SoC it to EM: electromagnetic side-channel attacks on a complex system-on-chip

Jake Longo[1] Elke De Mulder[2] Dan Page[1] Mike Tunstall[2]

[1]University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB. UK.

[2]Rambus Cryptography Research Division,
425 Market Street, 11th Floor,
San Francisco, CA 94105, United States.

16/09/15

Presentation Layout

- Motivation?
- Methodology outline and execution
- Summary of attack results
- Further comments
- Future work

Motivation?

Address some misconceptions of side-channel attacks on complex devices.

- High-clock rate targets → high sample rate equipment.
- Complex embedded systems → difficult DPA.
- High degree of parallelism → low SNR ~ intrinsic side-channel resistance.

Analysis Plan

- Target selection and identification
- Signal exploration
- Batch signal pre-processing
- Leakage detection
- Signal post-processing
- Textbook DPA

Target Platform

BeagleBone Black

Attack Environment

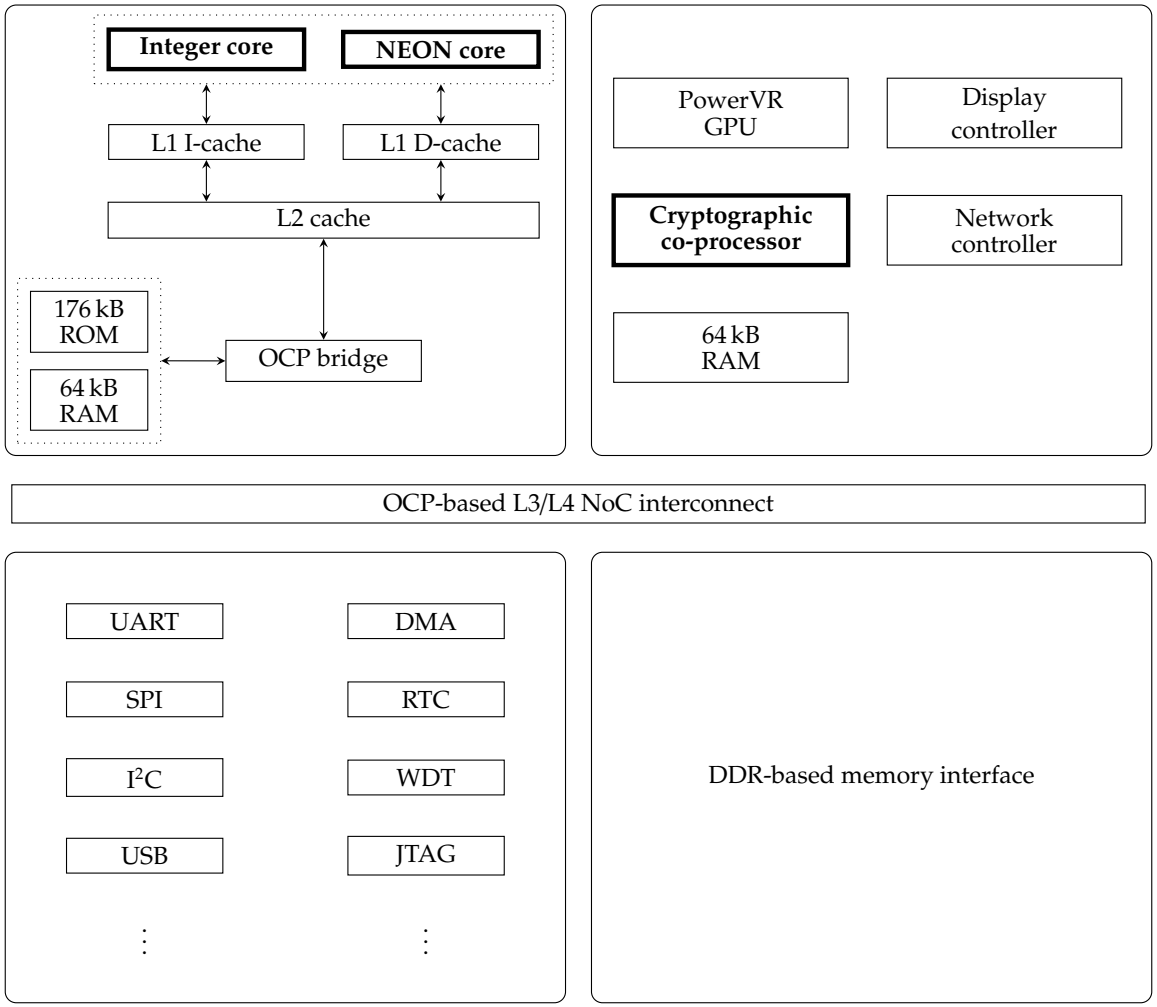
Hardware:

- ARM Cortex-A8 1 GHz CPU (High clock rate)
- ARM NEON SIMD (High degree of parallelism)
- TI proprietary cryptographic hardware (RNG, SHA-1, AES)

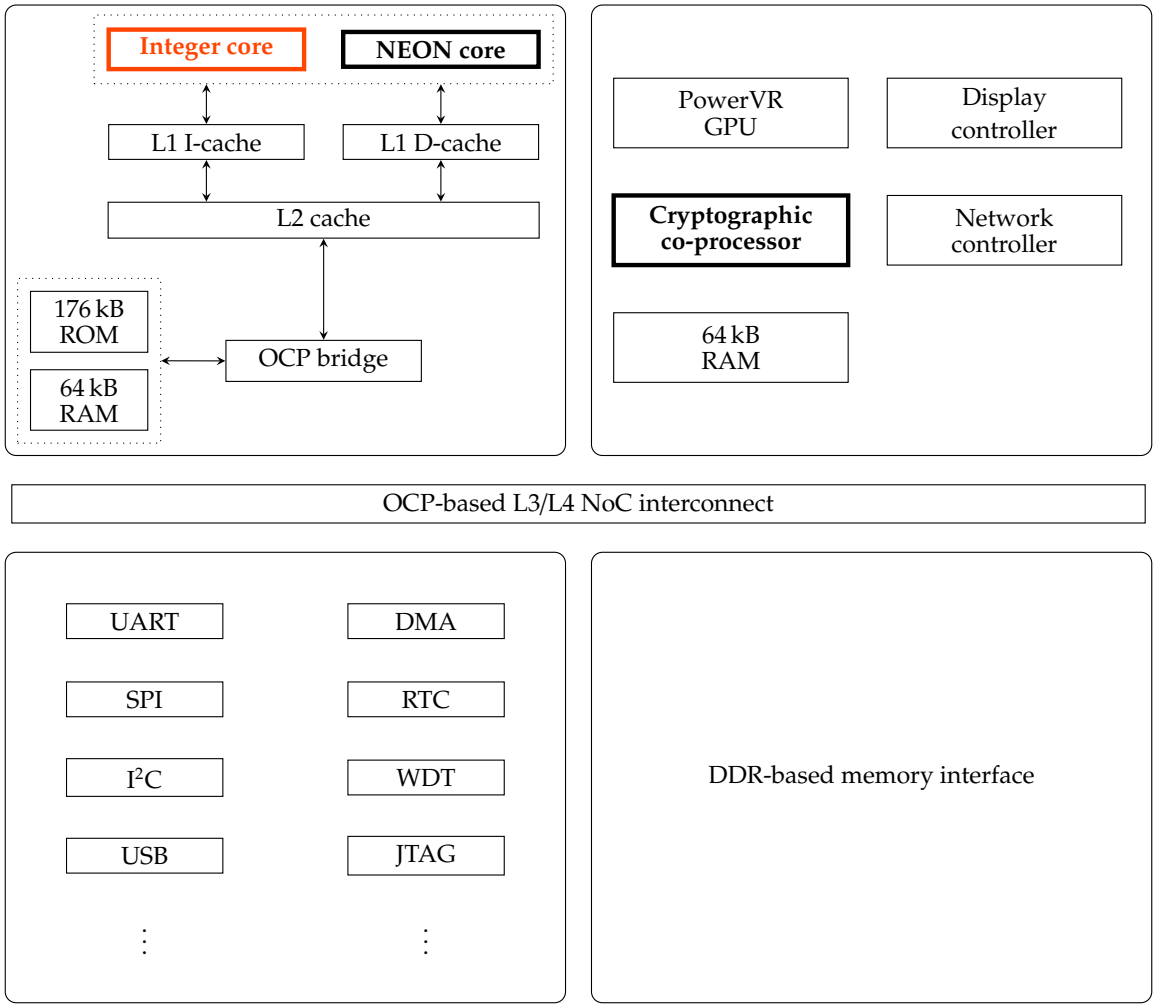
Software:

- Debian Wheezy (3.15) (Full unmodified Linux distribution)
- OpenSSL 1.0.1*j* (Bulk encryption)

Target Selection and Identification

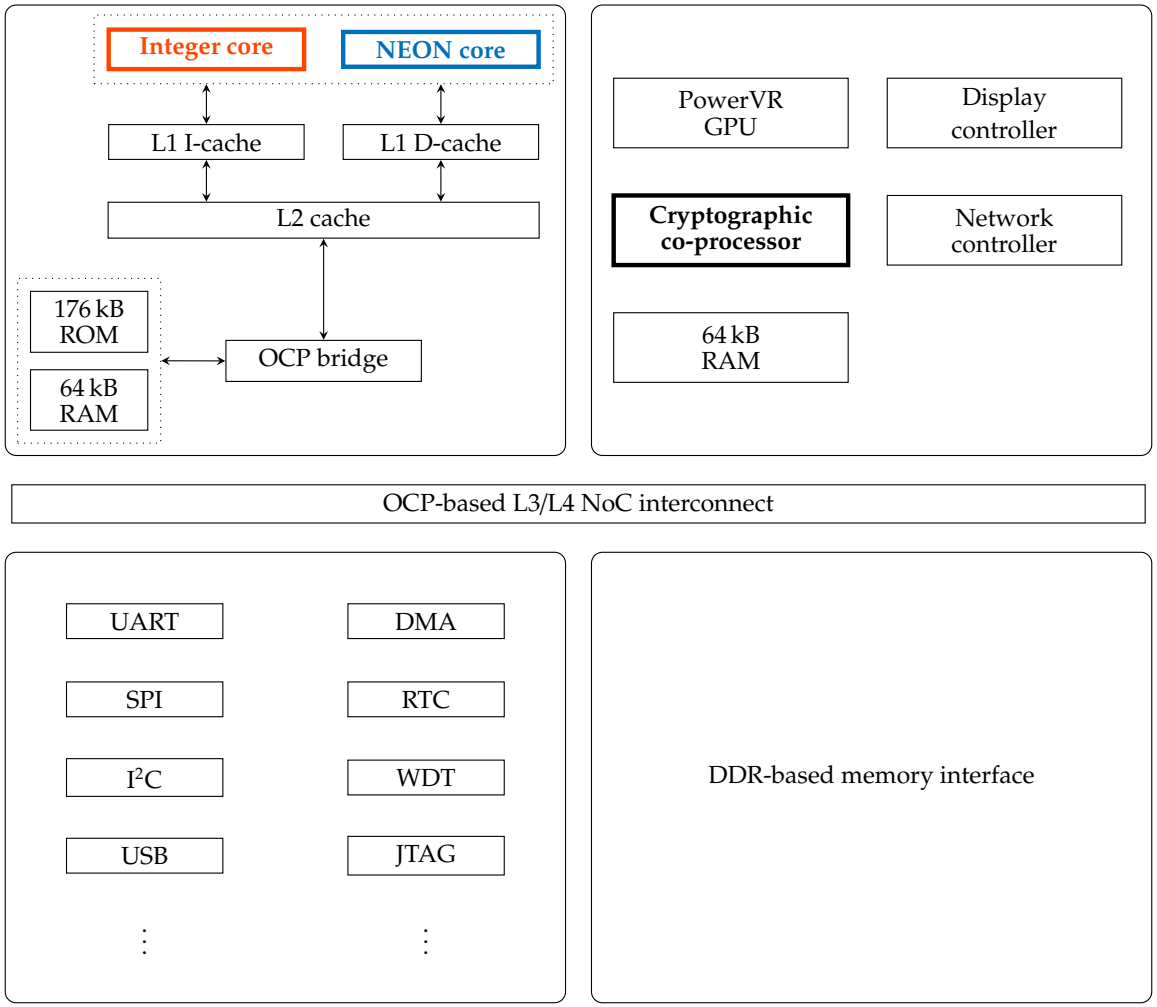


Target Selection and Identification



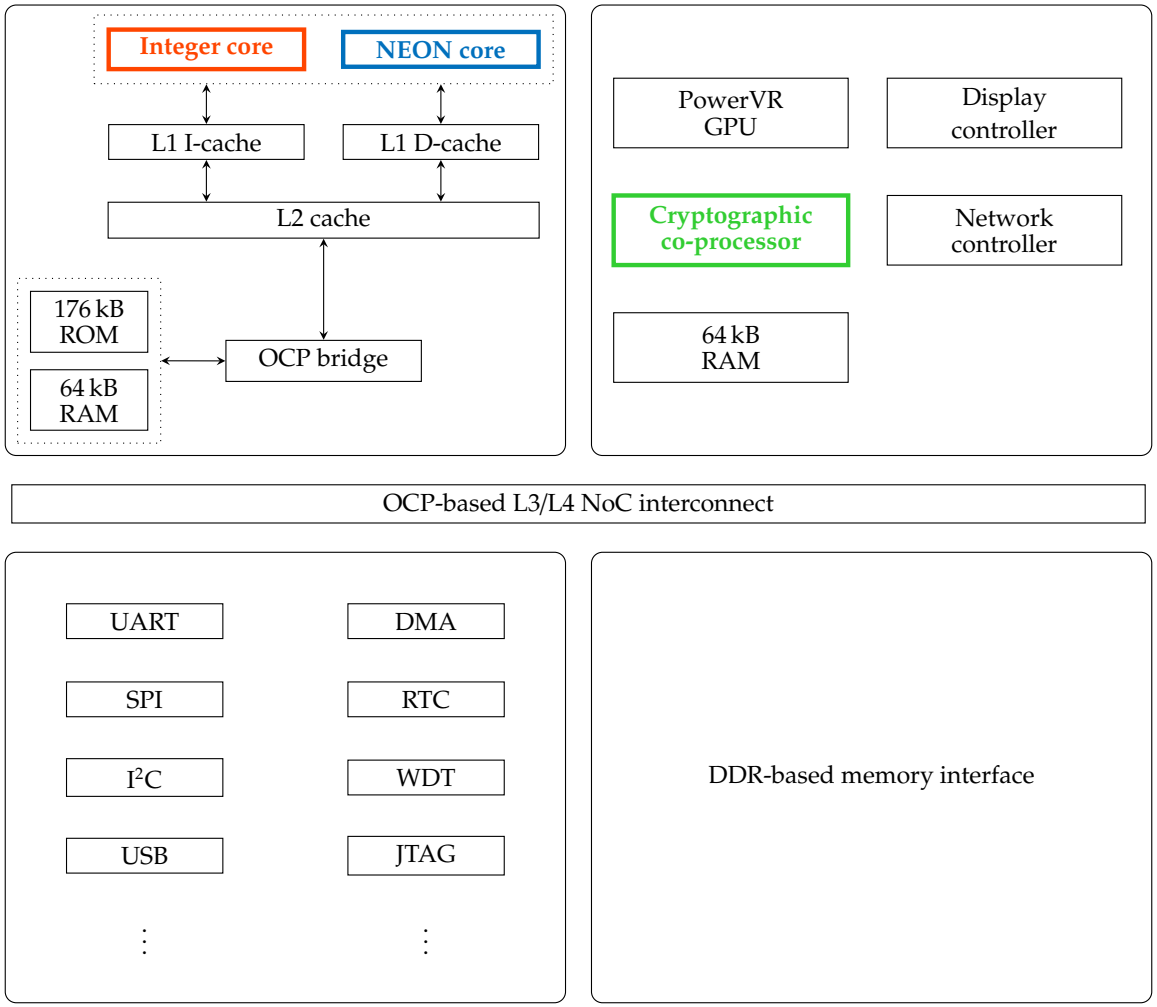
- OpenSSL software AES-128-CBC

Target Selection and Identification



- OpenSSL software AES-128-CBC
- OpenSSL NEON Bitsliced AES-128-CBC

Target Selection and Identification



- OpenSSL software AES-128-CBC
- OpenSSL NEON Bitsliced AES-128-CBC
- OpenSSL hardware accelerated AES-128-CBC

NEON?

"NEON technology is a 128-bit SIMD (Single Instruction, Multiple Data) architecture extension for the ARM Cortex™-A series processors."

- Clear use-cases for wide datapath bit-slicing.
- Gradually being adopted to accelerate crypto imlementations.

[BS12] D.J. Bernstein and P. Schwabe. "NEON Crypto". In: *CHES*. LNCS 7428, 2012, pp. 320–339.

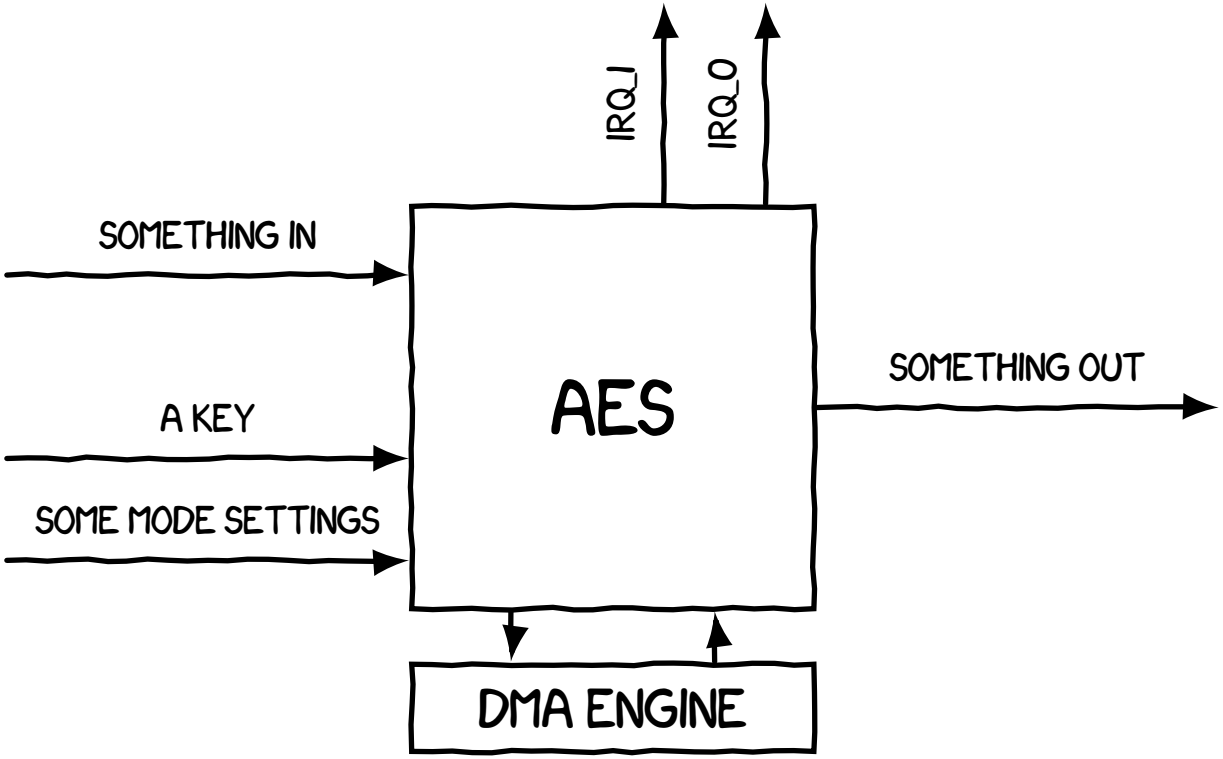
[Câm+13] D.F. Câmara et al. "Fast Software Polynomial Multiplication on ARM Processors Using the NEON Engine". In: *CD-ARES*. 2013, pp. 137–154.

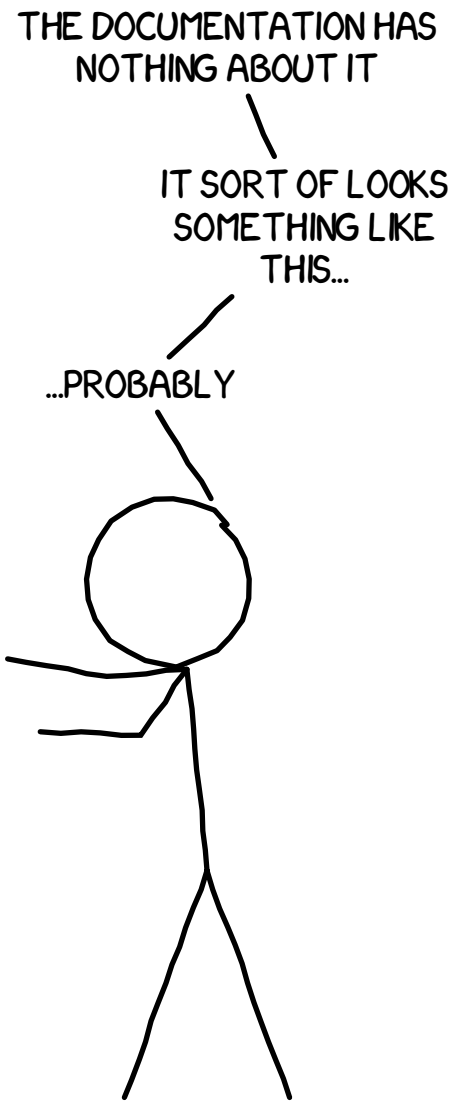
[Hol+13] S. Holzer-Graf et al. "Efficient Vector Implementations of AES-Based Designs: A Case Study and New Implemenations for Grøstl". In: *CT-RSA*. 2013, pp. 145–161.

[Seo+14] H. Seo et al. "Montgomery Modular Multiplication on ARM-NEON Revisited". In: *ICISC*. 2014, pp. 328–342.

[Wan+15] J. Wang et al. "Higher-Order Masking in Practice: A Vector Implementation of Masked AES for ARM NEON". In: *CT-RSA*. 2015, pp. 181–198.

Cryptographic Co-processor?



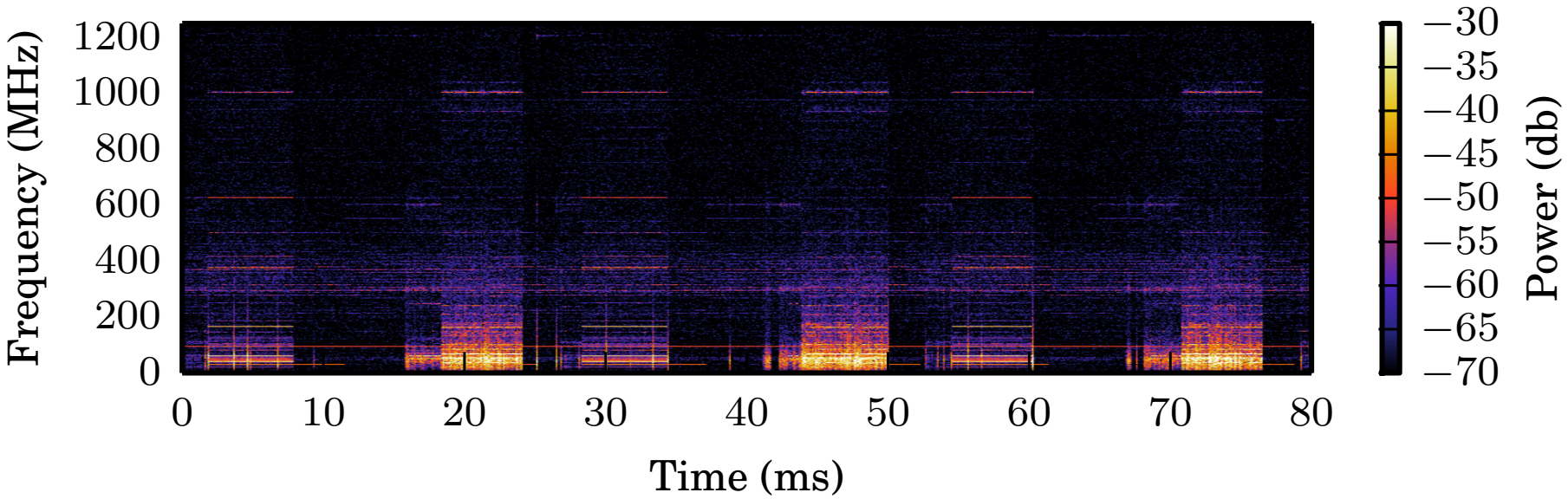


Signal Exploration (1)

Test loop

```
1 while true do
2   | sleep(0.08);
3   | openssl aes-128-cbc -in pt.bin -out ct.bin;
4   | sleep(0.025);
5   | matrixMultiply -in pt.bin;
6 end
```

Spectrogram

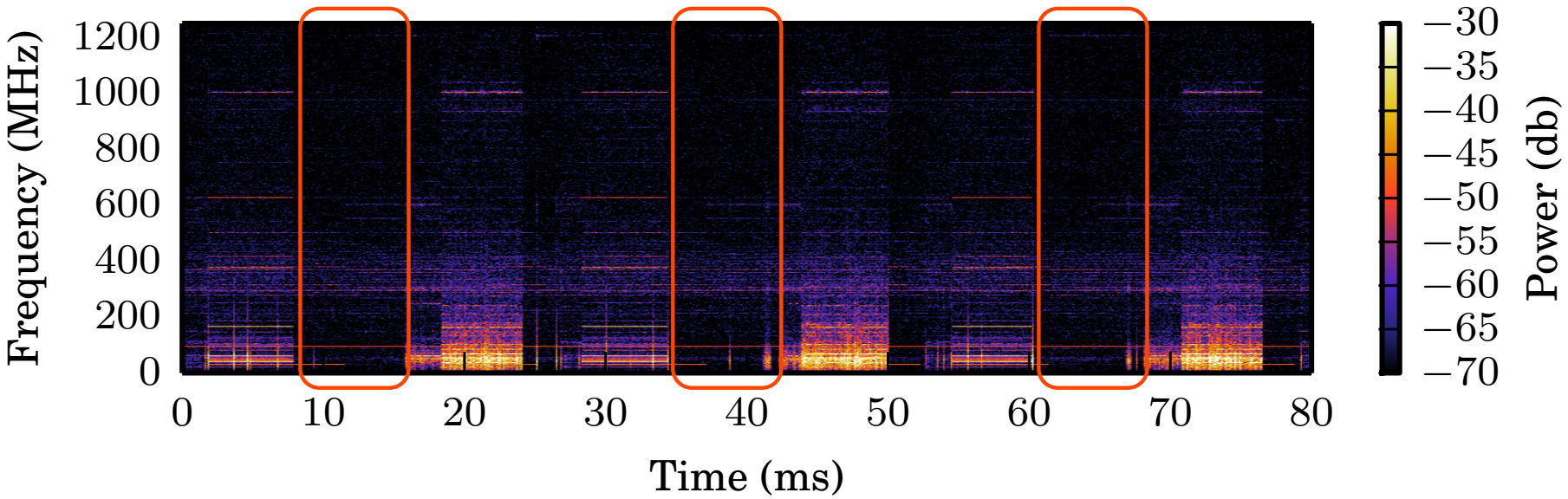


Signal Exploration (1)

Test loop

```
1 while true do
2     sleep(0.08);
3     openssl aes-128-cbc -in pt.bin -out ct.bin;
4     sleep(0.025);
5     matrixMultiply -in pt.bin;
6 end
```

Spectrogram

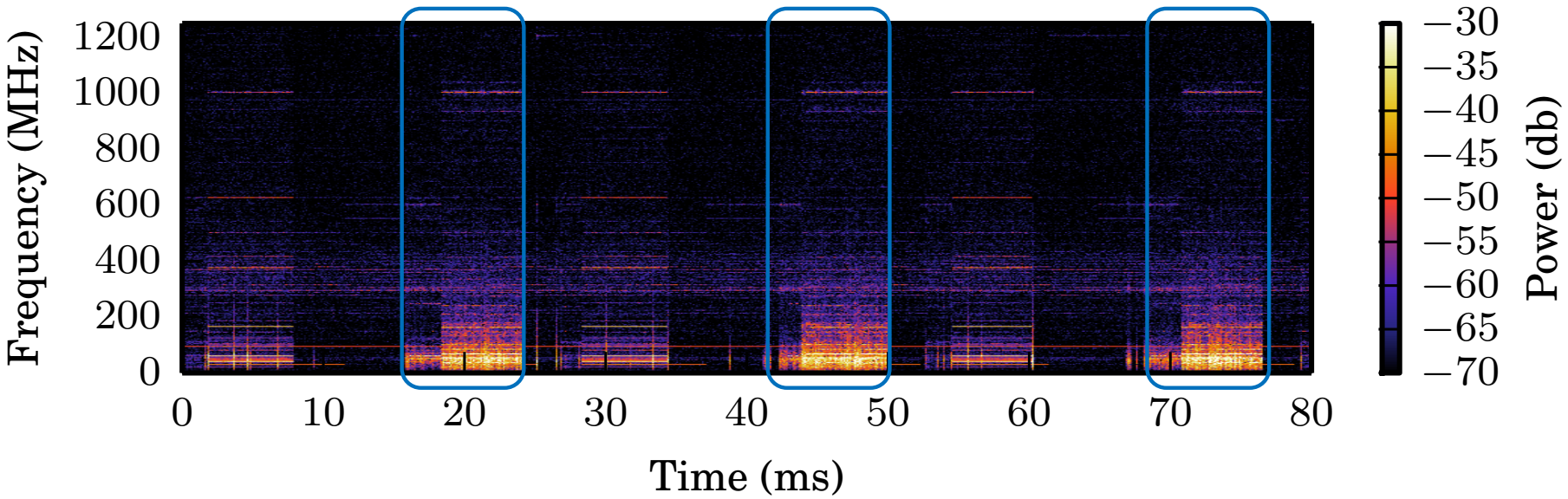


Signal Exploration (1)

Test loop

```
1 while true do
2 |  sleep(0.08);
3 |  openssl aes-128-cbc -in pt.bin -out ct.bin;
4 |  sleep(0.025);
5 |  matrixMultiply -in pt.bin;
6 end
```

Spectrogram

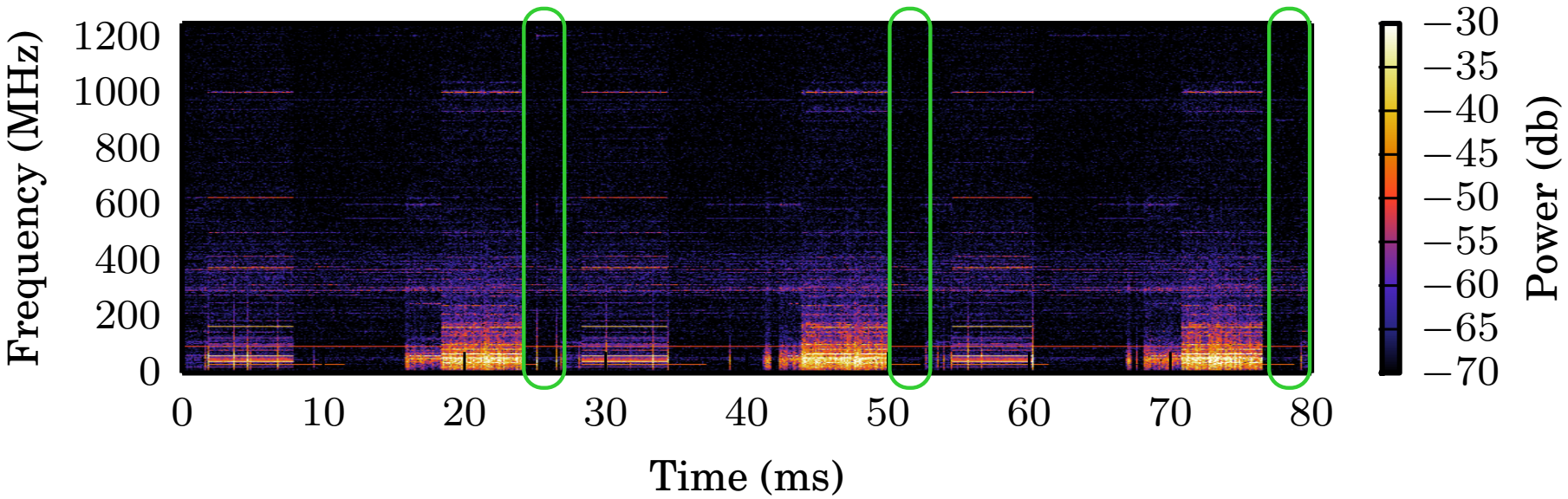


Signal Exploration (1)

Test loop

```
1 while true do
2 |  sleep(0.08);
3 |  openssl aes-128-cbc -in pt.bin -out ct.bin;
4 |  sleep(0.025);
5 |  matrixMultiply -in pt.bin;
6 end
```

Spectrogram

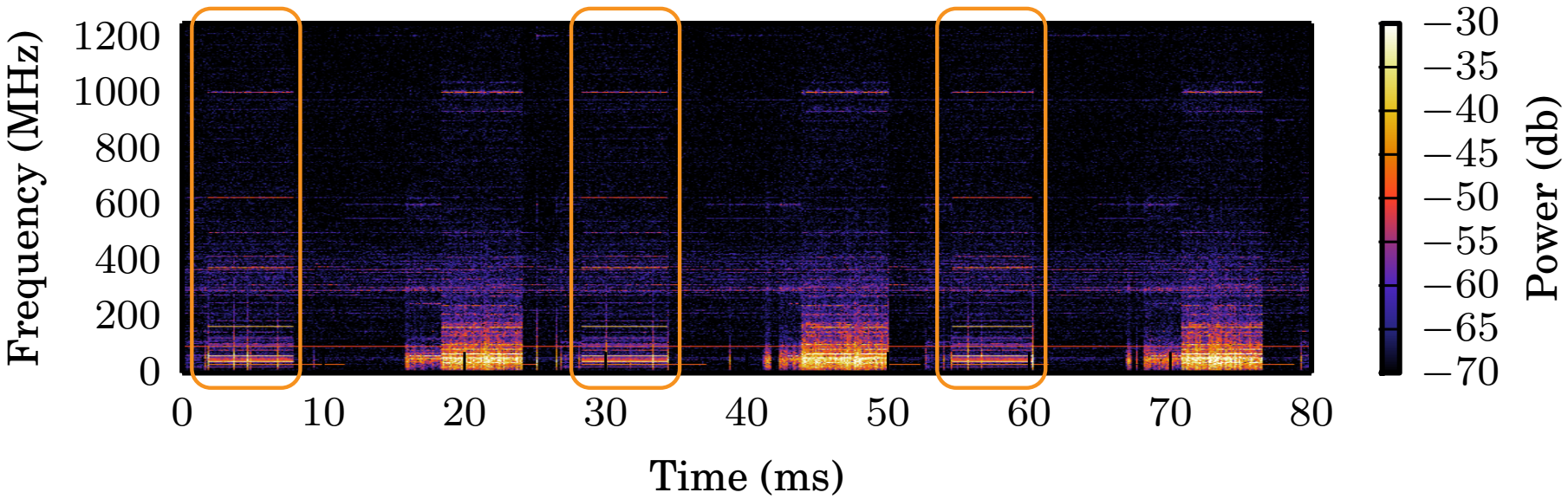


Signal Exploration (1)

Test loop

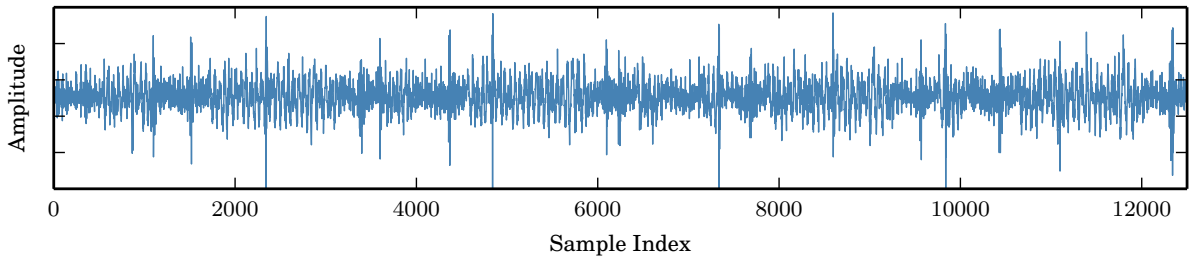
```
1 while true do
2   | sleep(0.08);
3   | openssl aes-128-cbc -in pt.bin -out ct.bin;
4   | sleep(0.025);
5   | matrixMultiply -in pt.bin;
6 end
```

Spectrogram

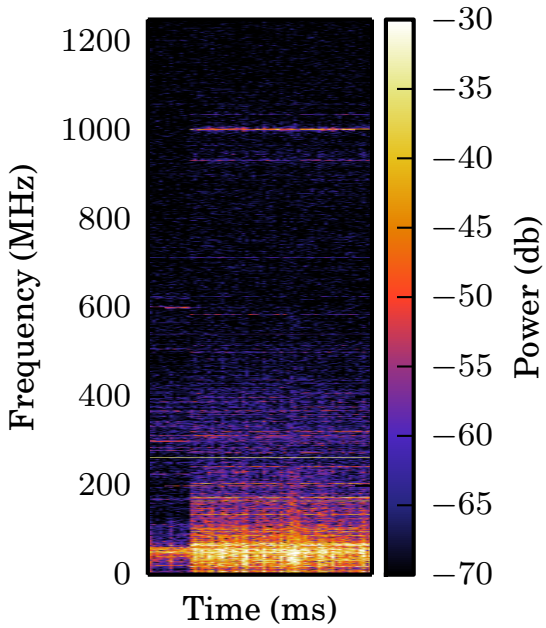


Signal Pre-processing (1)

OpenSSL S/W Trace

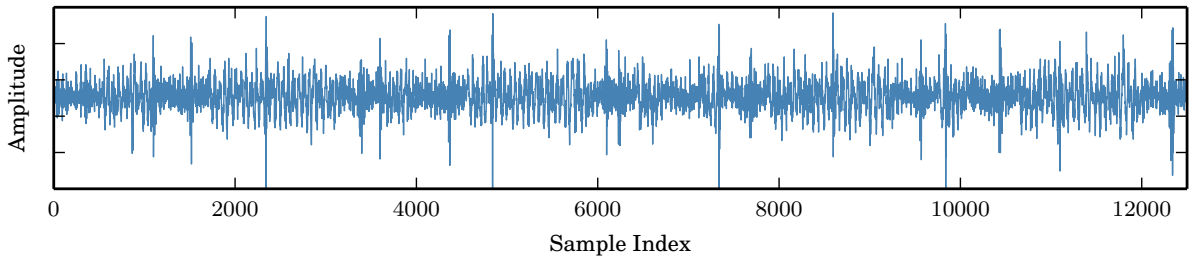


OpenSSL Frequency Response

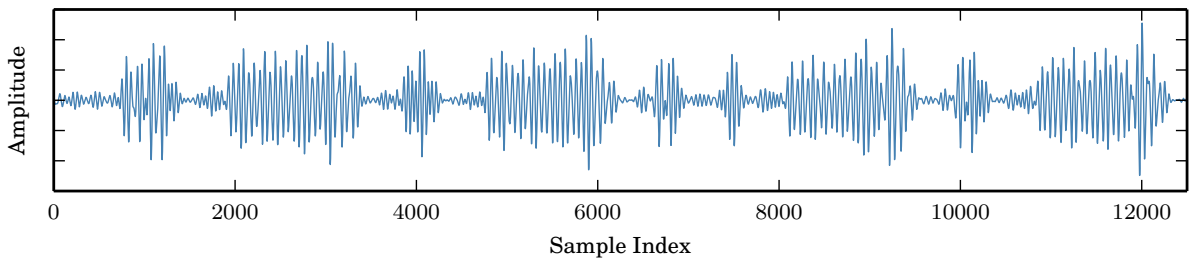


Signal Pre-processing (1)

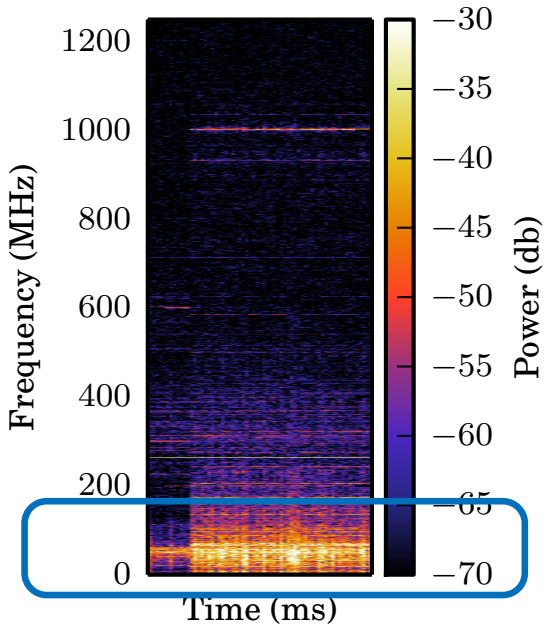
OpenSSL S/W Trace



OpenSSL S/W Trace – Filtered

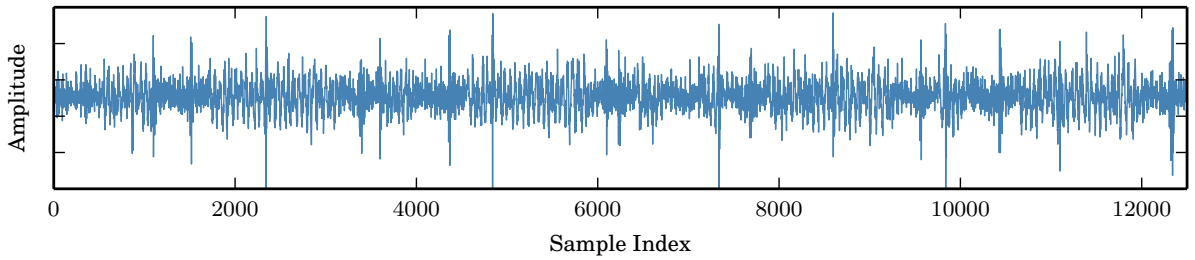


OpenSSL Frequency Response

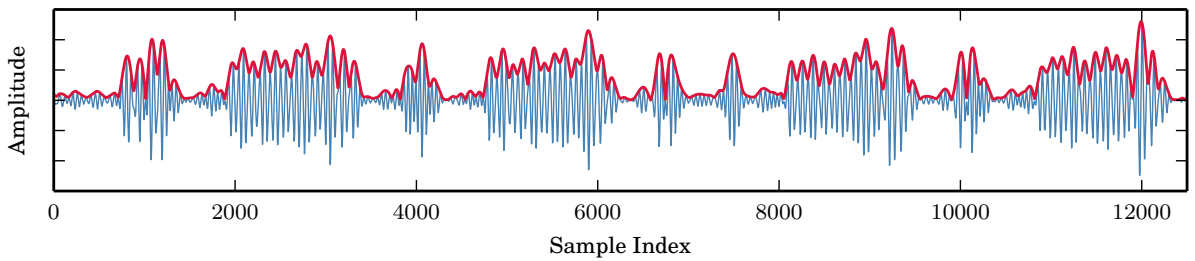


Signal Pre-processing (1)

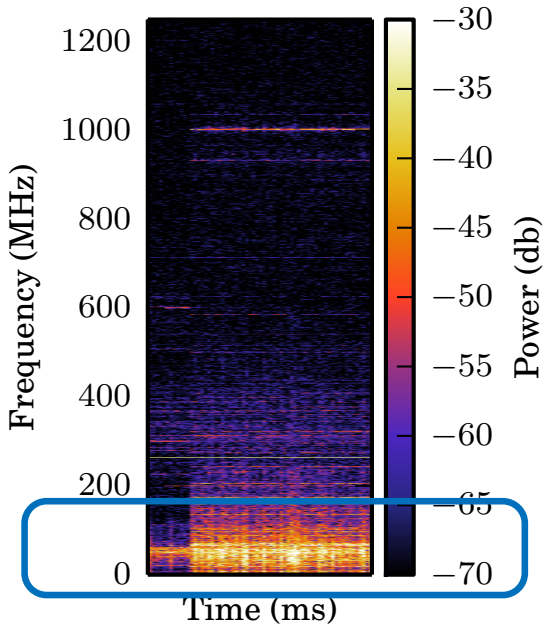
OpenSSL S/W Trace



OpenSSL S/W Trace – Filtered

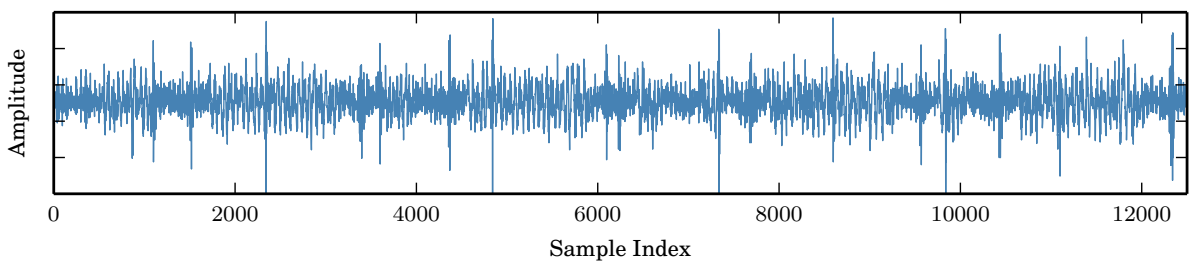


OpenSSL Frequency Response

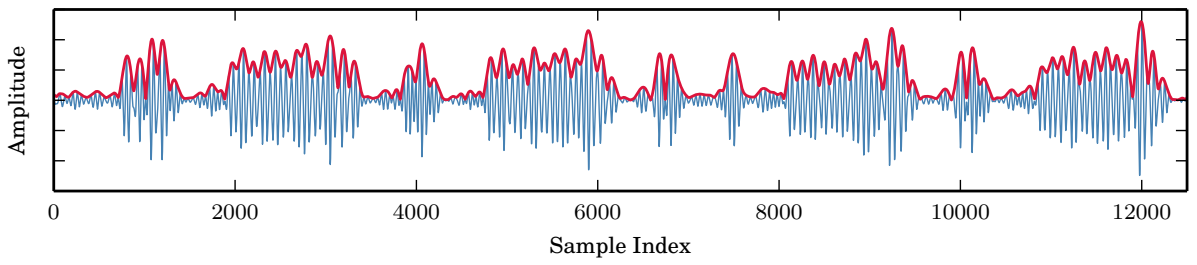


Signal Pre-processing (1)

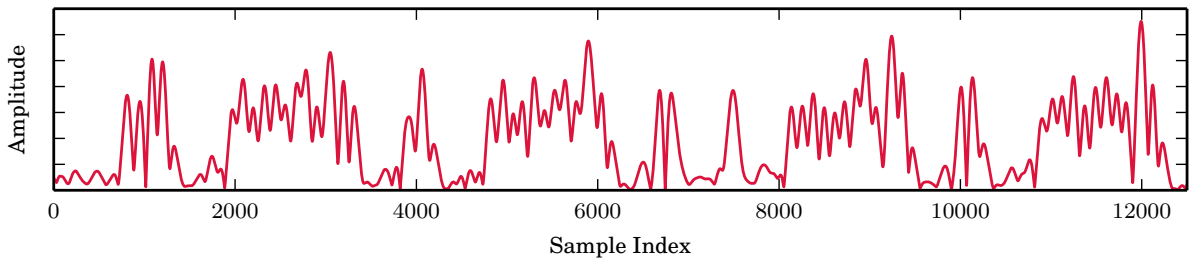
OpenSSL S/W Trace



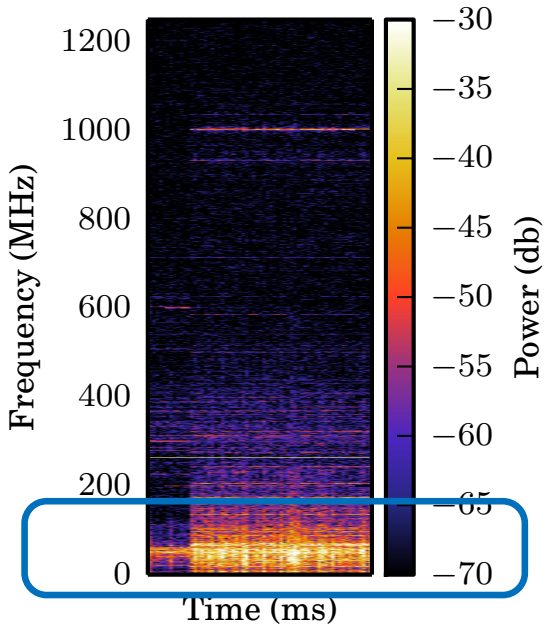
OpenSSL S/W Trace – Filtered



OpenSSL S/W Trace – Filtered & De-modulated

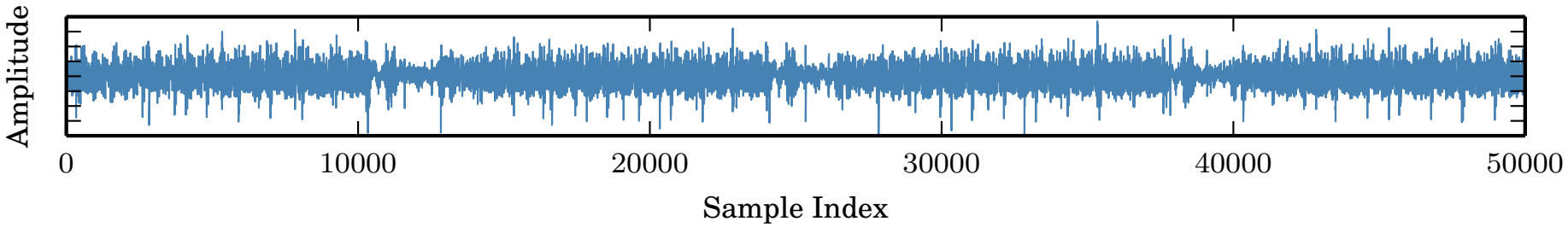


OpenSSL Frequency Response

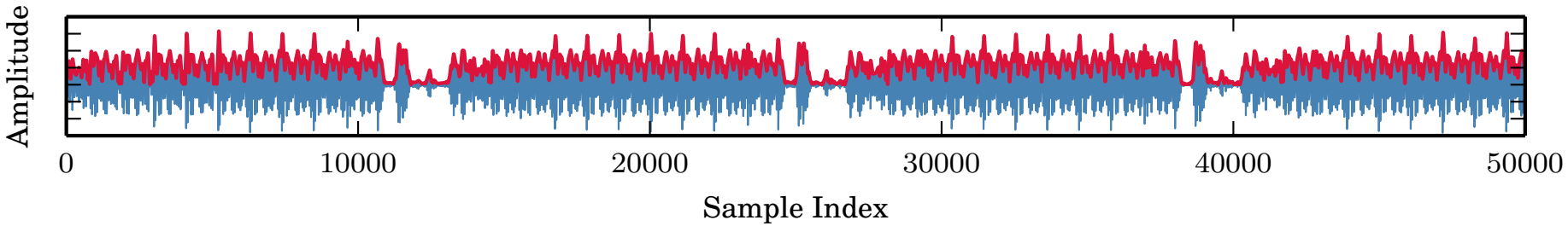


Signal Pre-processing (2)

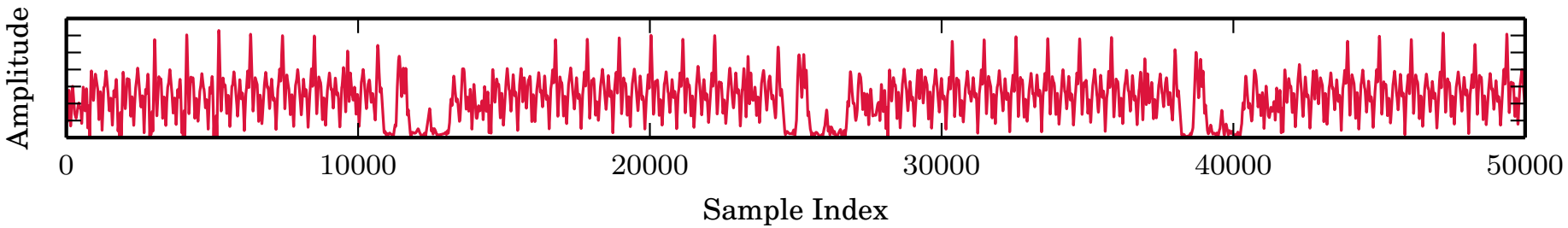
OpenSSL NEON Trace



OpenSSL NEON Trace – Filtered

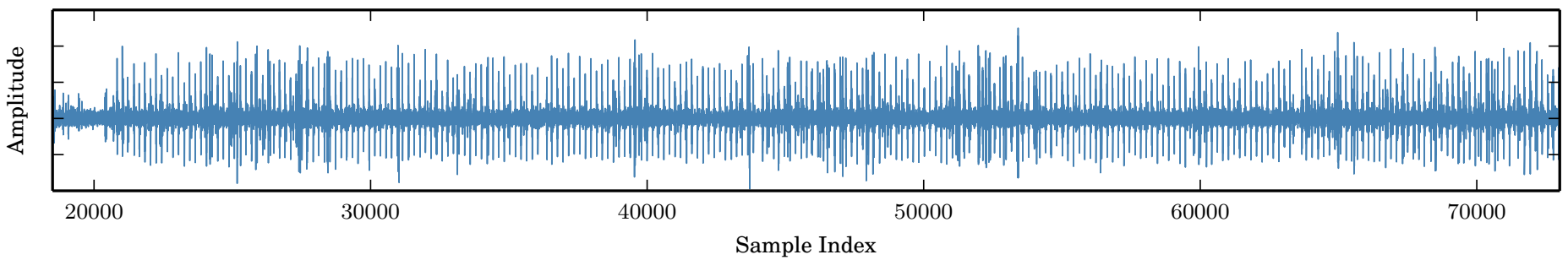


OpenSSL NEON Trace – Filtered & De-modulated

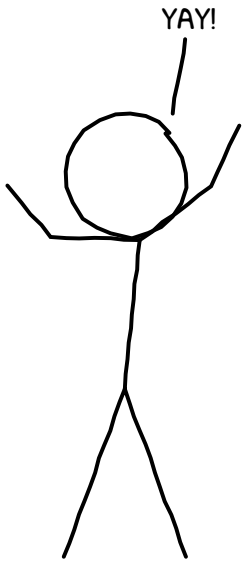


Signal Pre-processing (3)

OpenSSL H/W Trace

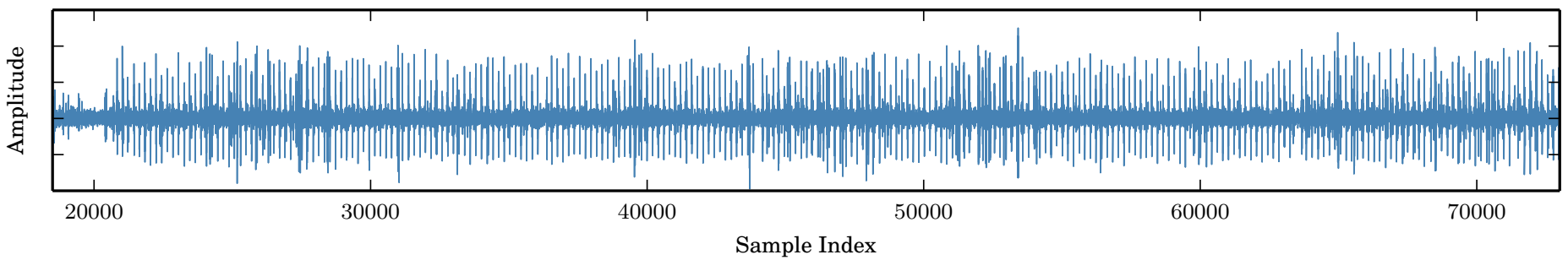


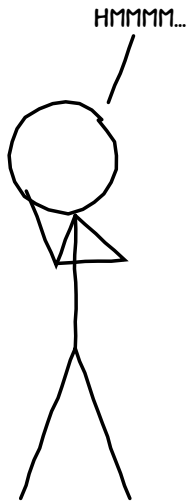
- Number of peaks match number of encryptions! ☺



Signal Pre-processing (3)

OpenSSL H/W Trace

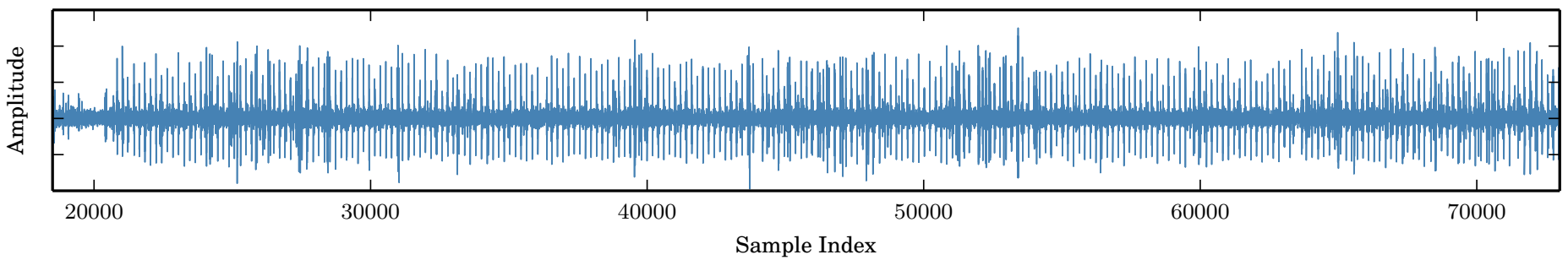


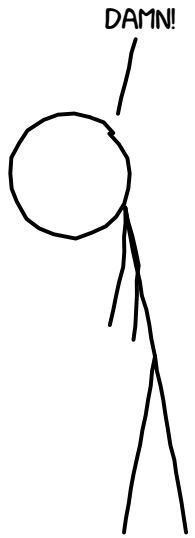


- Number of peaks match number of encryptions! ☺
- Peaks track by Hamming weight of plaintext...

Signal Pre-processing (3)

OpenSSL H/W Trace

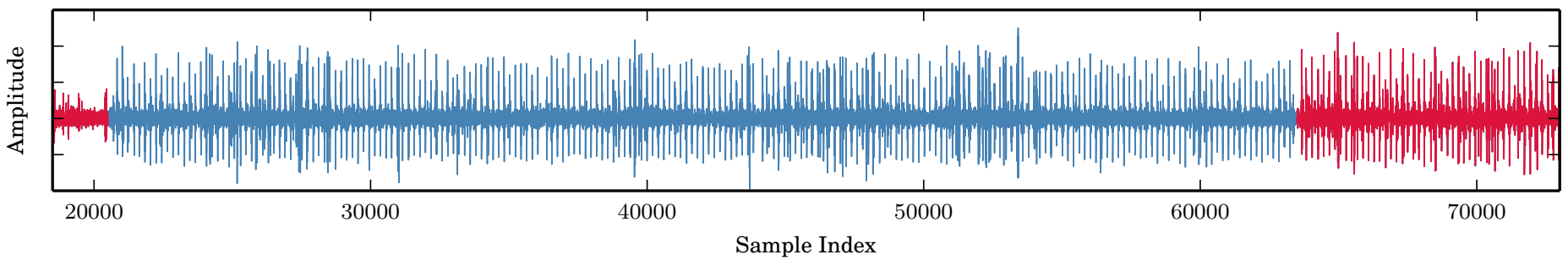


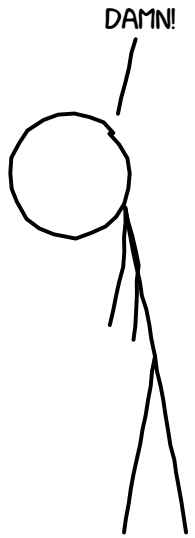


- Number of peaks match number of encryptions! ☺
- Peaks track by Hamming weight of plaintext...
- Peaks are DMA strobes ☹

Signal Pre-processing (3)

OpenSSL H/W Trace

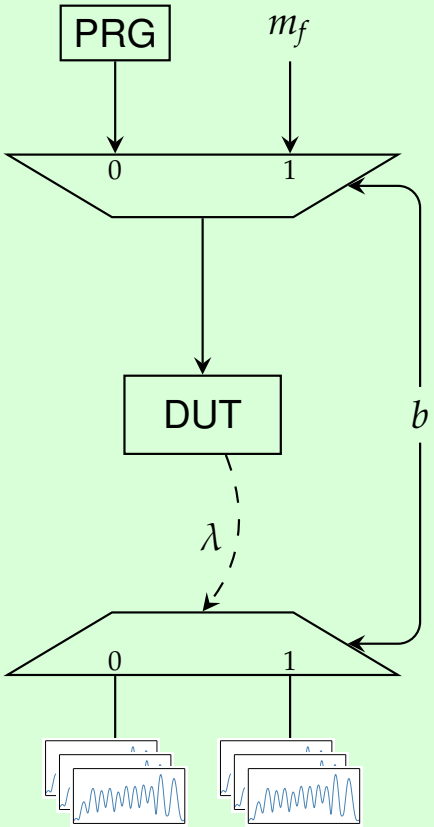




- Number of peaks match number of encryptions! ☺
- Peaks track by Hamming weight of plaintext...
- Peaks are DMA strobes ☹

Leakage detection (1)

Fixed-versus-Random Test



Fixed-versus-Random Test

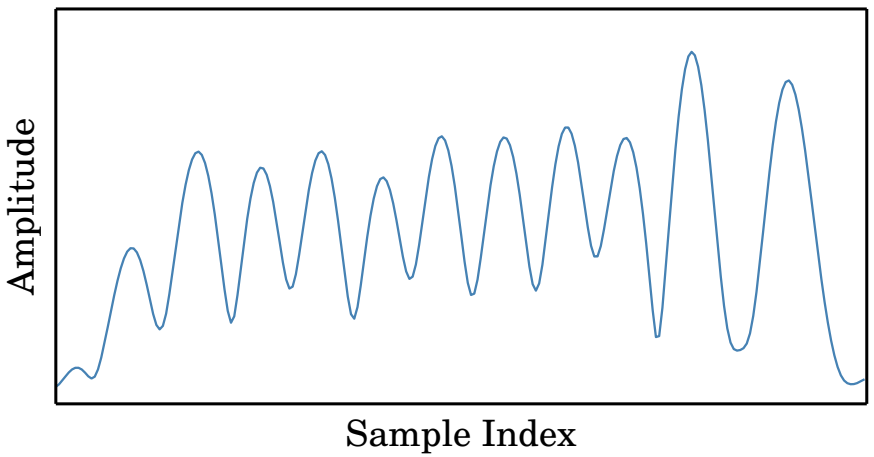
1 $m_f \xleftarrow{\$} \{0,1\}^{128}$
2 **for** $i \leftarrow 0$ **to** n **do**
3 $\quad b \xleftarrow{\$} \{0,1\}$
4 $\quad$ **if** $b = 0$ **then**
5 $\quad\quad m_r \xleftarrow{\$} \{0,1\}^{128}$
6 $\quad\quad \Lambda_0 \leftarrow \Lambda_0 \cup \{\lambda(\text{AES-128-CBC}_k(m_r))\}$
7 $\quad$ **else**
8 $\quad\quad \Lambda_1 \leftarrow \Lambda_1 \cup \{\lambda(\text{AES-128-CBC}_k(m_f))\}$
9 $\quad$ **end**
10 **end**

Welch's t-test

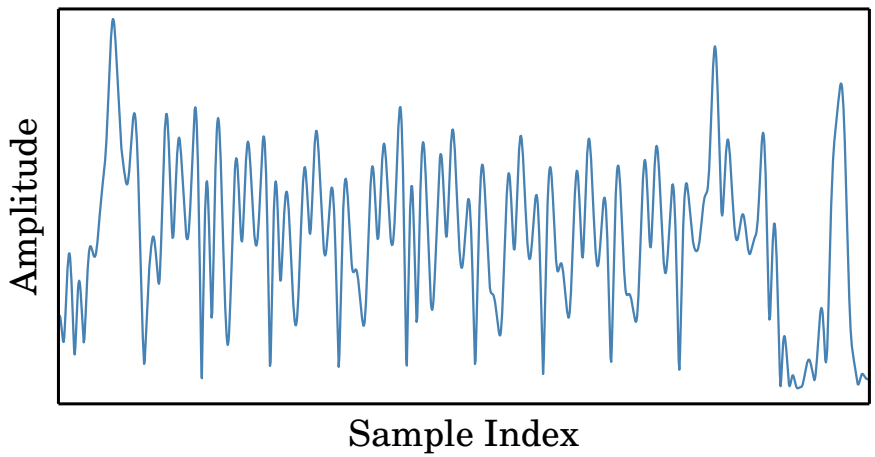
$$t = \frac{\bar{\Lambda}_0 - \bar{\Lambda}_1}{\sqrt{\frac{\sigma_0^2}{|\Lambda_0|} + \frac{\sigma_1^2}{|\Lambda_1|}}}$$

Leakage detection (2)

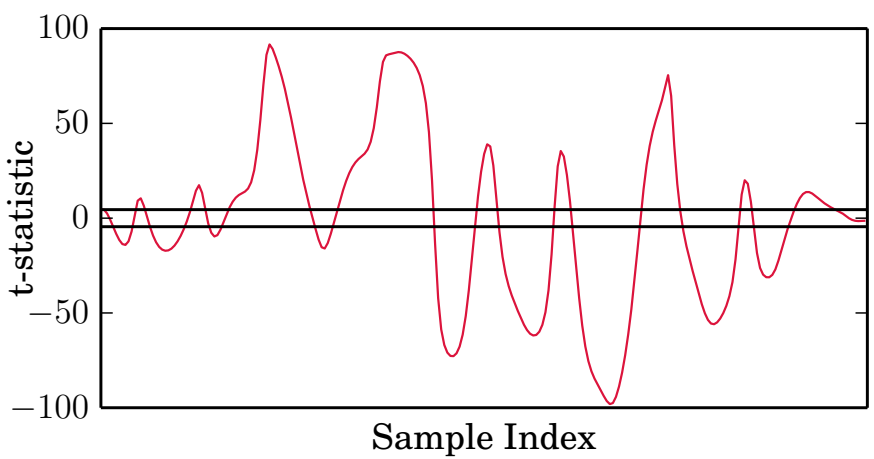
OpenSSL S/W Average



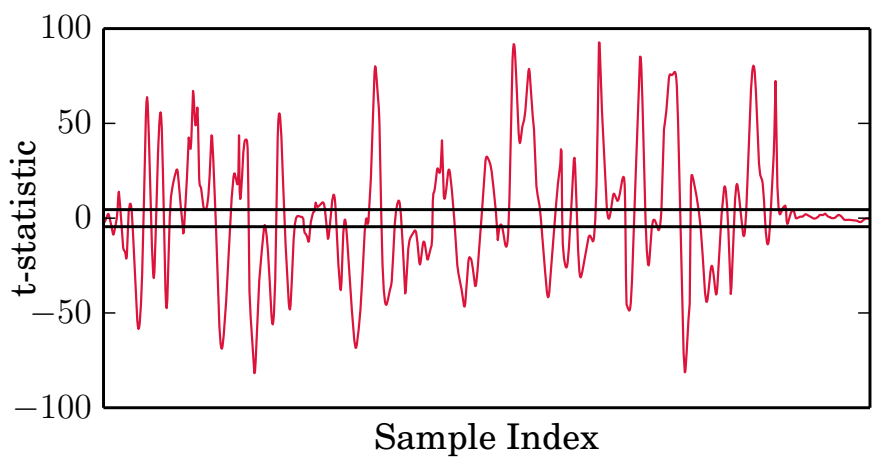
OpenSSL NEON Average



FvR OpenSSL S/W t-test

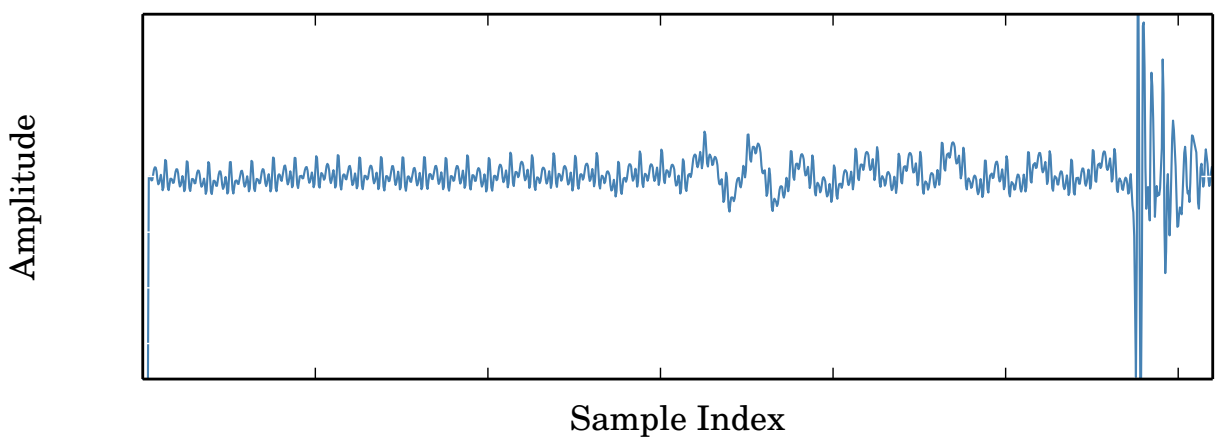


FvR OpenSSL NEON t-test

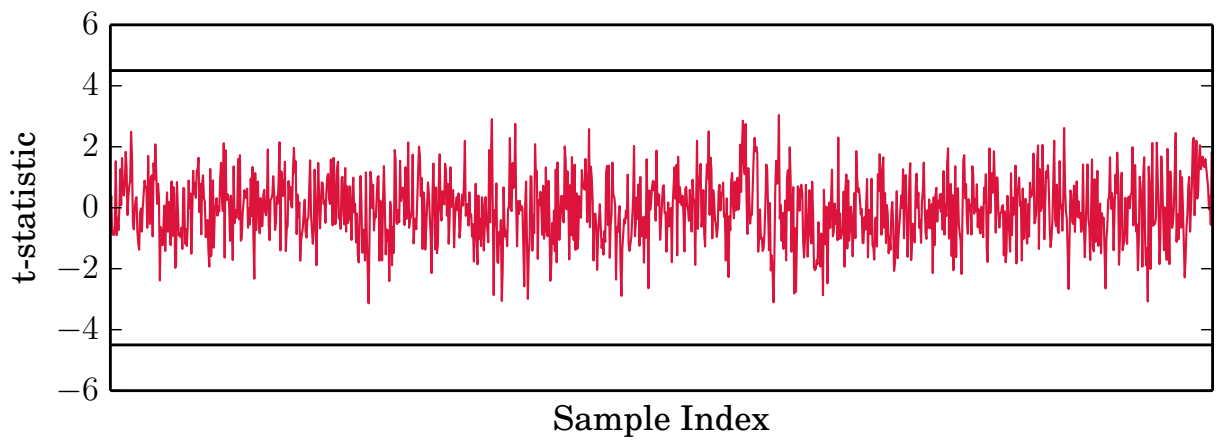


Leakage detection (3)

OpenSSL H/W Average

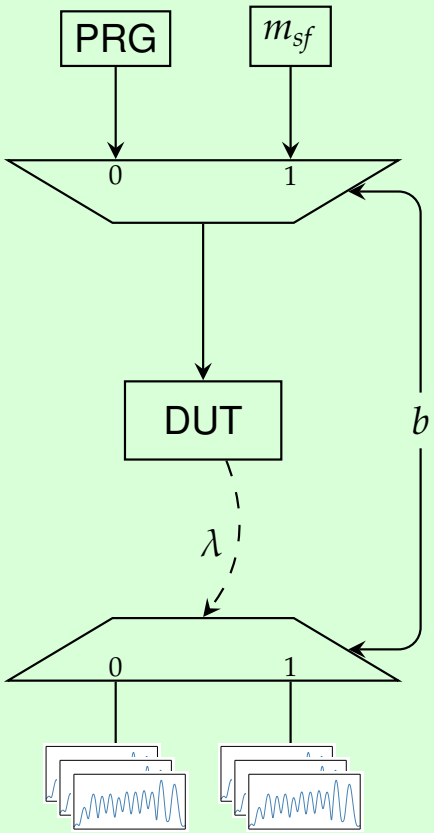


FvR OpenSSL H/W t-test



Leakage detection (4)

Semi Fixed-versus-Random

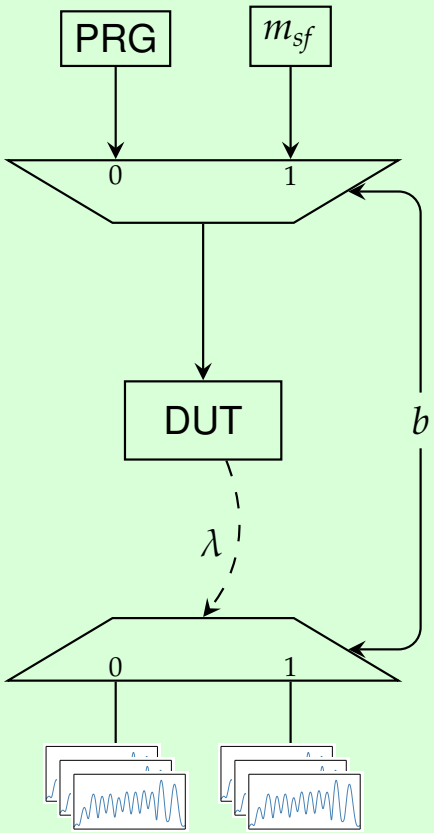


Semi Fixed-versus-Random Example Vectors

```
round[00].input:A6FE44D9FF7596F884BEDBAAFDBD3ABE
round[00].k_sch:382CDECF122333C71B124386107CE34F
round[00].start:9ED29A16ED56A53F9FAC982CEDC1D9F1
round[01].s_box:0BB5B84755B10675DB914671557835A1
round[01].s_row:0BB146A155913547DB78B87555B50671
round[01].m_col:3919CEB3707467D5E88D575C195F7FAE

                                    ⋮

round[09].k_sch:EDF9BA88533EBAB600001D3E003C1D00
round[10].start:0D848A8D000000000000000000000000
round[10].s_box:D75F7E5D6363636363636363636363
round[10].s_row:D76363636363635D63637E63635F6363
round[10].k_sch:305DD9EB6363635D63637E63635F6363
round[10].s_out:E73EBA880000000000000000000000
```

Leakage detection (4)

Semi Fixed-versus-Random

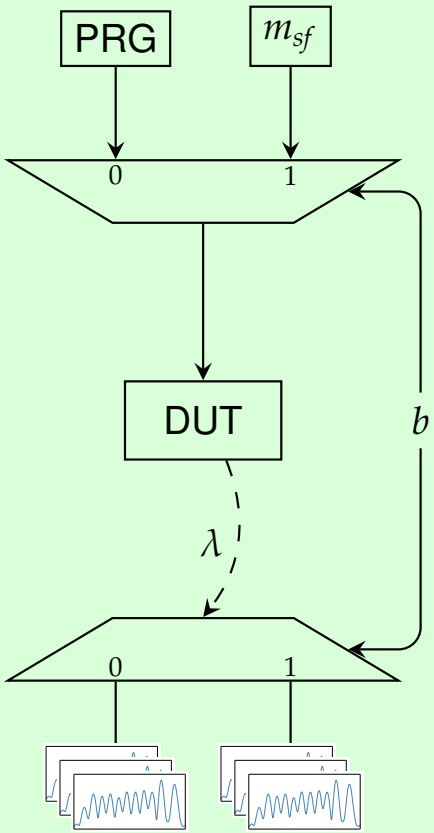


Semi Fixed-versus-Random Example Vectors

```
round[00].input:DBF0AE75B2BB6B56E89ECAC78188CF86
round[00].k_sch:C5D06552504A626F9F1B62E0B458A219
round[00].start:1E20CB27E2F109397785A82735D06D9F
round[01].s_box:72B71FCC98A10112F597C2CC96703CDB
round[01].s_row:72A1C2DB98973CCCF5701F1296B701CC
round[01].m_col:05AD3A587925389B6C268D4F382C6C94

                                    ⋮

round[09].k_sch:CD2CF959BC89F9320000C76B00A6C700
round[10].start:CD071DBF000000000000000000000000
round[10].s_box:BDC5A4086363636363636363636363
round[10].s_row:BD6363636363086363A46363C56363
round[10].k_sch:DFEA9A3A636363086363A46363C56363
round[10].s_out:6289F9590000000000000000000000
```

Leakage detection (4)

Semi Fixed-versus-Random

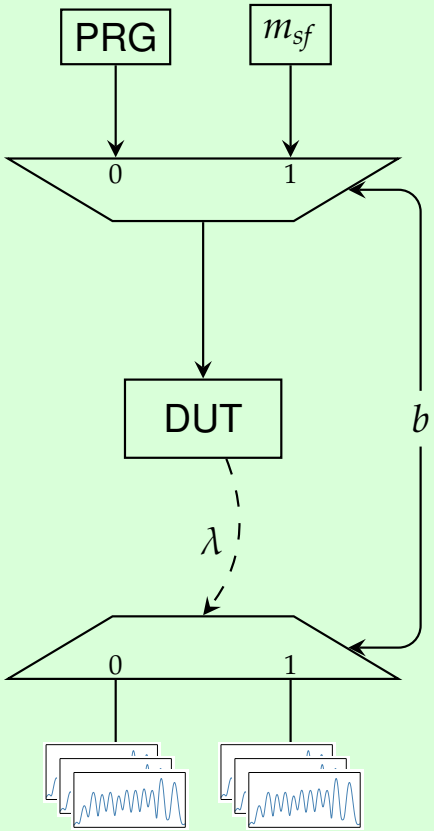


Semi Fixed-versus-Random Example Vectors

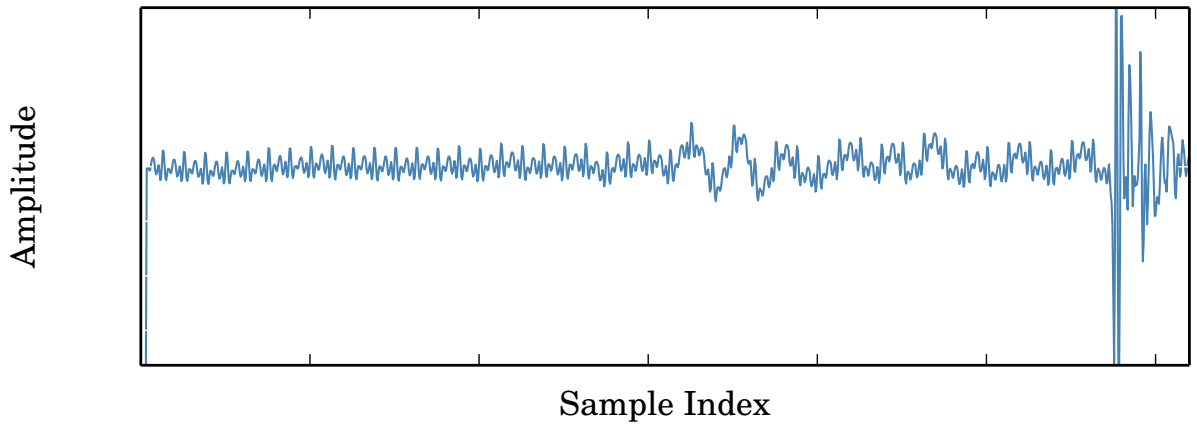
```
round[00].input:0B02F73CE36B4B962062B70737727265
round[00].k_sch:365FD0AE866D066A327CC70ED1BE4AD4
round[00].start:3D5D279265064DFC121E7009E6CC38B1
round[01].s_box:274CCC4F4D6FE3B0C97251018E4B07C8
round[01].s_row:276F51C84D72074FC94BCCB08E4CE301
round[01].m_col:66C2A9DC44EFE03C28A0CABC31291C24

                                    ⋮

round[09].k_sch:4B028D14D2898D0E0000C81A0027C800
round[10].start:74860EAF000000000000000000000000
round[10].s_box:9244AB796363636363636363636363
round[10].s_row:926363636363796363AB6363446363
round[10].k_sch:B1EAEE776363636796363AB6363446363
round[10].s_out:23898D140000000000000000000000
```

Leakage detection (4)

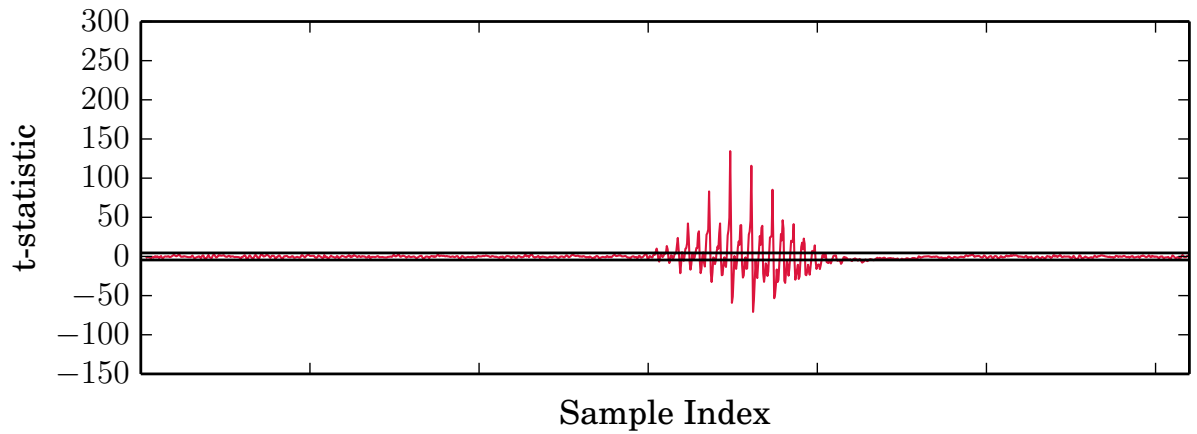
Semi Fixed-versus-Random



OpenSSL H/W Average

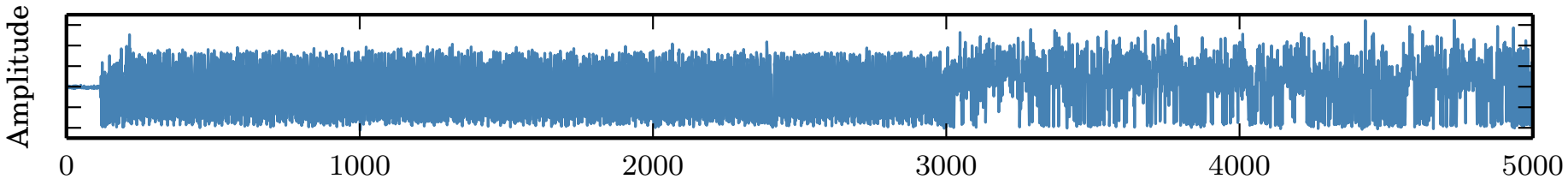


SFvR OpenSSL H/W t-test

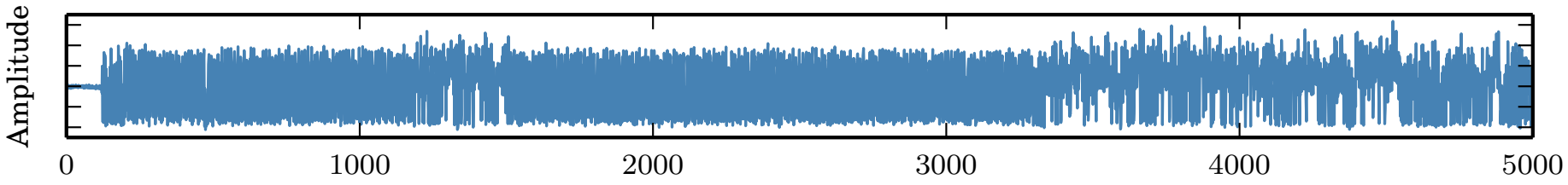


Signal Post-processing (1)

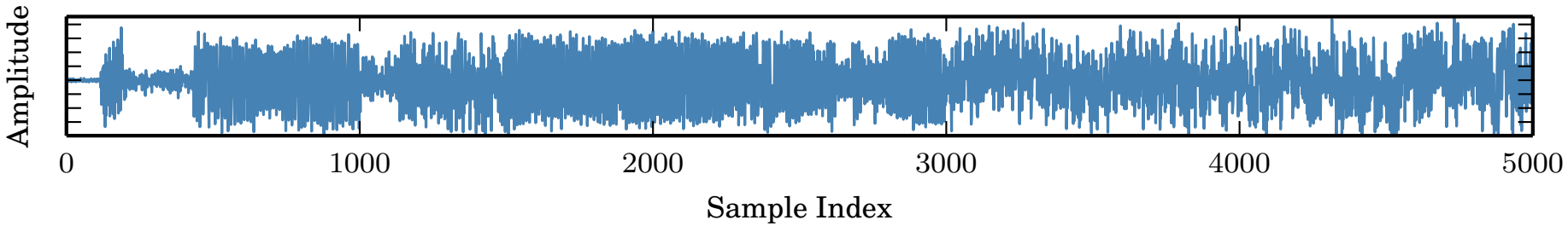
OpenSSL S/W Bulk Encryption



OpenSSL S/W Bulk Encryption Interrupted

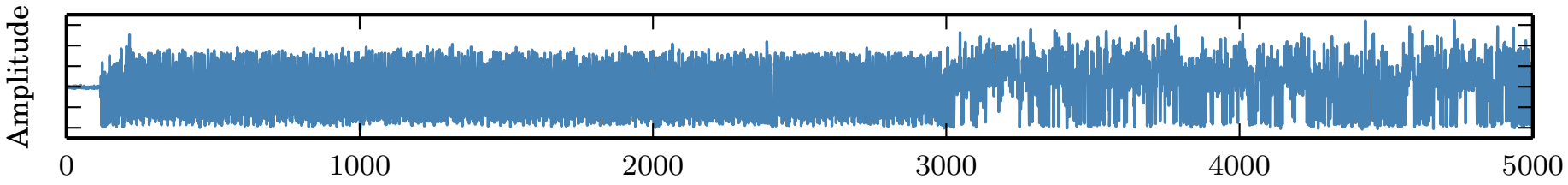


OpenSSL S/W Bulk Encryption Difference Trace

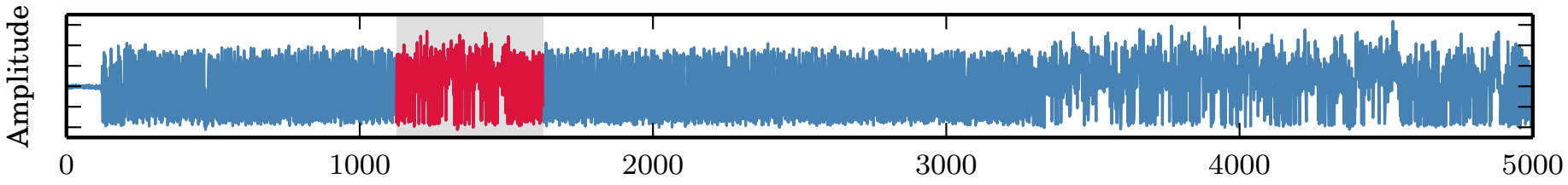


Signal Post-processing (1)

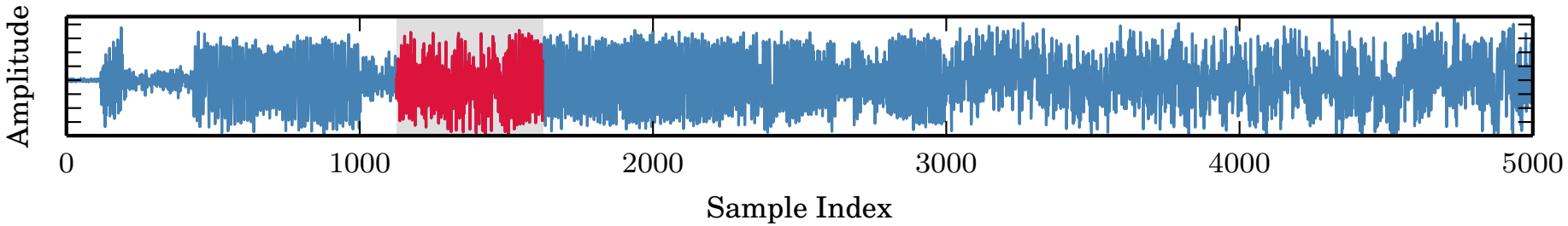
OpenSSL S/W Bulk Encryption



OpenSSL S/W Bulk Encryption Interrupted

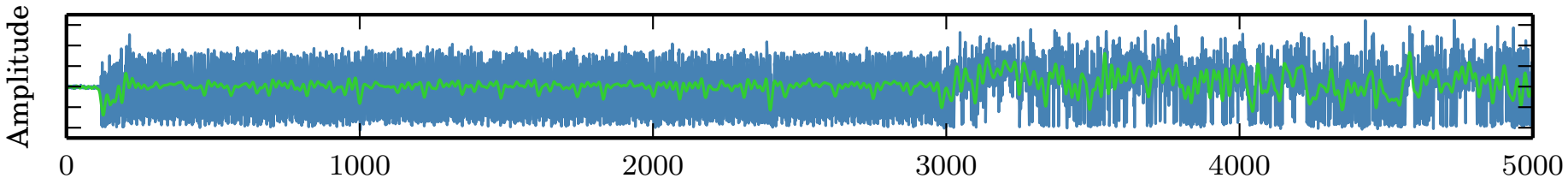


OpenSSL S/W Bulk Encryption Difference Trace

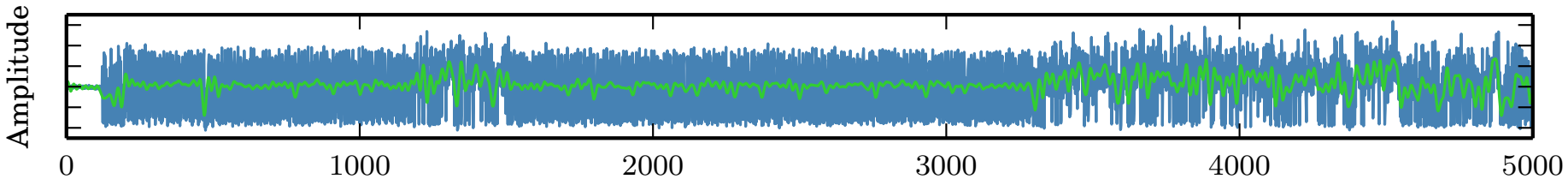


Signal Post-processing (1)

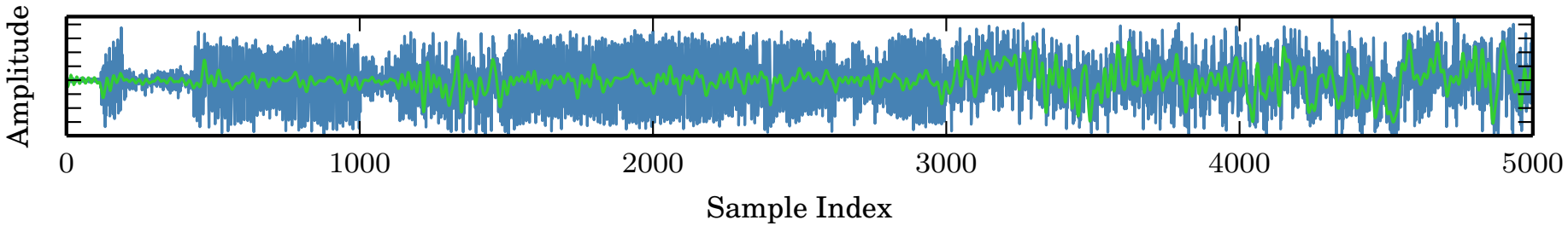
OpenSSL S/W Bulk Encryption



OpenSSL S/W Bulk Encryption Interrupted

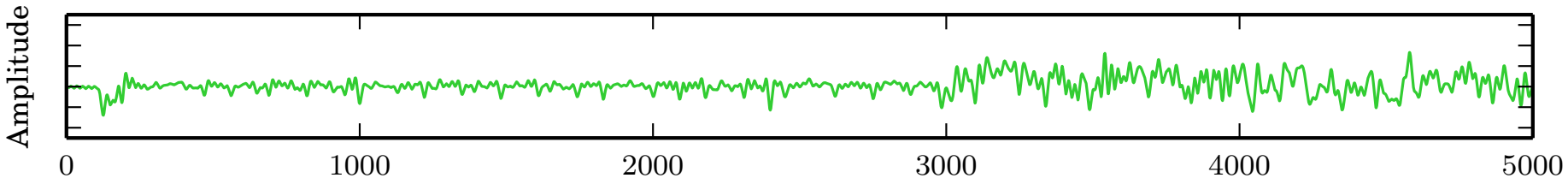


OpenSSL S/W Bulk Encryption Difference Trace

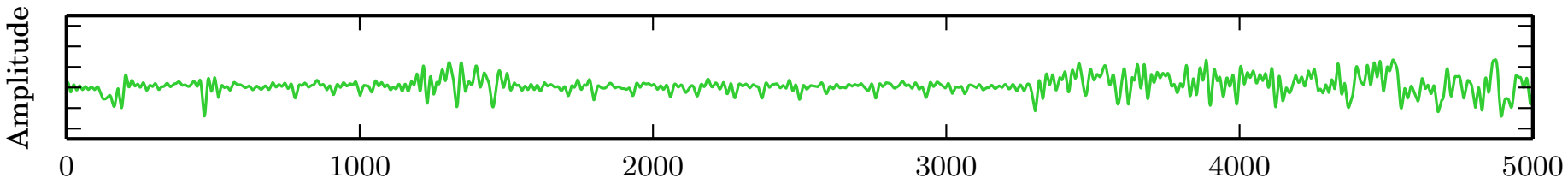


Signal Post-processing (1)

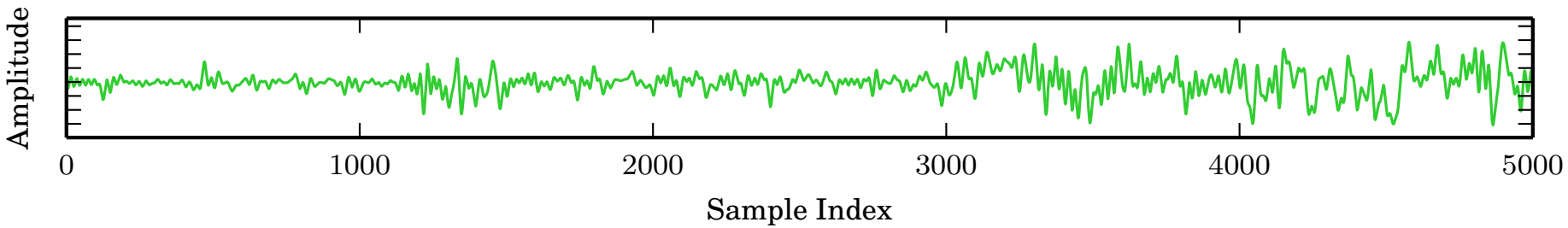
OpenSSL S/W Bulk Encryption



OpenSSL S/W Bulk Encryption Interrupted

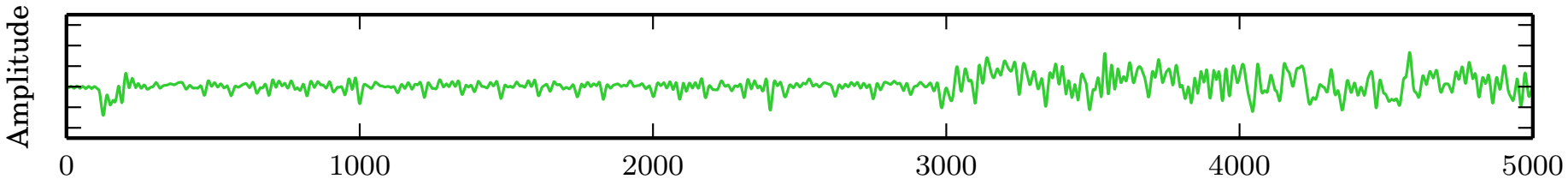


OpenSSL S/W Bulk Encryption Difference Trace

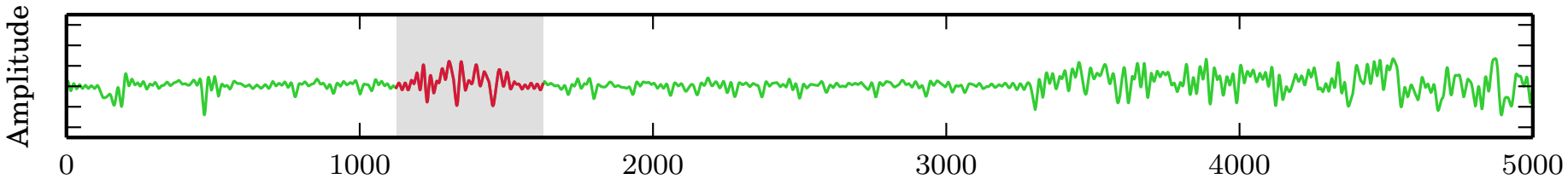


Signal Post-processing (1)

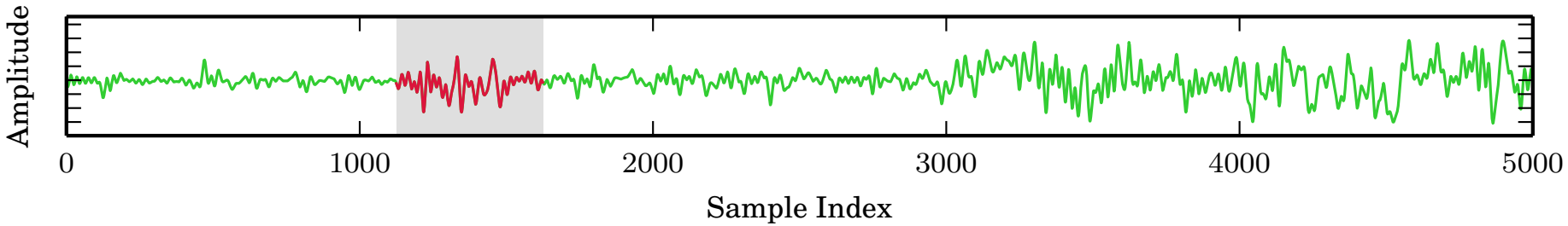
OpenSSL S/W Bulk Encryption



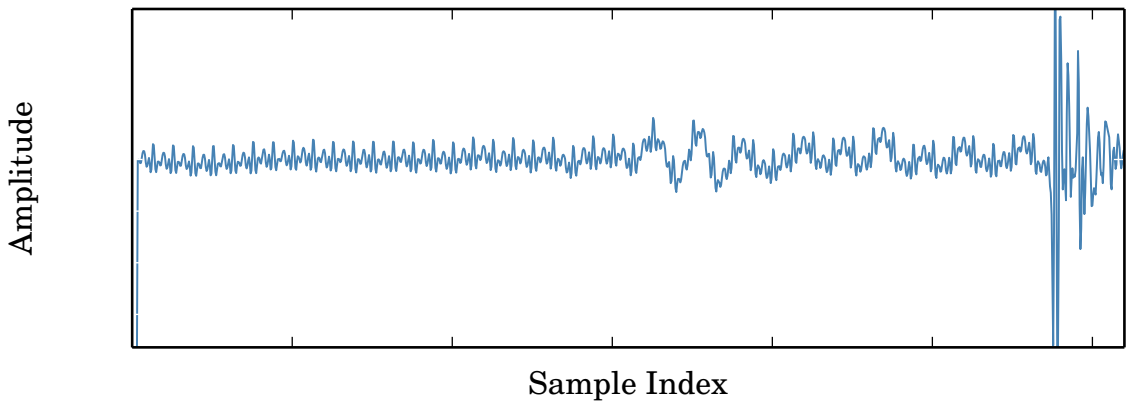
OpenSSL S/W Bulk Encryption Interrupted



OpenSSL S/W Bulk Encryption Difference Trace



Signal Post-processing (3)



Fourier Transform computation

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt$$

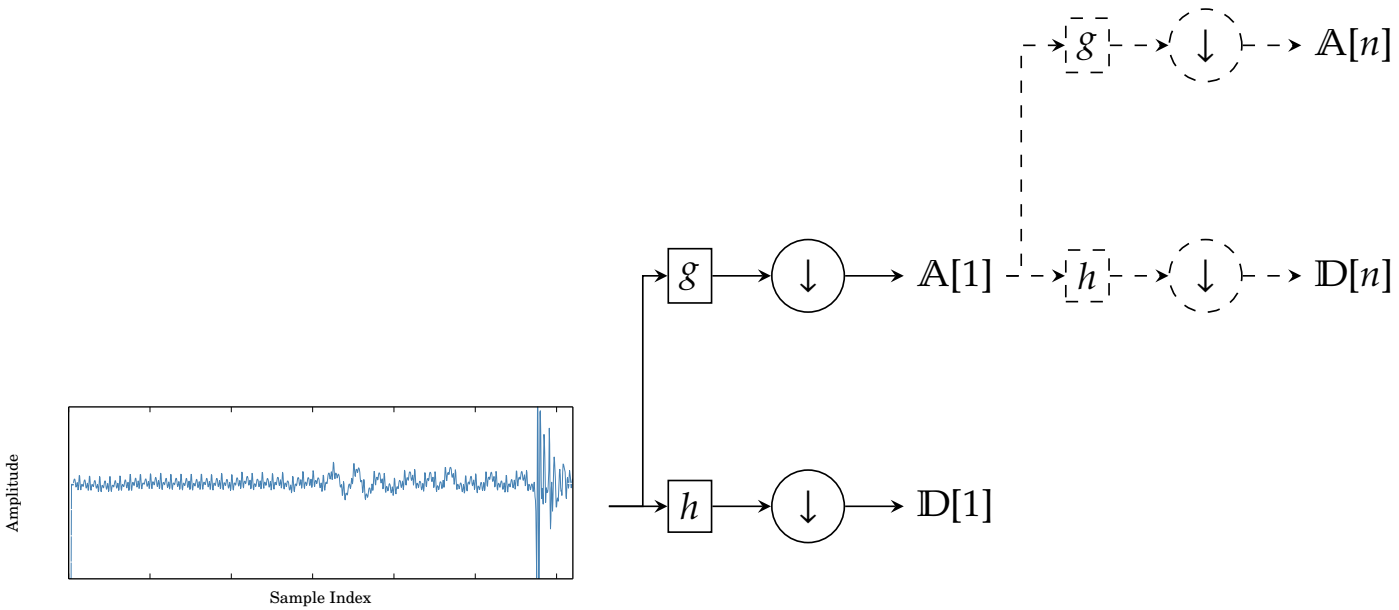
Continuous Wavelet Transform computation

$$CWT(d,s) = \frac{1}{\sqrt{d}} \int f(t) \psi\left(\frac{t-s}{d}\right) dt$$

Signal Post-processing (3)

Continuous Wavelet Transform computation

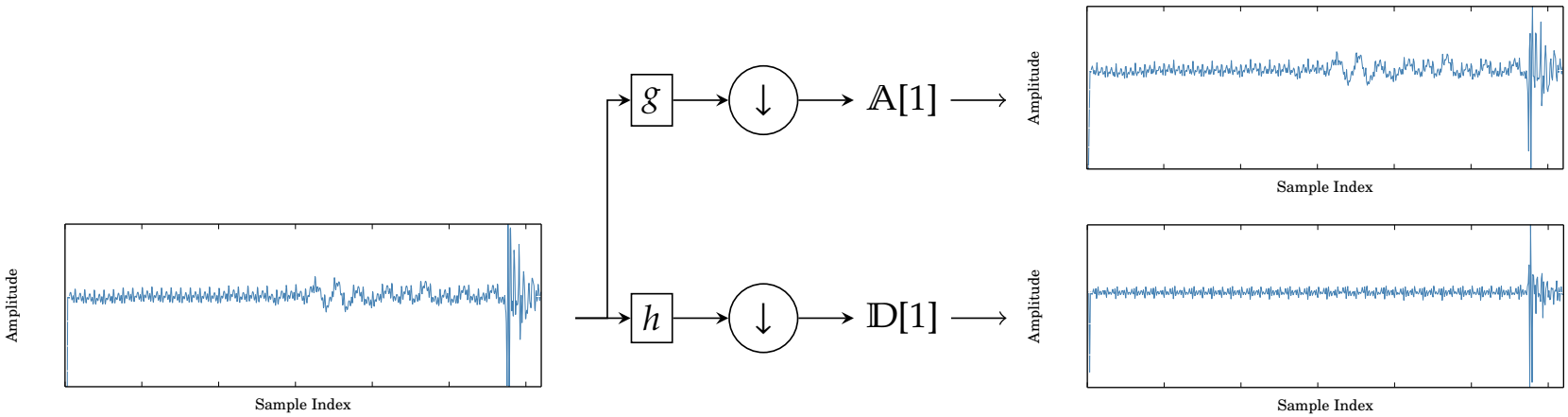
$$CWT(d,s) = \frac{1}{\sqrt{d}} \int f(t)\psi\left(\frac{t-s}{d}\right) dt$$



Signal Post-processing (3)

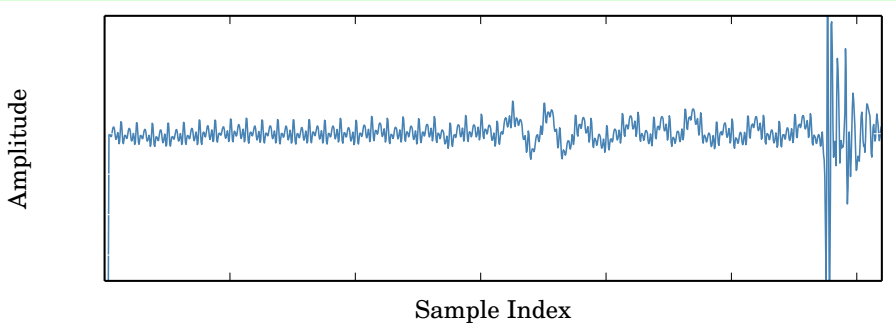
Continuous Wavelet Transform computation

$$CWT(d,s) = \frac{1}{\sqrt{d}} \int f(t)\psi\left(\frac{t-s}{d}\right) dt$$

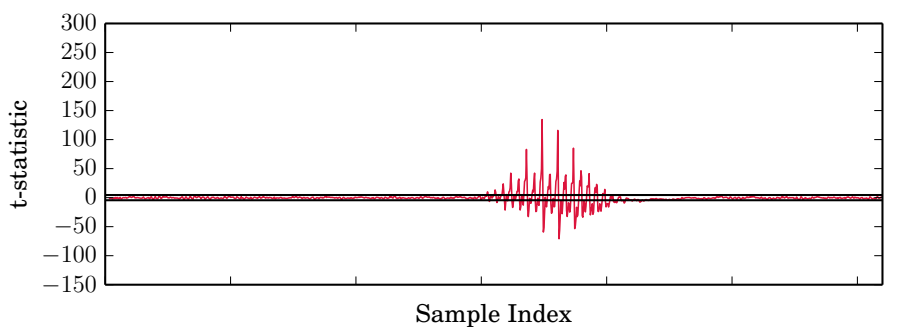


Signal Post-processing (3)

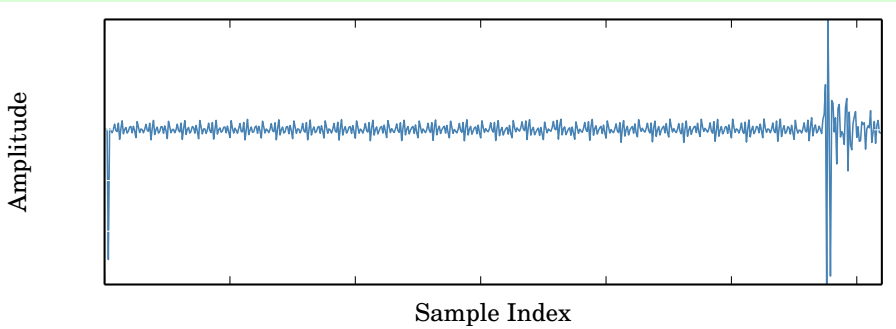
OpenSSL H/W Average



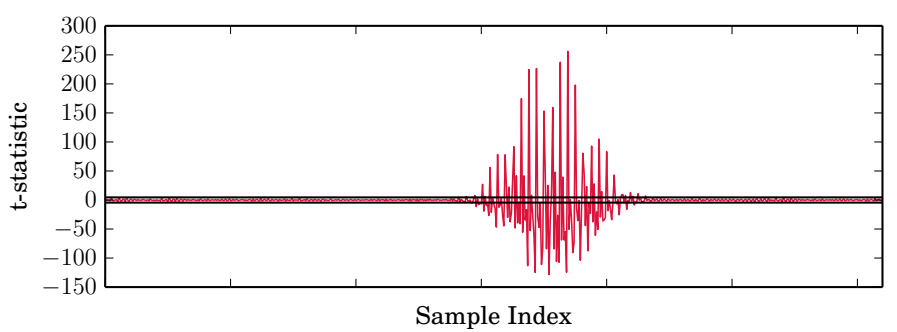
SFvR OpenSSL H/W t-test



OpenSSL H/W details (𝔻[1])



SFvR OpenSSL H/W details (𝔻[1]) t-test



Attack Results

Summary of Attack Results

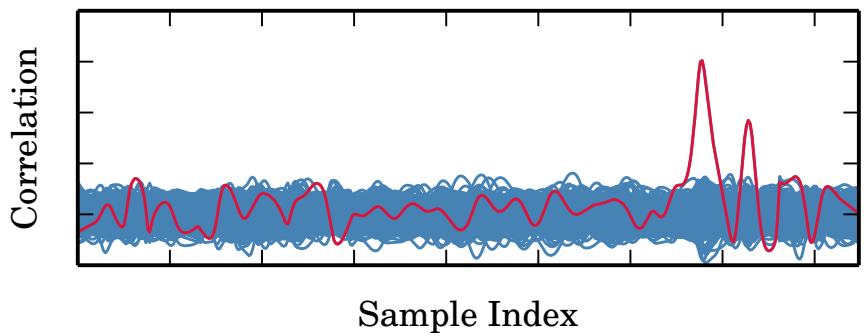
Implementation	Hardware	Trigger	Acquisitions	Data
T-tables	ARM core	GPIO-based	3000	46 kB
T-tables	ARM core	Network-based	100	400 kB
Hardware	Co-processor	DMA-based	500 000	7GB
Bit-sliced	NEON core	GPIO-based	5000	625 kB

Attack Results

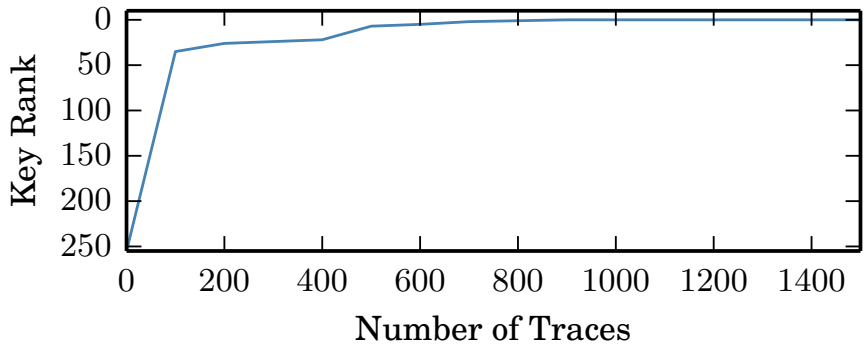
Summary of Attack Results

Implementation	Hardware	Trigger	Acquisitions	Data
T-tables	ARM core	GPIO-based	3000	46 kB
T-tables	ARM core	Network-based	100	400 kB
Hardware	Co-processor	DMA-based	500 000	7GB
Bit-sliced	NEON core	GPIO-based	5000	625 kB

OpenSSL S/W Bit Correlation



OpenSSL S/W Bit Key Rank

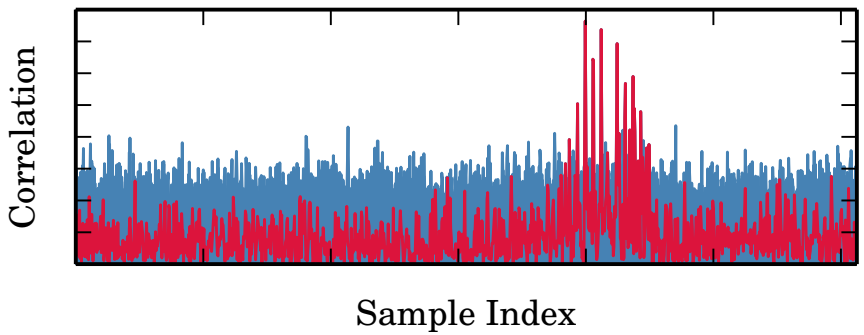


Attack Results

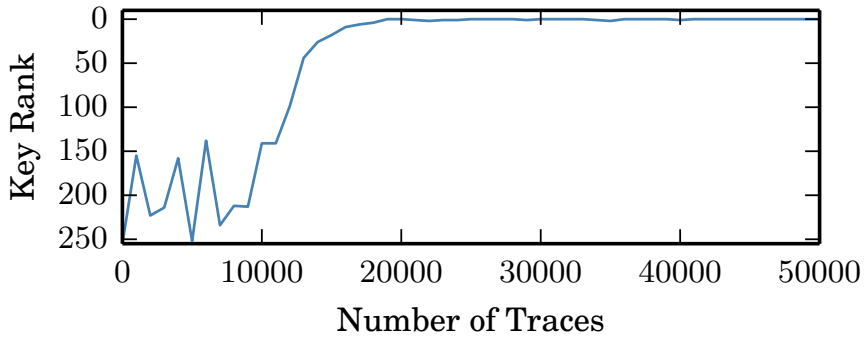
Summary of Attack Results

Implementation	Hardware	Trigger	Acquisitions	Data
T-tables	ARM core	GPIO-based	3000	46 kB
T-tables	ARM core	Network-based	100	400 kB
Hardware	Co-processor	DMA-based	50 000 ~~500 000~~	<1GB ~~7GB~~
Bit-sliced	NEON core	GPIO-based	5000	625 kB

OpenSSL H/W HD Byte Correlation



OpenSSL H/W Byte Key Rank

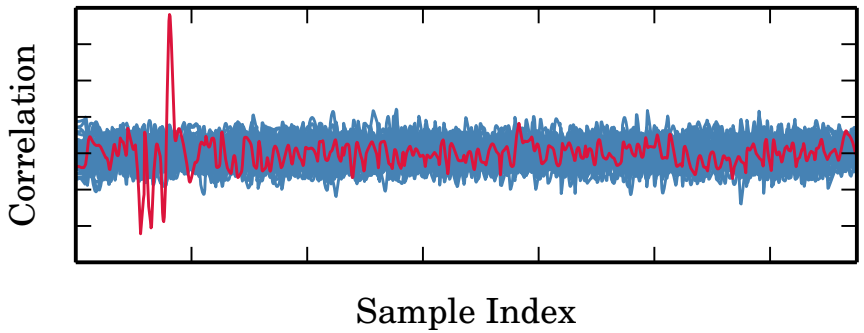


Attack Results

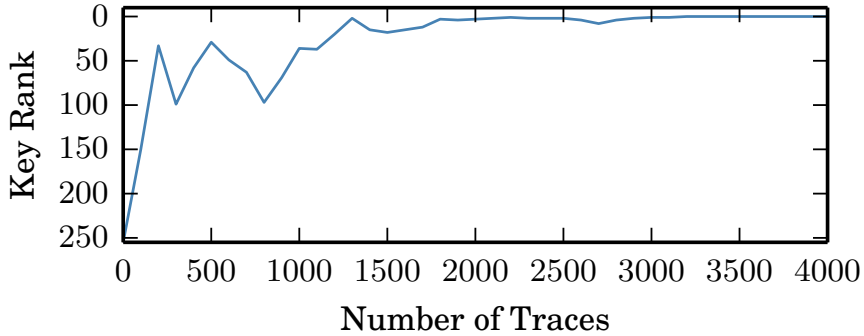
Summary of Attack Results

Implementation	Hardware	Trigger	Acquisitions	Data
T-tables	ARM core	GPIO-based	3000	46 kB
T-tables	ARM core	Network-based	100	400 kB
Hardware	Co-processor	DMA-based	50 000 ~~500 000~~	<1GB ~~7GB~~
Bit-sliced	NEON core	GPIO-based	5000	625 kB

OpenSSL Neon HW Byte Correlation



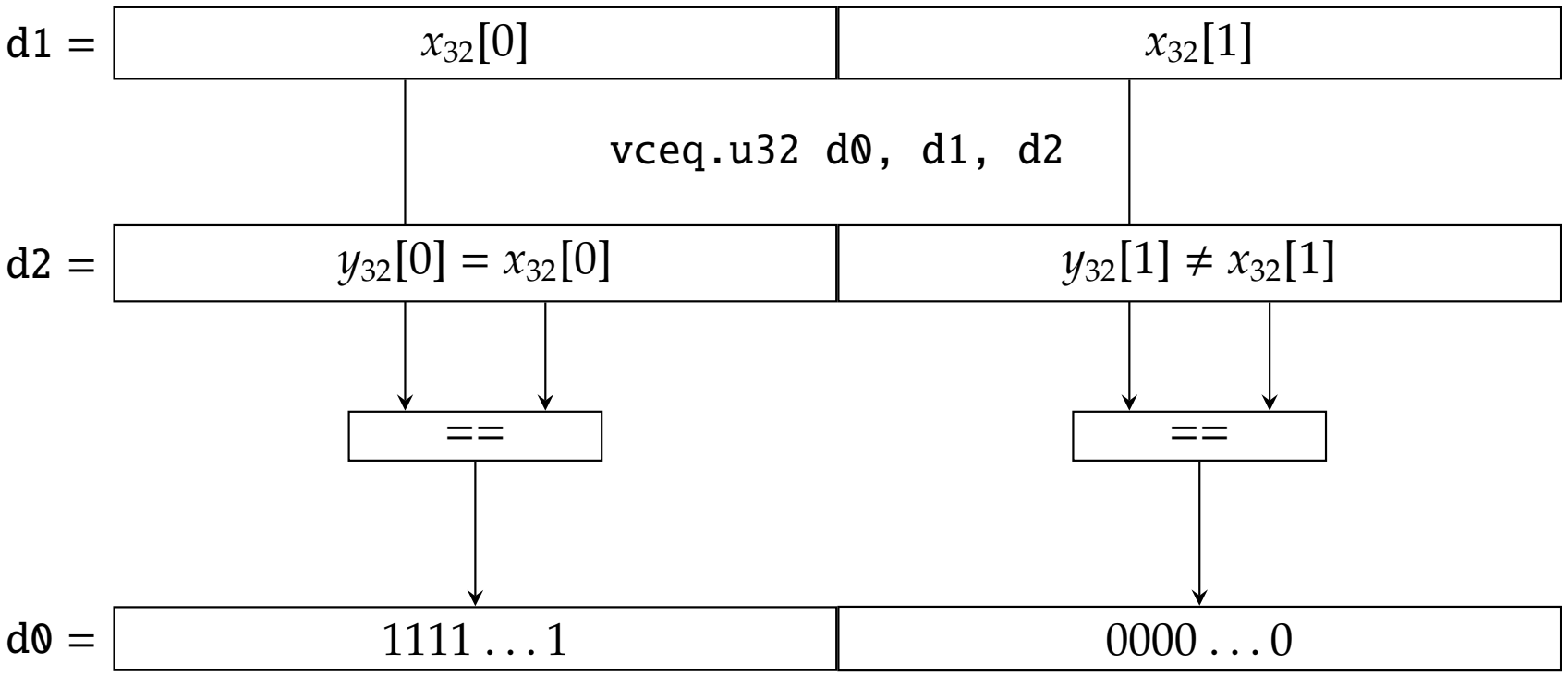
OpenSSL Neon Byte Key Rank



NEON Analysis

- NEON use-cases obviously extend beyond wide datapath for bit-slicing.
- Good: constant-time, e.g., for MAC verification

```
/* Verify tag */
A = vceqq_u32(A, LOADU(c +  0));
return 0xFFFFFFFF == (vgetq_lane_u32(A, 0) & vgetq_lane_u32(A, 1)  &
                      vgetq_lane_u32(A, 2) & vgetq_lane_u32(A, 3)) ? 0 : -1;
```

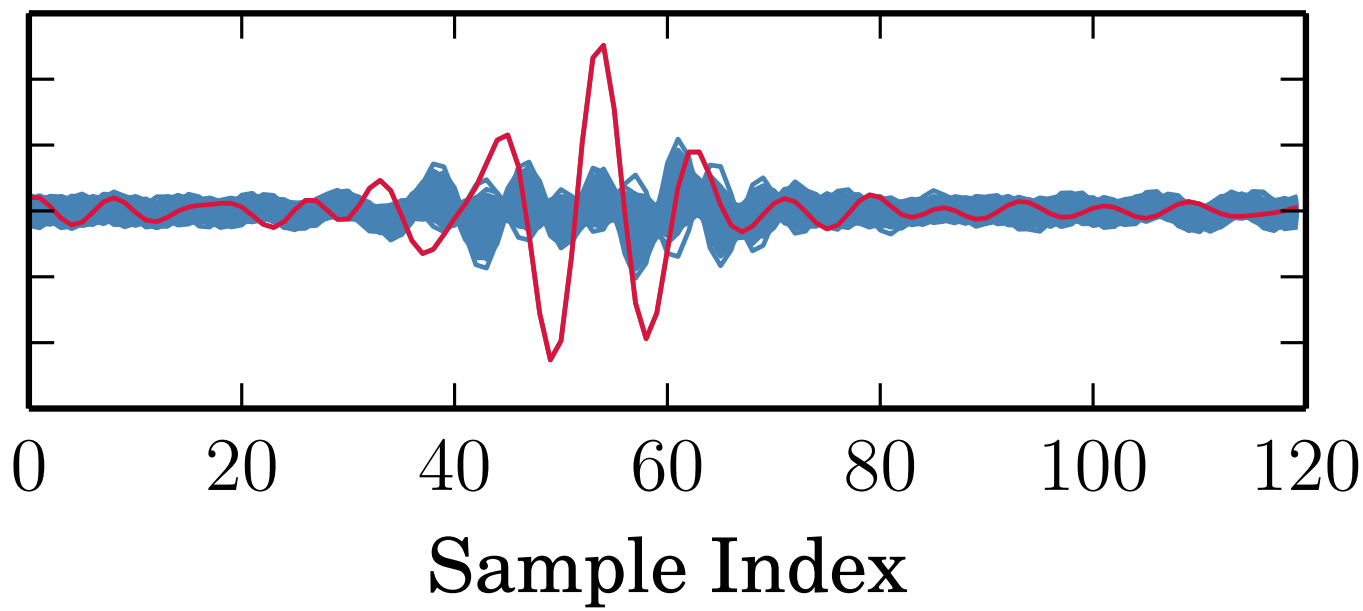


NEON Analysis

- NEON use-cases obviously extend beyond wide datapath for bit-slicing.
- Good: constant-time, e.g., for MAC verification

```
/* Verify tag */
A = vceqq_u32(A, LOADU(c +  0));
return 0xFFFFFFFF == (vgetq_lane_u32(A, 0) & vgetq_lane_u32(A, 1)  &
                      vgetq_lane_u32(A, 2) & vgetq_lane_u32(A, 3)) ? 0 : -1;
```

- Bad: strong EM-based leakage; ad hoc countermeasures can be tricky.



Conclusions

Take away points:

- Device complexity does not necessitate a complex attack. Main core and NEON attacks can both be carried out with a low-end oscilloscope and handmade probes.
- A clear attack methodology is invaluable to attack an unknown device.
- Using advanced hardware features (i.e. NEON SIMD) may accelerate an implementation but also inadvertently introduce new side-channel leaks.
- OS-level software that targets embedded systems should (long-term) consider the impact of side-channels attacks in deployment.

Future Work:

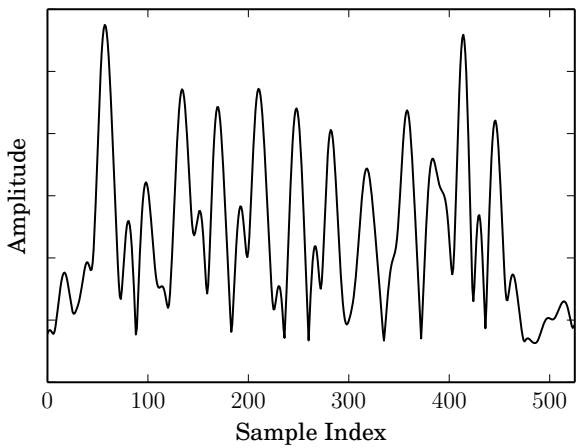
- Semi-fixed versus random tests are a great tool but how do we translate this into a leakage model?
- Hardware engine are increasingly common (ARMv8 architecture even has provisions for it). How do we synchronise on a signal we can't identify?

Questions?

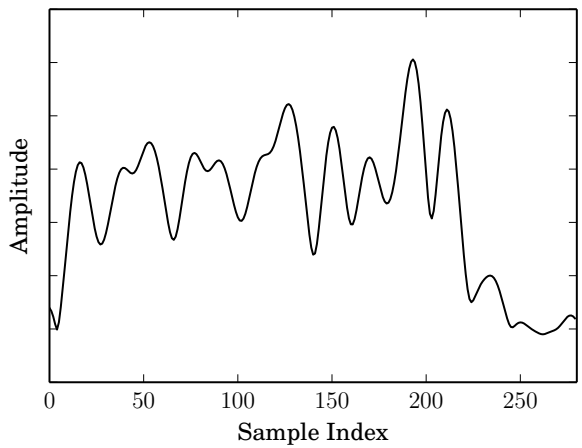

THANK YOU FOR THE TALK
JUST ONE SMALL QUESTION...
...WHAT IS THIS DPA THINGY?

Clock scaling?

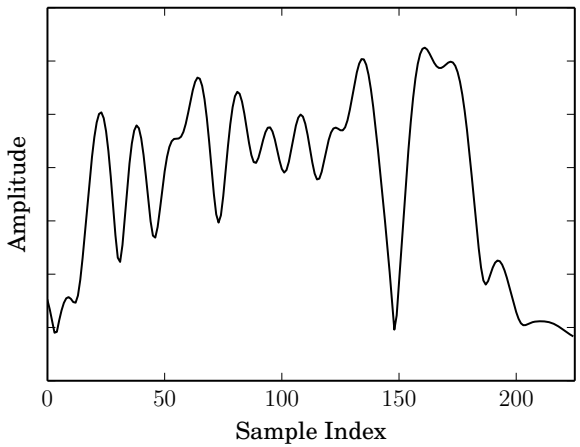
300 MHz



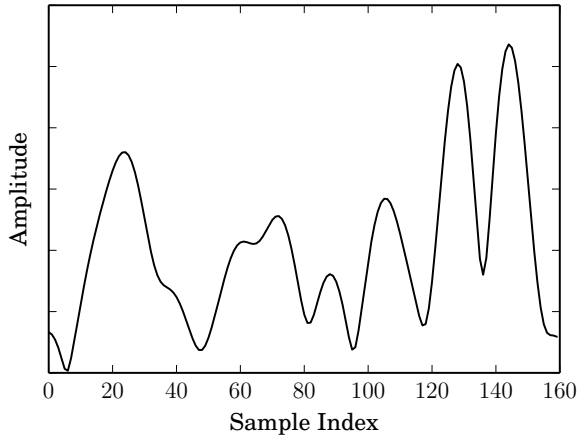
600 MHz



800 MHz

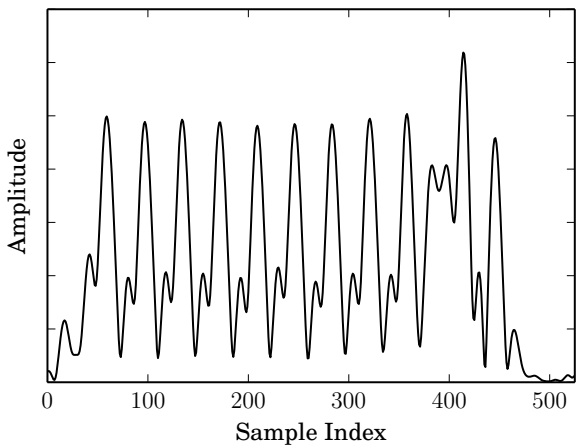


1000 MHz

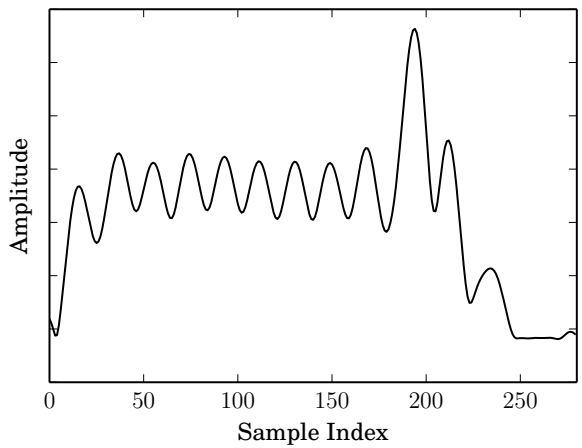


Clock scaling cleaned up!

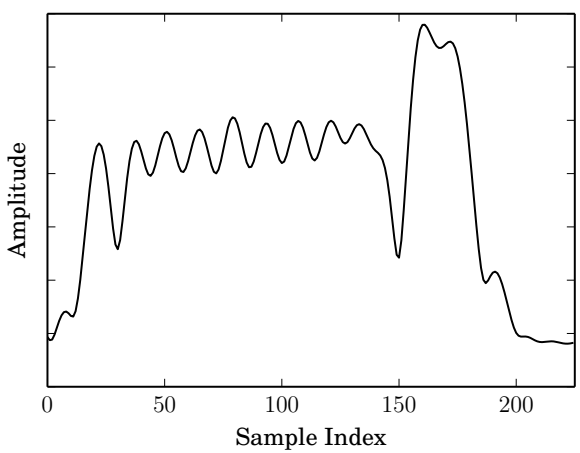
300 MHz



600 MHz



800 MHz



1000 MHz