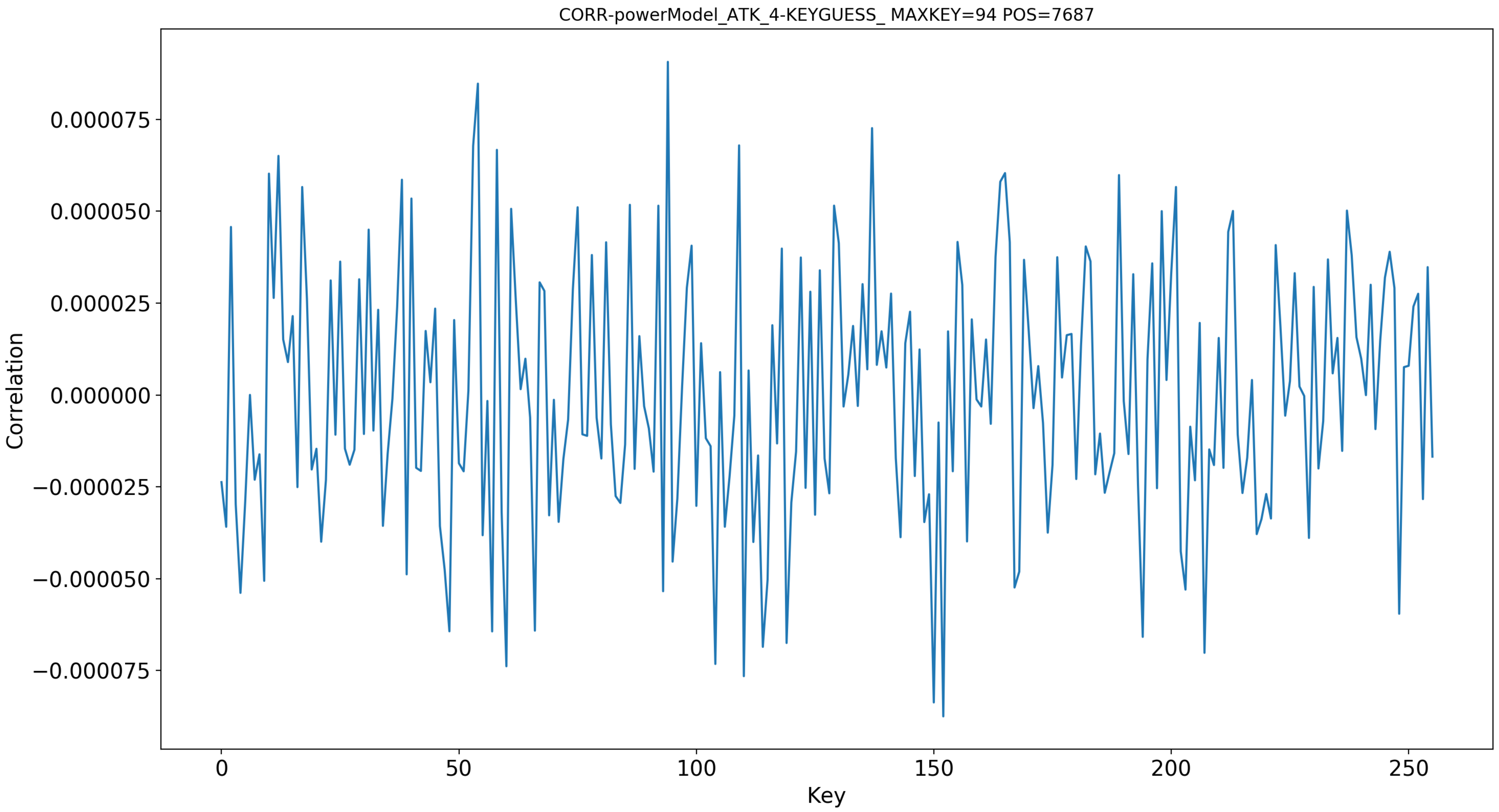

CORR-powerModel_ATK_4-KEYGUESS_ MAXKEY=54 POS=7682
Correlation
0.000100
0.000075
0.000050
0.000025
0.000000
−0.000025
−0.000050
−0.000075
−0.000100
0
50
100
150
200
250
Key

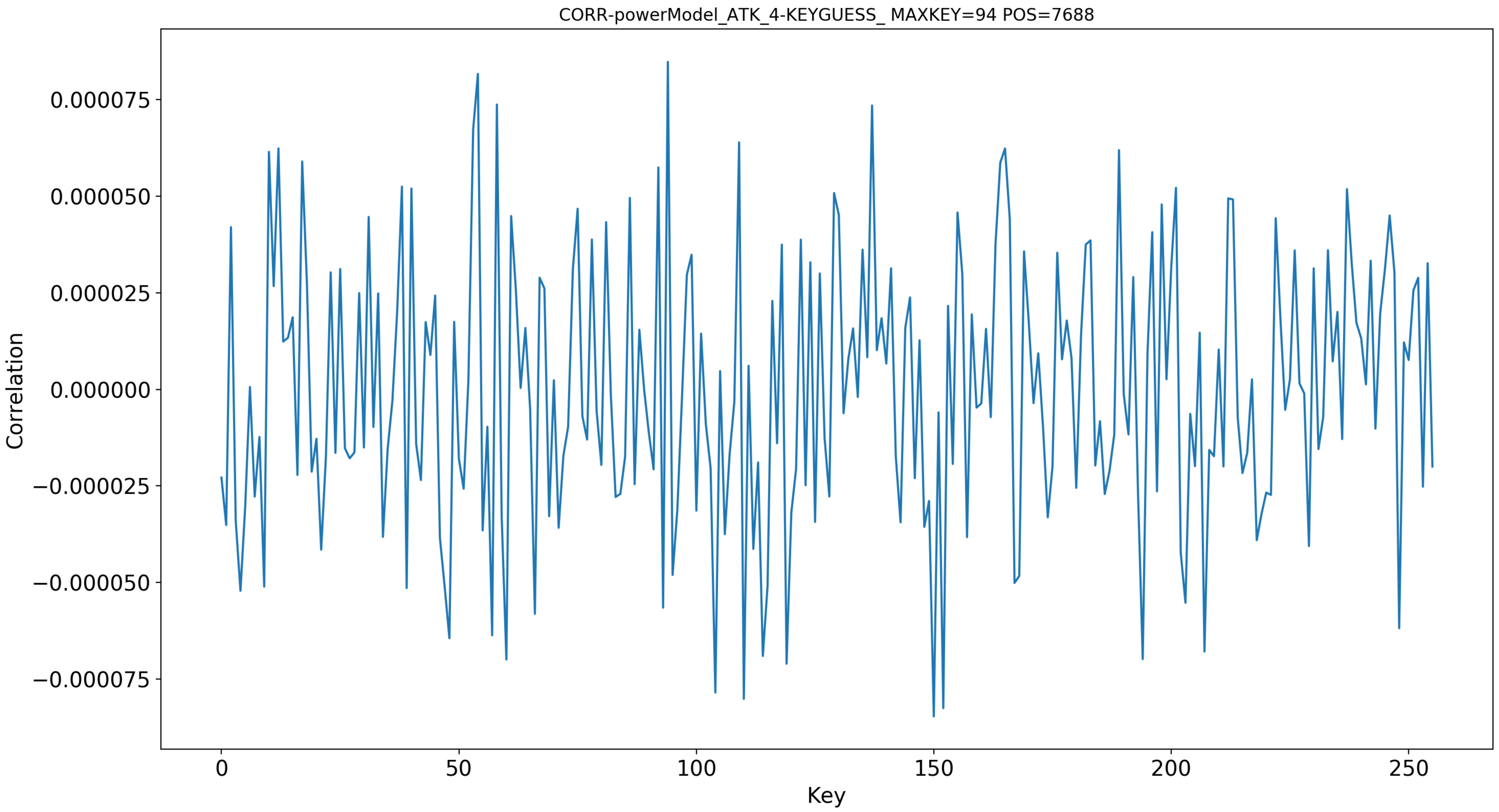

CORR-powerModel_ATK_4-KEYGUESS_ MAXKEY=54 POS=7683
Correlation
Key
0.000100
0.000075
0.000050
0.000025
0.000000
−0.000025
−0.000050
−0.000075
−0.000100
0
50
100
150
200
250

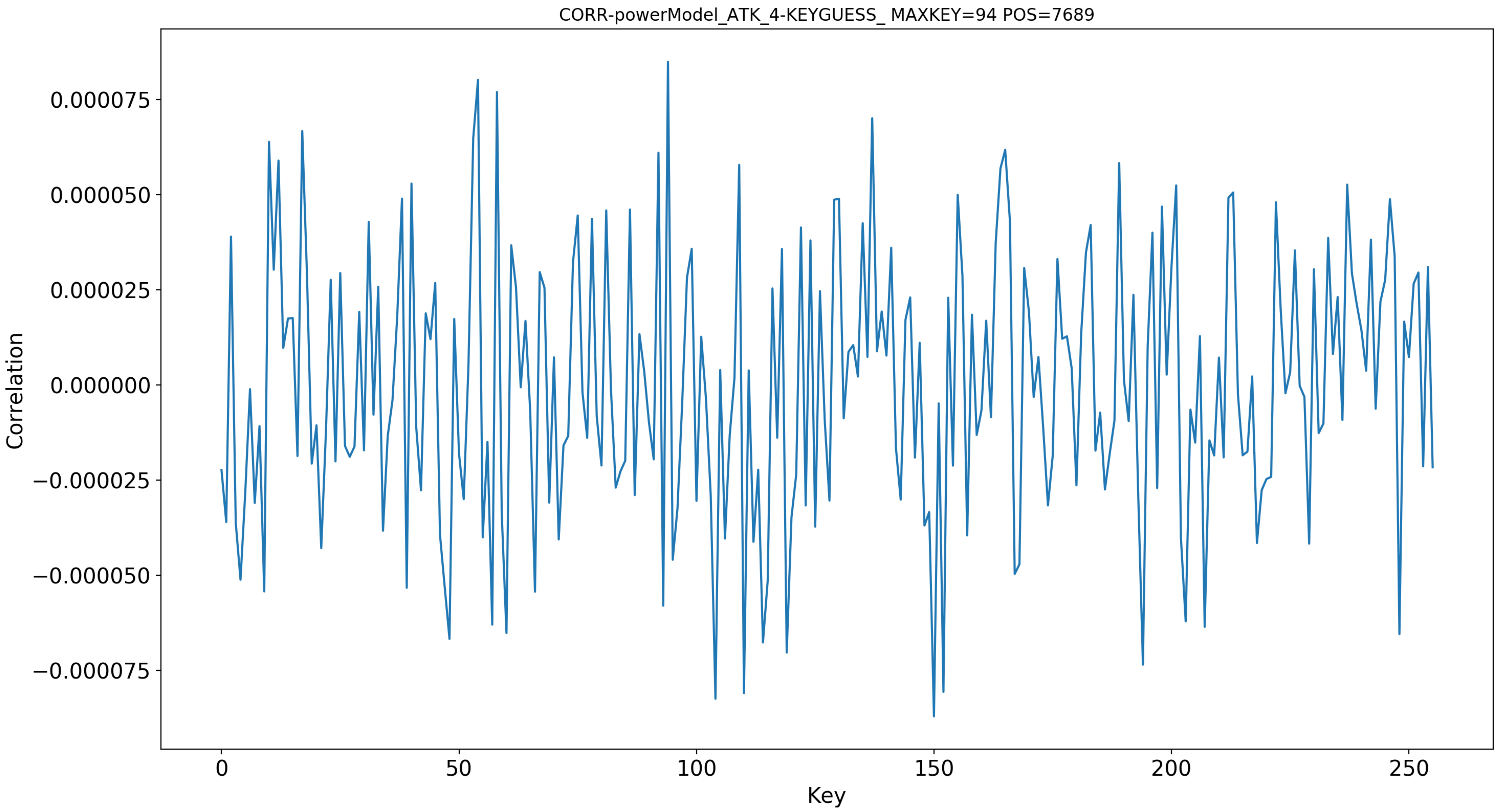

CORR-powerModel_ATK_4-KEYGUESS_ MAXKEY=94 POS=7684
Correlation
Key
0.000100
0.000075
0.000050
0.000025
0.000000
−0.000025
−0.000050
−0.000075
−0.000100
0
50
100
150
200
250

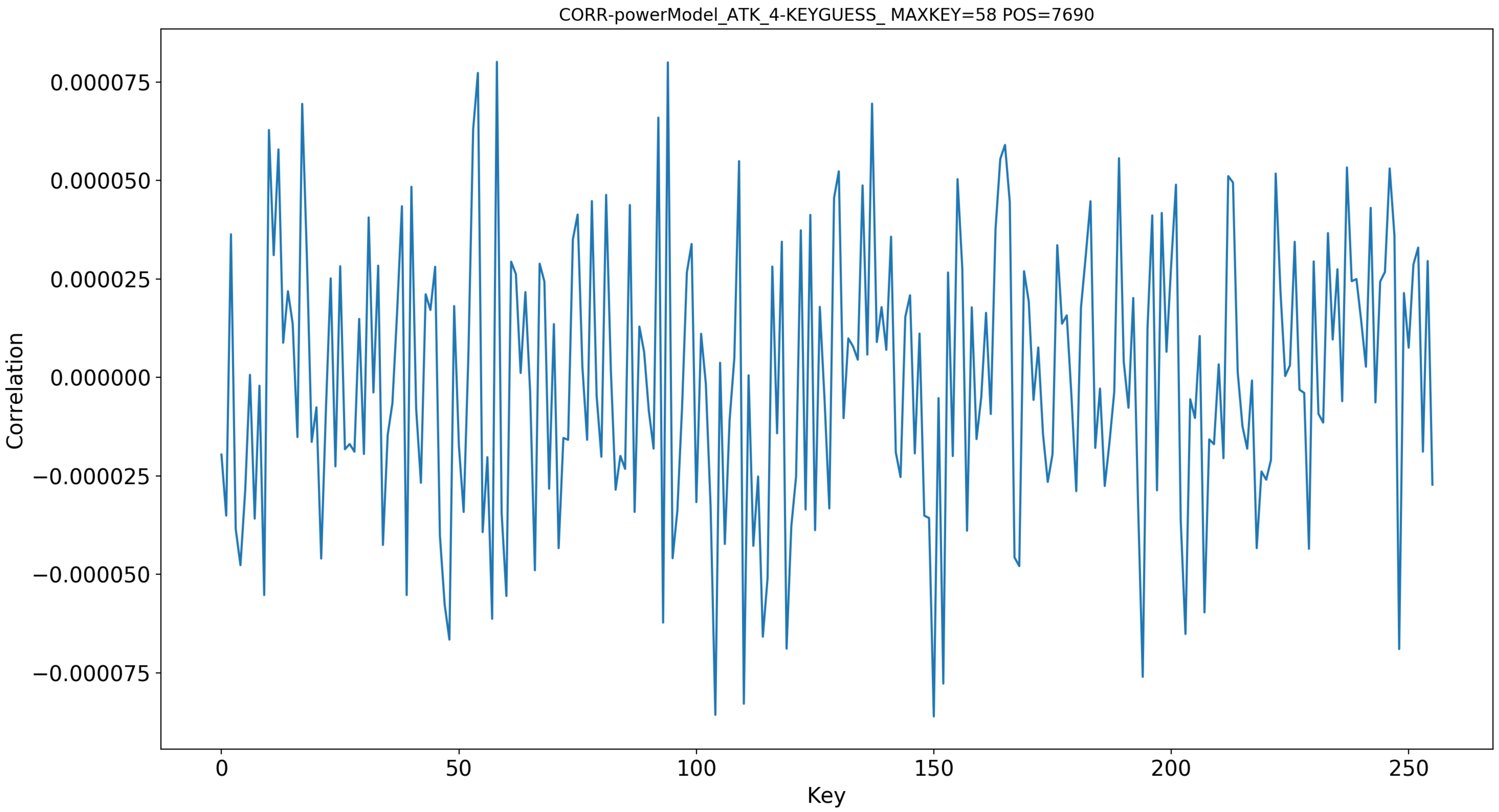

CORR-powerModel_ATK_4-KEYGUESS_ MAXKEY=94 POS=7685
Correlation
Key
0.000100
0.000075
0.000050
0.000025
0.000000
−0.000025
−0.000050
−0.000075
−0.000100
0
50
100
150
200
250

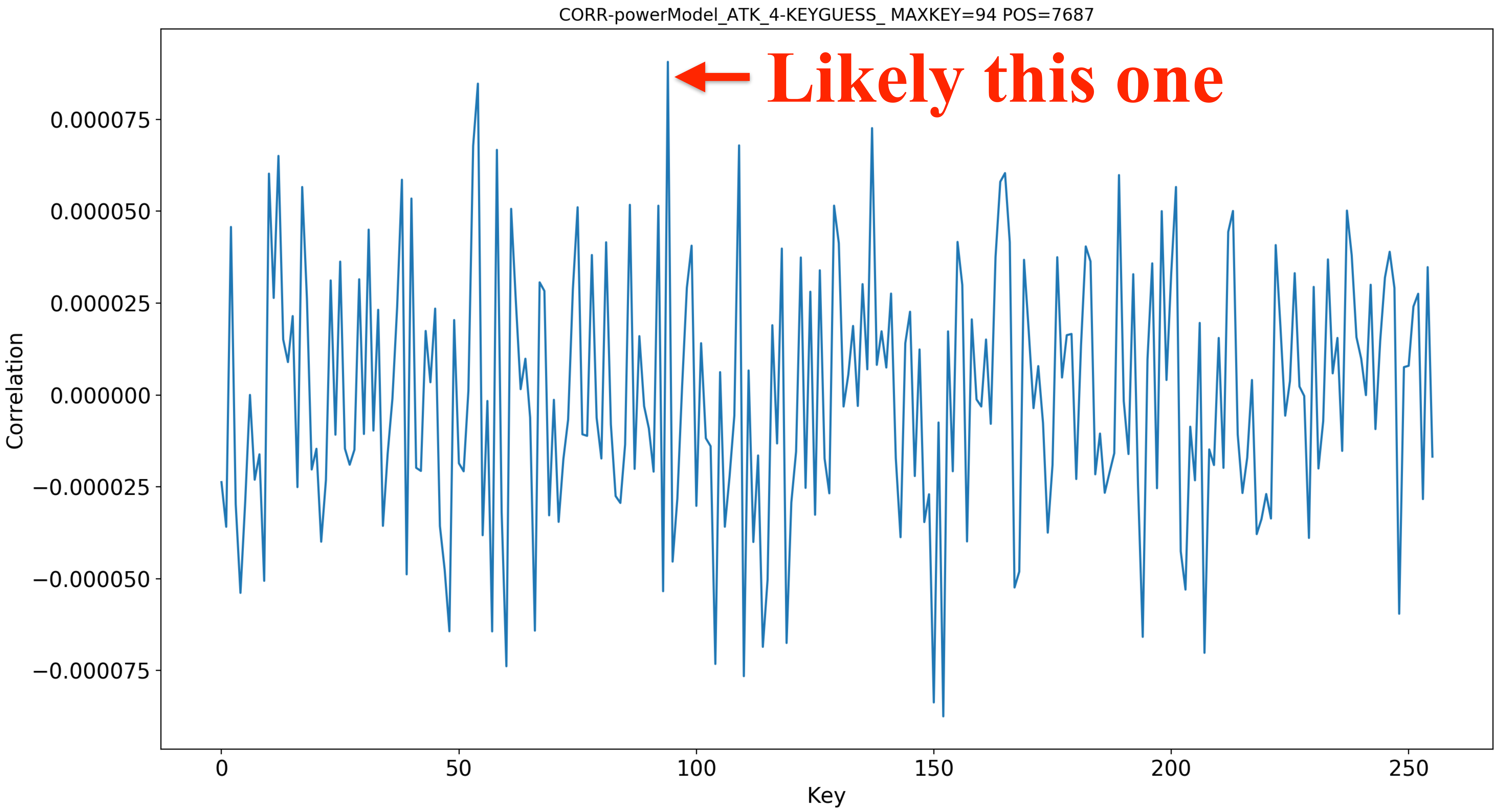

CORR-powerModel_ATK_4-KEYGUESS_ MAXKEY=94 POS=7686
Correlation
0.000075
0.000050
0.000025
0.000000
−0.000025
−0.000050
−0.000075
−0.000100
0
50
100
150
200
250
Key


CORR-powerModel_ATK_4-KEYGUESS_ MAXKEY=94 POS=7687
Correlation
Key
0.000075
0.000050
0.000025
0.000000
−0.000025
−0.000050
−0.000075
0
50
100
150
200
250


CORR-powerModel_ATK_4-KEYGUESS_ MAXKEY=94 POS=7688
Correlation
Key
0.000075
0.000050
0.000025
0.000000
−0.000025
−0.000050
−0.000075
0
50
100
150
200
250

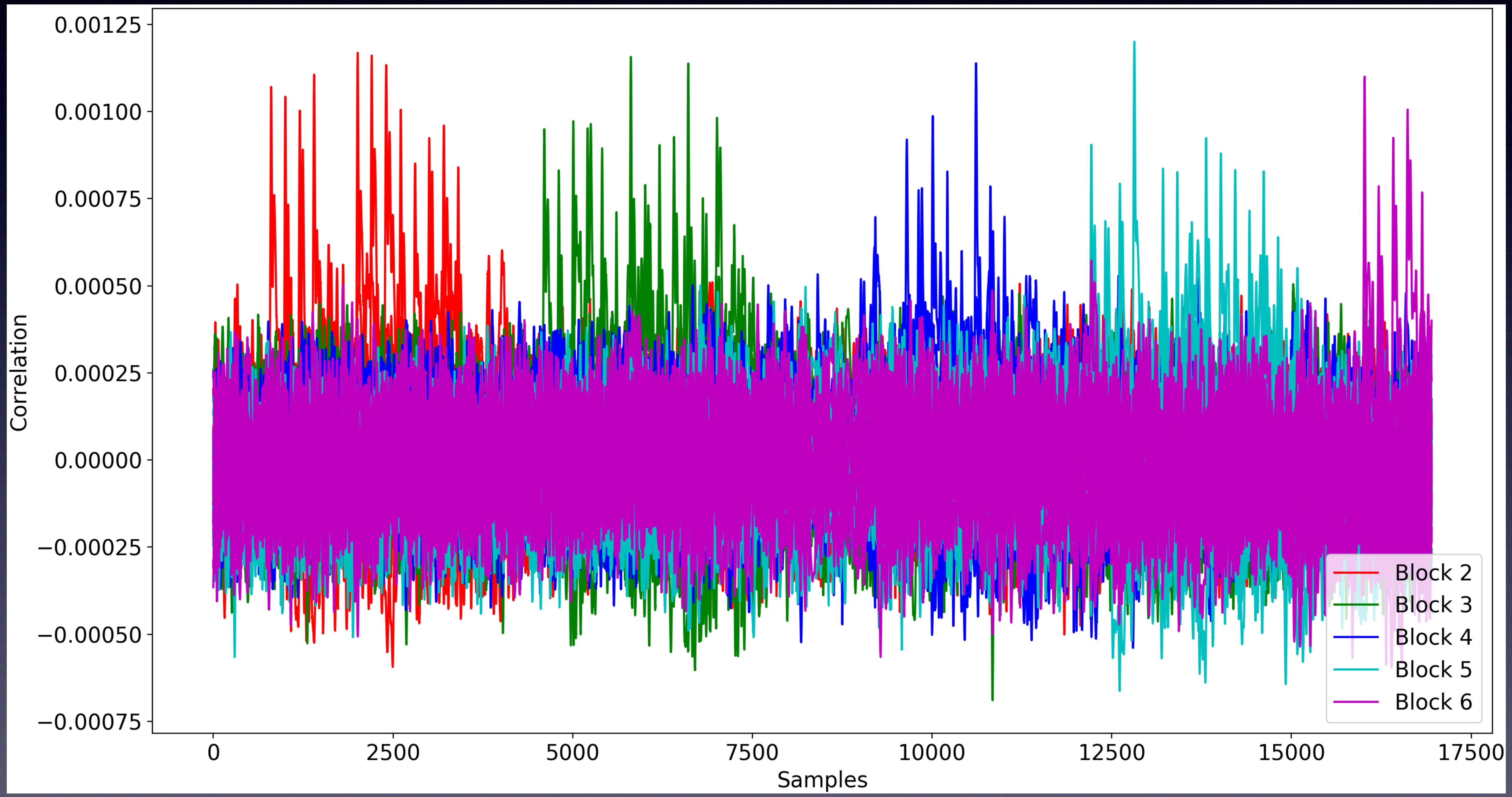

CORR-powerModel_ATK_4-KEYGUESS_ MAXKEY=94 POS=7689
Correlation
0.000075
0.000050
0.000025
0.000000
−0.000025
−0.000050
−0.000075
0
50
100
150
200
250
Key

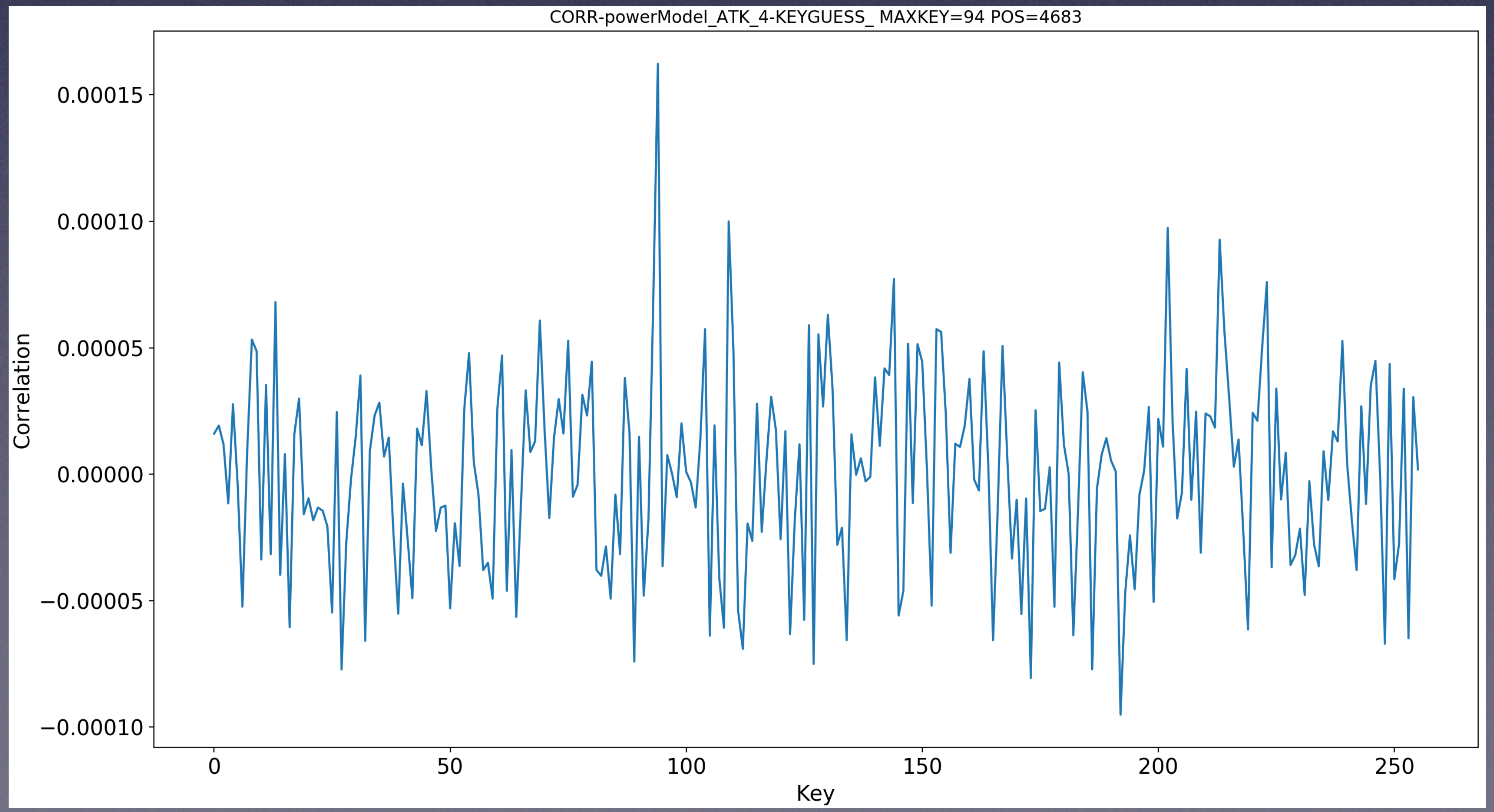

CORR-powerModel_ATK_4-KEYGUESS_ MAXKEY=58 POS=7690
Correlation
0.000075
0.000050
0.000025
0.000000
−0.000025
−0.000050
−0.000075
0
50
100
150
200
250
Key


CORR-powerModel_ATK_4-KEYGUESS_ MAXKEY=94 POS=7687
Likely this one
Correlation
0.000075
0.000050
0.000025
0.000000
−0.000025
−0.000050
−0.000075
Key
0
50
100
150
200
250

(noisy) Key recovery strategies

- Check other blocks





Target key SHA1 (A4 GID):

846017165c9f50304fb465fee6978f22dc82da10

Firmware decryption

```
$ unzip iPhone3,2_7.1.2_11D257_Restore.ipsw kernelcache.release.n90b
Archive:  iPhone3,2_7.1.2_11D257_Restore.ipsw
  inflating: kernelcache.release.n90b
$ xxd -p kernelcache.release.n90b | tr -d '\n' | grep -o "4741424b.*" | xxd -r -p | dd bs=1 skip=0
     x14 count=48 2>/dev/null | openssl aes-256-cbc -iv 00000000000000000000000000000000 -d -nopad
      -K $(cat A4GIDKey.txt) | xxd -p | tr -d '\n' ;echo
054fa7c7537f0d7f5271349656d729e6f24fa28626283eb1e252fec878ab0716d0fd7b6e62cf114fcd1ce132ba96d633
$
```

Kernelcache

- **IV**: `054fa7c7537f0d7f5271349656d729e6`
- **Key**: `f24fa28626283eb1e252fec878ab0716d0fd7b6e62cf114fcd1ce132ba96d633`

Recovered UID key

- Crack passcode on GPU
- My implementation

digits	iPhone	RTX 2080 TI	8×RTX 2080 TI
4	13 minutes	2 seconds	< 1 second
6	22 hours	3 minutes	26 seconds
7	9 days	35 minutes	4 minutes
8	92 days	5 hours	43 minutes
9	925 days	58 hours	7 hours
10	25 years	24 days	3 days
11	253 years	243 days	30 days
12	2536 years	2439 days	304 days

Recovered UID key

- Crack passcode on GPU

Hashcat

- ~~My implementation~~

Table 1: Worst-case passcode search time

digits	iPhone	Vega64	8×Vega64
4	13 minutes	< 1 second	< 1 second
6	22 hours	16 seconds	2 seconds
7	9 days	2 minutes	20 seconds
8	92 days	27 minutes	3 minutes
9	925 days	5 hours	33 minutes
10	25 years	45 hours	6 hours
11	253 years	18 days	56 hours
12	2536 years	188 days	23 days

Questions?