TVLA ON NTT

OPTIMIST PQC SESSION 2025

Panasayya Yalla

OUTLINE

- What is SCA?
- What is TVLA?
- What are NTT transformations?
- Relevance of NTT for PQC side channel testing?
- TVLA on NTT in ML-DSA.
- Conclusions

SIDE CHANNEL ANALYSIS (SCA)

WHAT IS A SIDE CHANNEL?

An unintended interface for monitoring or operating a device, resulting from its physical implementation.

Intended

Card reader (3x)

Keypad

Screen

USB

Printer

Speaker

GPRS / WiFi

Power / charging

Unintended

Sound

EM radiation

Power consumption

Side channel examples

Time

Power consumption

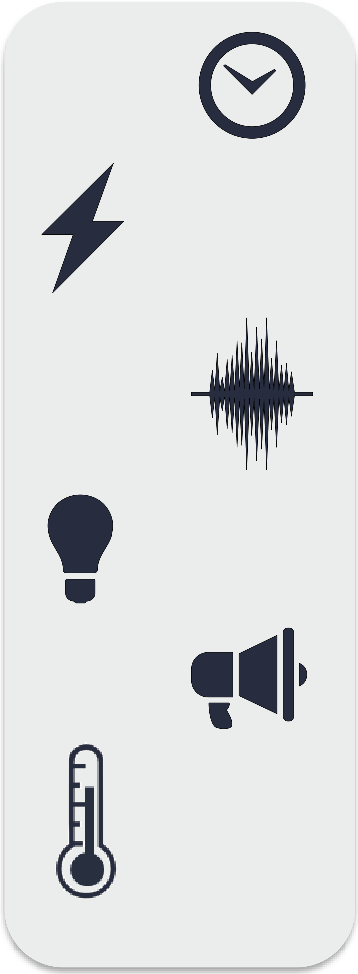
Electromagnetic radiation

Light

Sound

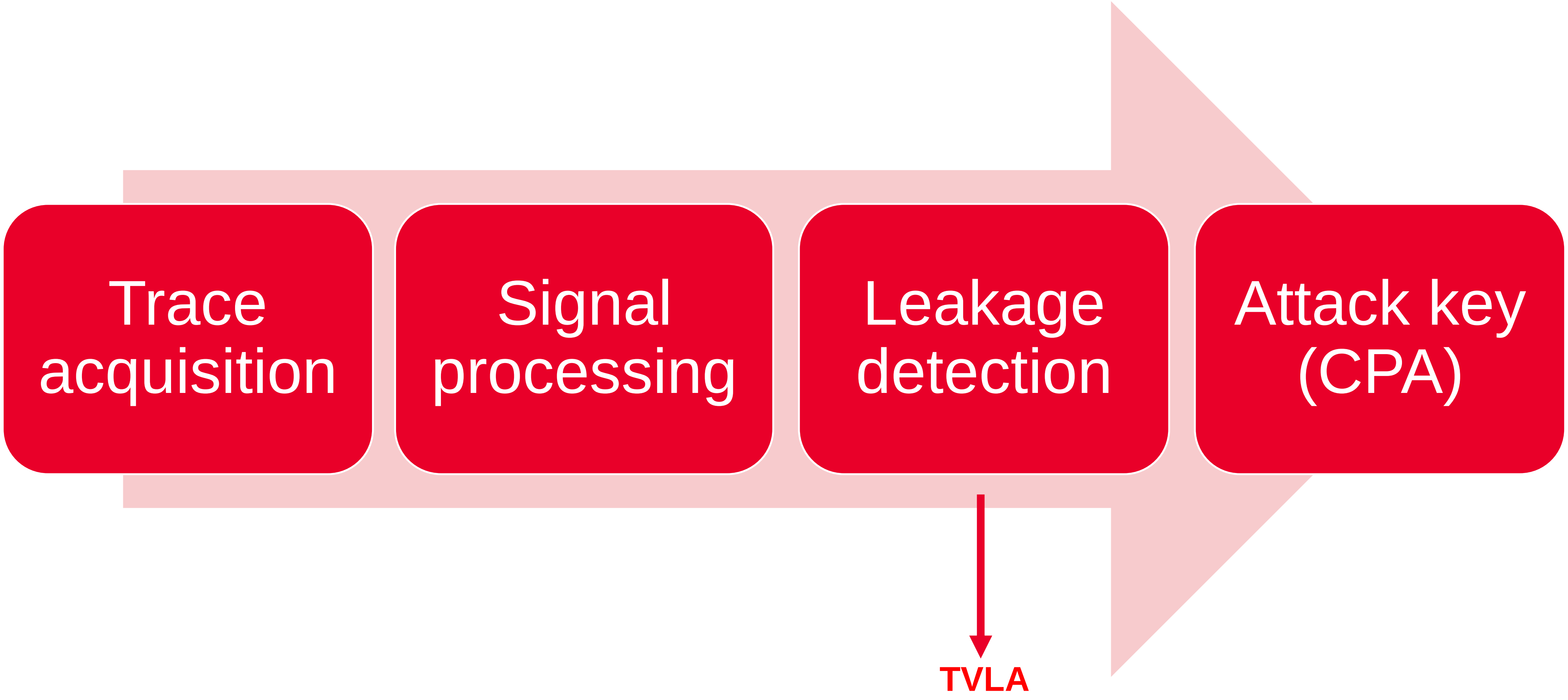
Temperature

…



TEST VECTOR LEAKAGE ASSESSMENT (TVLA)

GENERAL STEPS IN SCA



TVLA METHODOLOGY

Test **V**ector **L**eakage **A**ssessment is common industry practice for security evaluations in the context of SCA

Focus on information leakage, not key extraction

- Is there leakage: yes / no?
- Detecting leakage is much easier than extracting the key

Originally proposed for validation-based testing such as the FIPS 140-2 approach

Principle:

- Use an **efficient** test to detect difference in leakage
- **Maximize** the difference by crafted test vectors

TVLA highlights

- Acquire two sets of traces: Group A & Group B
- Make sure the statistical distribution of data in A & B are very different, *e.g.* random vs. constant plaintext
- Apply Welch's T-test to A & B:
  - Standard statistical test of equality tailored for population with unequal variances

$$t(I) = \frac{X_A(I) - X_B(I)}{\sqrt{\frac{S_A^2(I)}{N_A} + \frac{S_B^2(I)}{N_B}}}$$

- T-test spikes → difference in signal distribution detected → leakage present
- Device is leaky if t > 4.5, for 99.999% confidence

GROUPS FOR TVLA TESTING

- Fixed vs random,
- Semi-fixed vs semi-fixed
- Biased vs random
  - Hamming Weight (HW)
  - Hamming Distance (HD)
  - Identity (ID)

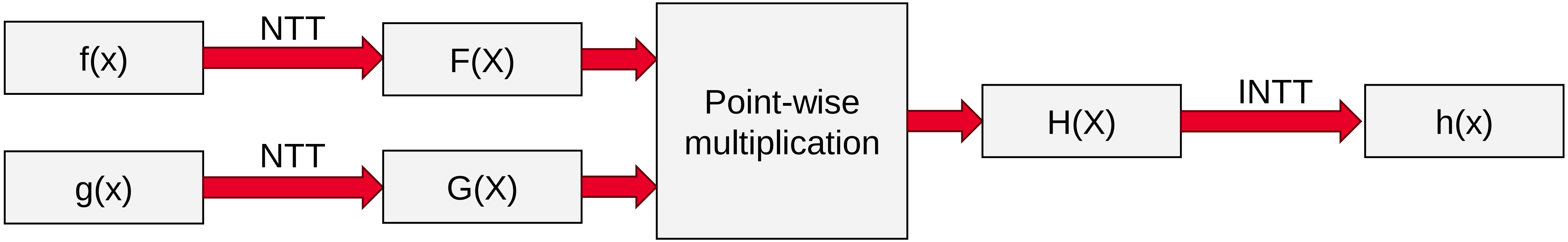
NUMBER THEORETIC TRANSFORM (NTT)

NUMBER THEORETIC TRANSFORM (NTT)

- Schoolbook multiplication $O(n^2)$

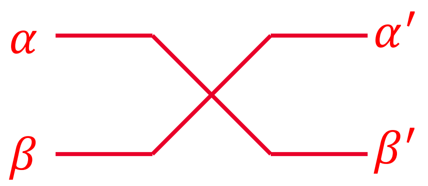
$$
\begin{array}{r}
1 + 2x + 3x^2 + 4x^3 \\
5 + 6x + 7x^2 + 8x^3 \\
\hline
8x^3 + 16x^4 + 24x^5 + 32x^6 \\
7x^2 + 14x^3 + 21x^4 + 28x^5 \quad \\
6x + 12x^2 + 18x^3 + 24x^4 \quad\quad \\
5 + 10x + 15x^2 + 20x^3 \quad\quad\quad \\
\hline
5 + 16x + 34x^2 + 60x^3 + 61x^4 + 52x^5 + 32x^6
\end{array}
$$

- Fast Fourier Transform (FFT) over finite fields
- Divide-and-conquer $O(n \log n)$
- Requires NTT and INTT transformations
- Workflow for computing $f(x) \cdot g(x) = h(x)$
- In NTT domain $(1X + 2X^2 + 3X^3) \cdot (4X + 5X^2 + 6X^3) = 4X + 10X^2 + 18X^3$



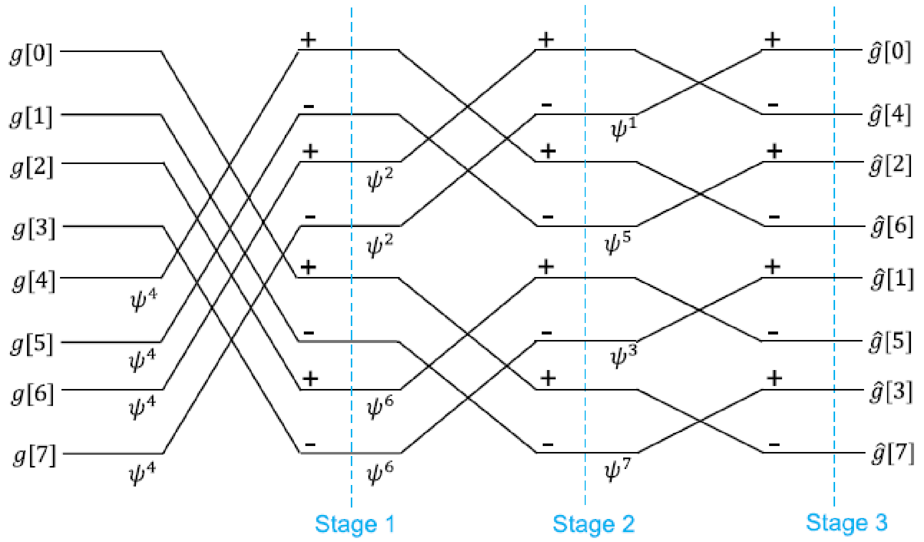
NTT BUTTERFLIES

- Divide: NTT has $\log_2 N$ stages
- Conquer: each stage has $\frac{N}{2}$ butterflies, each butterfly has 2 inputs α, β + 2 outputs α', β' + a twiddle constant ζ



$$\alpha' = \alpha + \zeta\beta, \qquad \beta' = \alpha - \zeta\beta$$

Example: $N = 8$, three stages



A. Satriawan, I. Syafalni, R. Mareta, I. Anshori, W. Shalannanda and A. Barra, "Conceptual Review on Number Theoretic Transform and Comprehensive Review on Its Implementations," in IEEE Access, vol. 11, pp. 70288-70316, 2023, doi: 10.1109/ACCESS.2023.3294446.

PQC NIST STANDARD: ML-DSA

Algorithm 7 ML-DSA.Sign_internal(sk, M', rnd)

Deterministic algorithm to generate a signature for a formatted message M'.

Input: Private key $sk \in \mathbb{B}^{32+32+64+32\cdot((\ell+k)\cdot \text{bitlen}\,(2\eta)+dk)}$, formatted message $M' \in \{0,1\}^*$, and per message randomness or dummy variable $rnd \in \mathbb{B}^{32}$.
Output: Signature $\sigma \in \mathbb{B}^{\lambda/4+\ell\cdot 32\cdot(1+\text{bitlen}\,(\gamma_1-1))+\omega+k}$.

$(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow$ skDecode(sk)
$\hat{\mathbf{s}}_1 \leftarrow$ NTT($\mathbf{s}_1$)
$\hat{\mathbf{s}}_2 \leftarrow$ NTT($\mathbf{s}_2$)
$\hat{\mathbf{t}}_0 \leftarrow$ NTT($\mathbf{t}_0$)
$\hat{\mathbf{A}} \leftarrow$ ExpandA(ρ) ▷ $\mathbf{A}$ is generated and stored in NTT representation as $\hat{\mathbf{A}}$
$\mu \leftarrow$ H(BytesToBits(tr)$||M'$, 64) ▷ message representative that may optionally be computed in a different cryptographic module
$\rho'' \leftarrow$ H($K||rnd||\mu$, 64) ▷ compute private random seed
$\kappa \leftarrow 0$ ▷ initialize counter κ
$(\mathbf{z}, \mathbf{h}) \leftarrow \bot$
while $(\mathbf{z}, \mathbf{h}) = \bot$ **do** ▷ rejection sampling loop
  $\mathbf{y} \in R_q^\ell \leftarrow$ ExpandMask(ρ'', κ)
  $\mathbf{w} \leftarrow$ NTT^{-1}($\hat{\mathbf{A}} \circ$ NTT($\mathbf{y}$))
  $\mathbf{w}_1 \leftarrow$ HighBits($\mathbf{w}$) ▷ signer's commitment
  ▷ HighBits is applied componentwise (see explanatory text in Section 7.4)
  $\tilde{c} \leftarrow$ H($\mu||$w1Encode($\mathbf{w}_1$), $\lambda/4$) ▷ commitment hash
  $c \in R_q \leftarrow$ SampleInBall($\tilde{c}$) ▷ verifier's challenge
  $\hat{c} \leftarrow$ NTT(c)
  $\langle\langle c\mathbf{s}_1\rangle\rangle \leftarrow$ NTT^{-1}($\hat{c} \circ \hat{\mathbf{s}}_1$)
  $\langle\langle c\mathbf{s}_2\rangle\rangle \leftarrow$ NTT^{-1}($\hat{c} \circ \hat{\mathbf{s}}_2$)
  $\mathbf{z} \leftarrow \mathbf{y} + \langle\langle c\mathbf{s}_1\rangle\rangle$ ▷ signer's response
  $\mathbf{r}_0 \leftarrow$ LowBits($\mathbf{w} - \langle\langle c\mathbf{s}_2\rangle\rangle$)
  ▷ LowBits is applied componentwise (see explanatory text in Section 7.4)
  if $||\mathbf{z}||_\infty \geq \gamma_1 - \beta$ **or** $||\mathbf{r}_0||_\infty \geq \gamma_2 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \bot$ ▷ validity checks
  else
    $\langle\langle c\mathbf{t}_0\rangle\rangle \leftarrow$ NTT^{-1}($\hat{c} \circ \hat{\mathbf{t}}_0$)
    $\mathbf{h} \leftarrow$ MakeHint($-\langle\langle c\mathbf{t}_0\rangle\rangle, \mathbf{w} - \langle\langle c\mathbf{s}_2\rangle\rangle + \langle\langle c\mathbf{t}_0\rangle\rangle$) ▷ Signer's hint
    ▷ MakeHint is applied componentwise (see explanatory text in Section 7.4)
    if $||\langle\langle c\mathbf{t}_0\rangle\rangle||_\infty \geq \gamma_2$ **or** the number of 1's in $\mathbf{h}$ is greater than ω, **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \bot$
    end if
  end if
  $\kappa \leftarrow \kappa + \ell$ ▷ increment counter
end while
$\sigma \leftarrow$ sigEncode($\tilde{c}$, $\mathbf{z}$ mod$^{\pm}q$, $\mathbf{h}$)
return σ

https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf

Private

NTT and INTT

Template attacks

- Each polynomial has 256 coefficients
- Each coefficient is an element of mod q, where q = 8380417 < 2^23
- ML-DSA NTT has 8 stages, each stage has 128 butterflies

TVLA ON NTT MODELS

TVLA FOR NTT – HW MODEL

Find NTT input such that the HW of intermediate values in i-th Stage is low (set to be h)

Steps:

1. Choose a desired HW h in stage i.
2. Find a random integer partition of h, that is, $h = h_0 + h_1 + \cdots + h_{255}$ and place partitions in a vector with 256 entries $(h_0, h_1, \ldots, h_{255})$.
3. As each NTT Stage is bijective, evolve the vector $(h_0, h_1, \ldots, h_{255})$ from i-th stage to $(i - 1)$-th stage, then to $(i - 2)$-th stage, until hitting the input of NTT.

What is a good HW □ <200 is good enough

TVLA FOR NTT – HD MODEL

Find NTT input such that the HD of intermediate values before and after i-th Stage is lower than a threshold h

Do we require control the HW of XOR of values before and after a linear transformation? Yes and No

- Yes, with a "zero trick", by setting $\beta = 0$, we have $A\begin{pmatrix}\alpha \\ 0\end{pmatrix} = \begin{pmatrix}\alpha \\ \alpha\end{pmatrix}$, with $A = \begin{pmatrix}1 & \zeta \\ 1 & -\zeta\end{pmatrix}$. So essentially, we gain a HD of $\mathrm{H}D(\alpha)$ for each butterfly.
- No, bruteforce every block in a stage

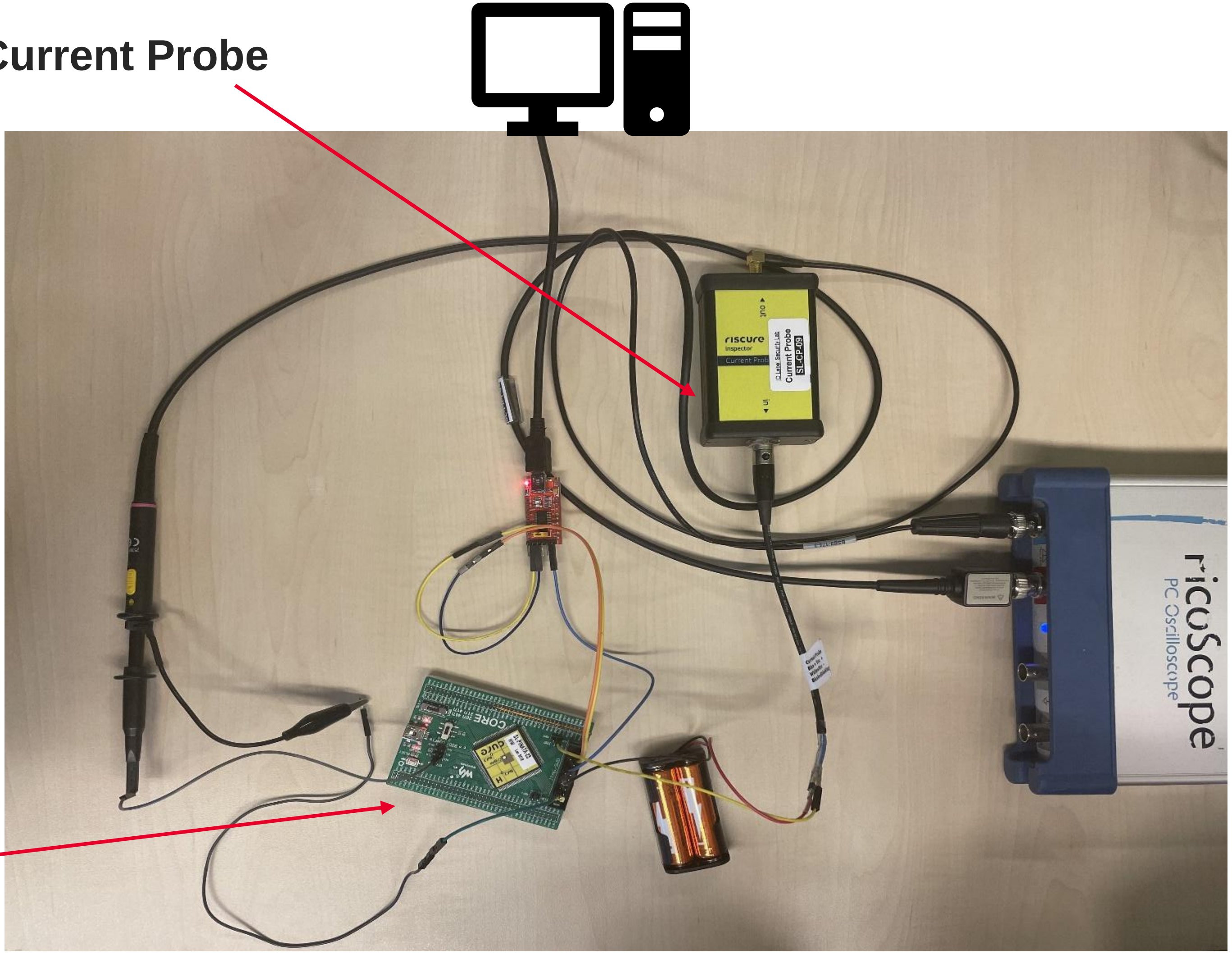
Steps:

1. Fix a desired HD threshold h in stage i.
2. Find a random integer partition of h, that is, $h = h_0 + h_1 + \cdots + h_{127}$ and place partitions in a vector with 256 entries, such that h_0 becomes the α for the 0-th butterfly at Stage i, that h_0 becomes the α for the 1-st butterfly at Stage i+1, and so on.
3. Evolve the vector back until finding the input of NTT.

EXPERIMENT SETUP

Running standalone NTT implementation

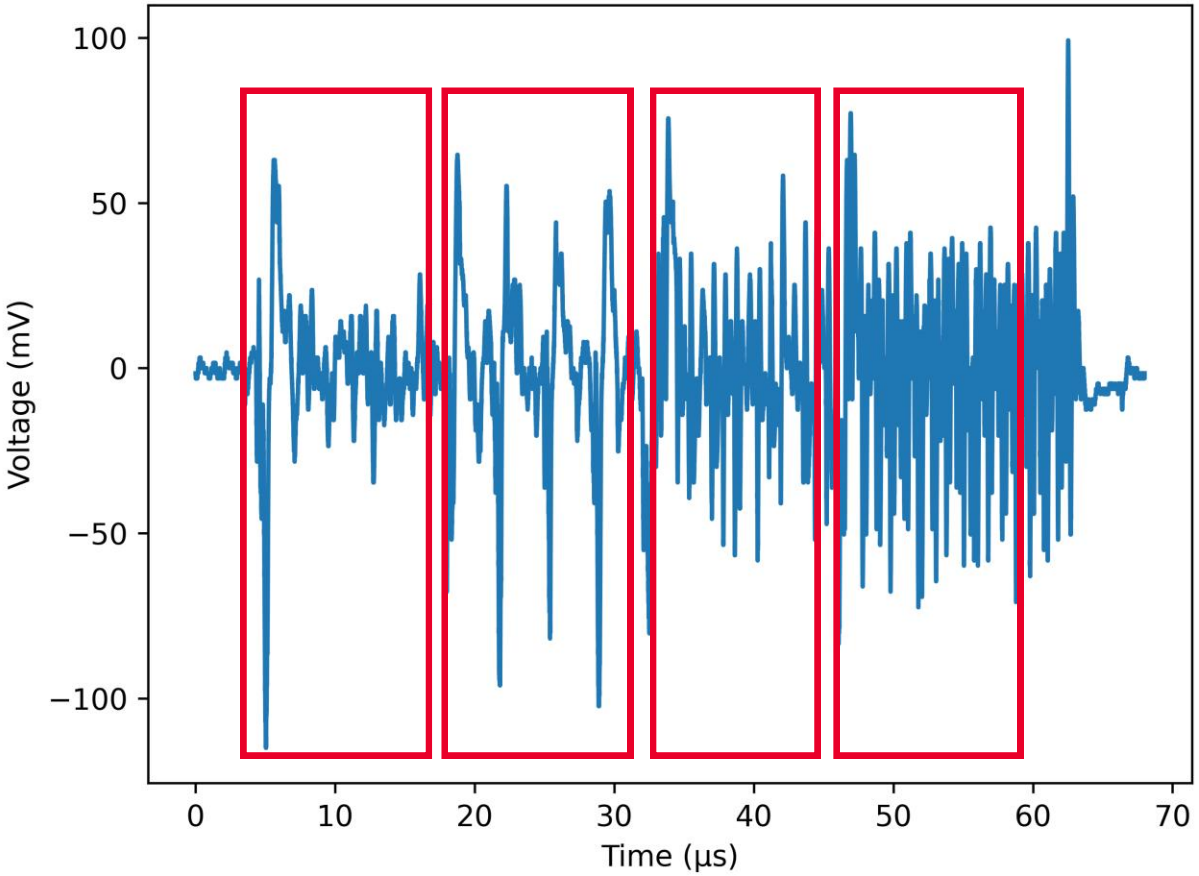
Current Probe



Test Target
(Piñata)

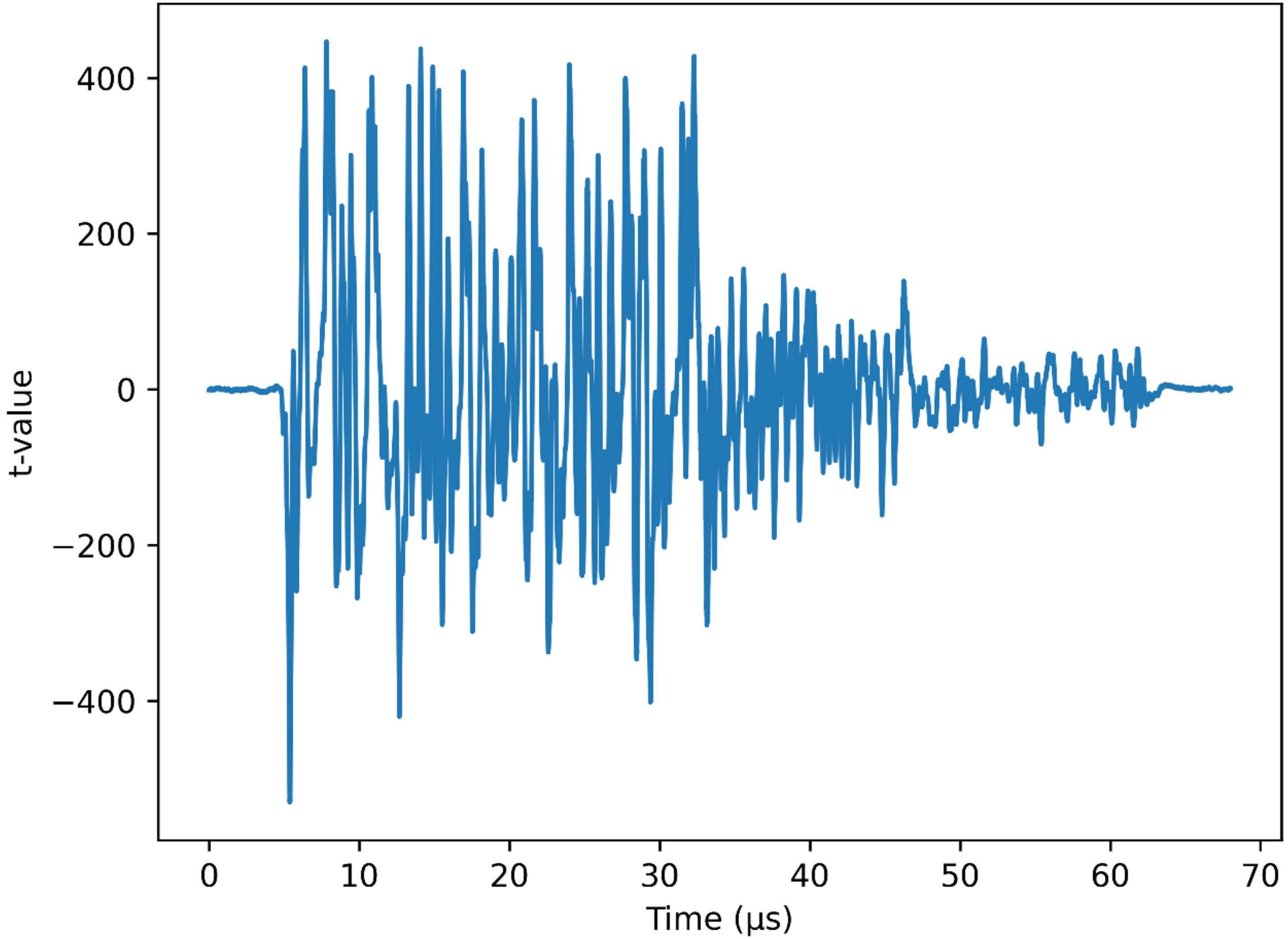
[GKS20] Greconici, D.O.C., Kannwischer, M.J., Sprenkels, A.: Compact dilithium implementations on Cortex-M3 and Cortex-M4. IACR Trans. Cryptogr. Hardw. Embed. Syst. 2021(1), 1–24 (2020)

AN EXAMPLE POWER TRACE



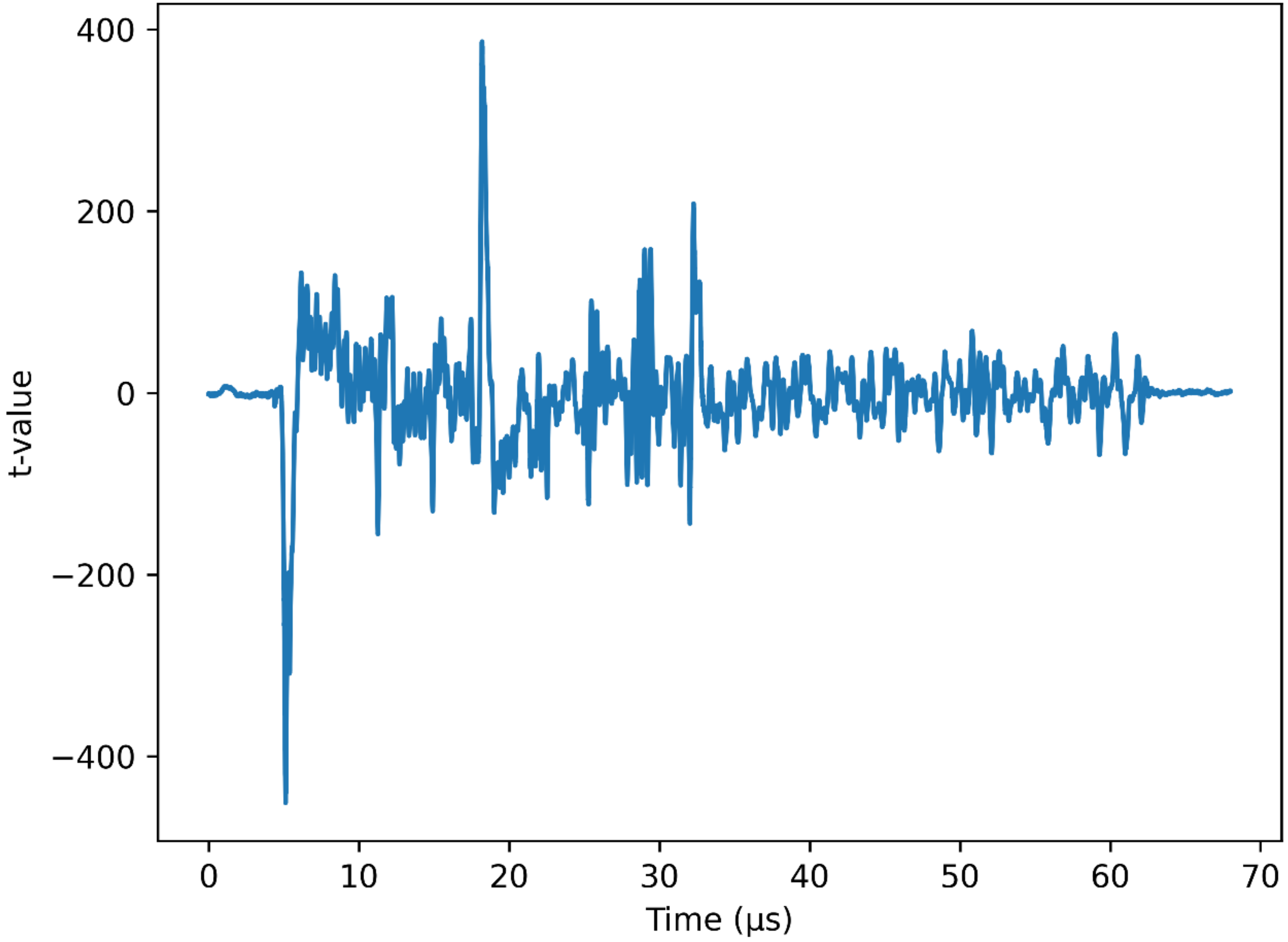
FIXED VS RANDOM TVLA (HW MODEL)

Biasing: Stage 2 with HW == 94

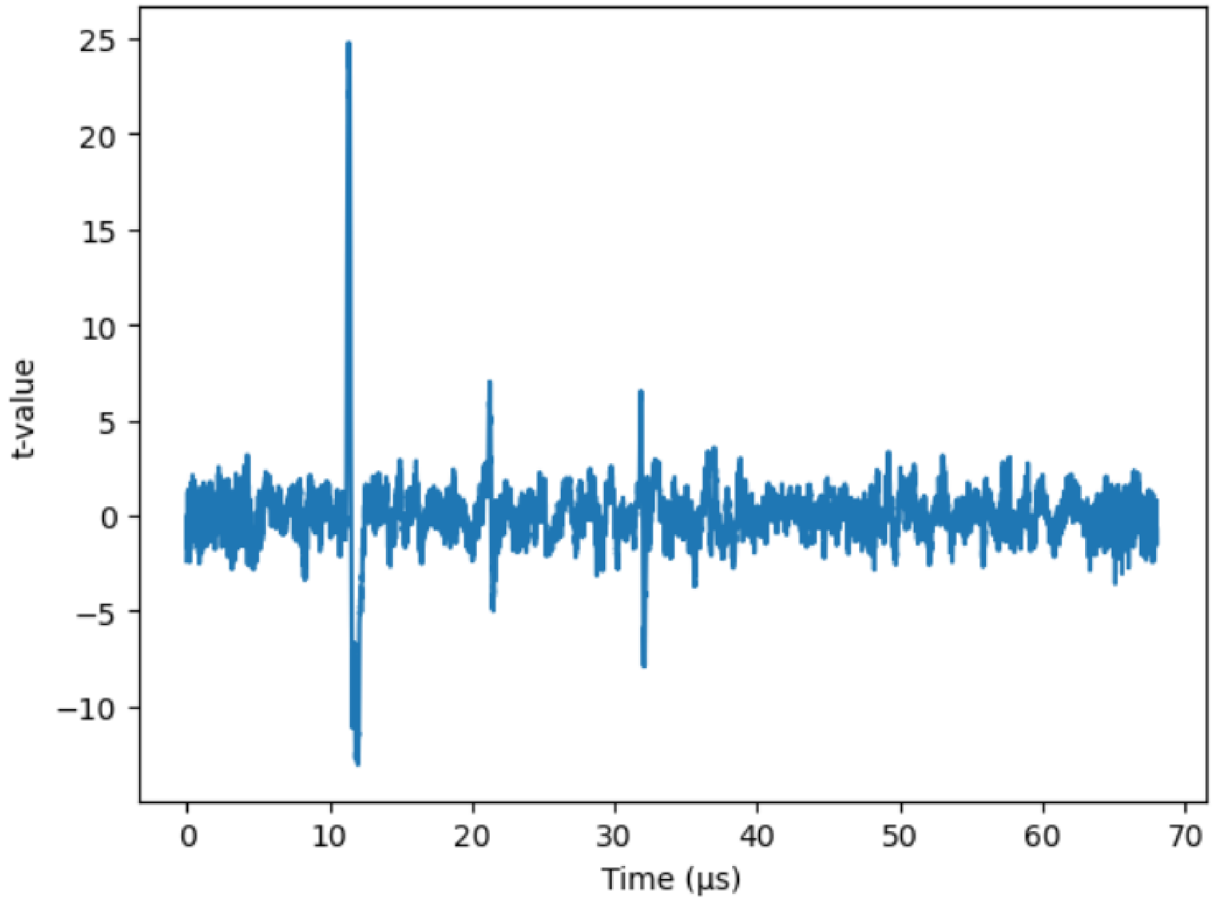


FIXED VS RANDOM TVLA (HD MODEL)

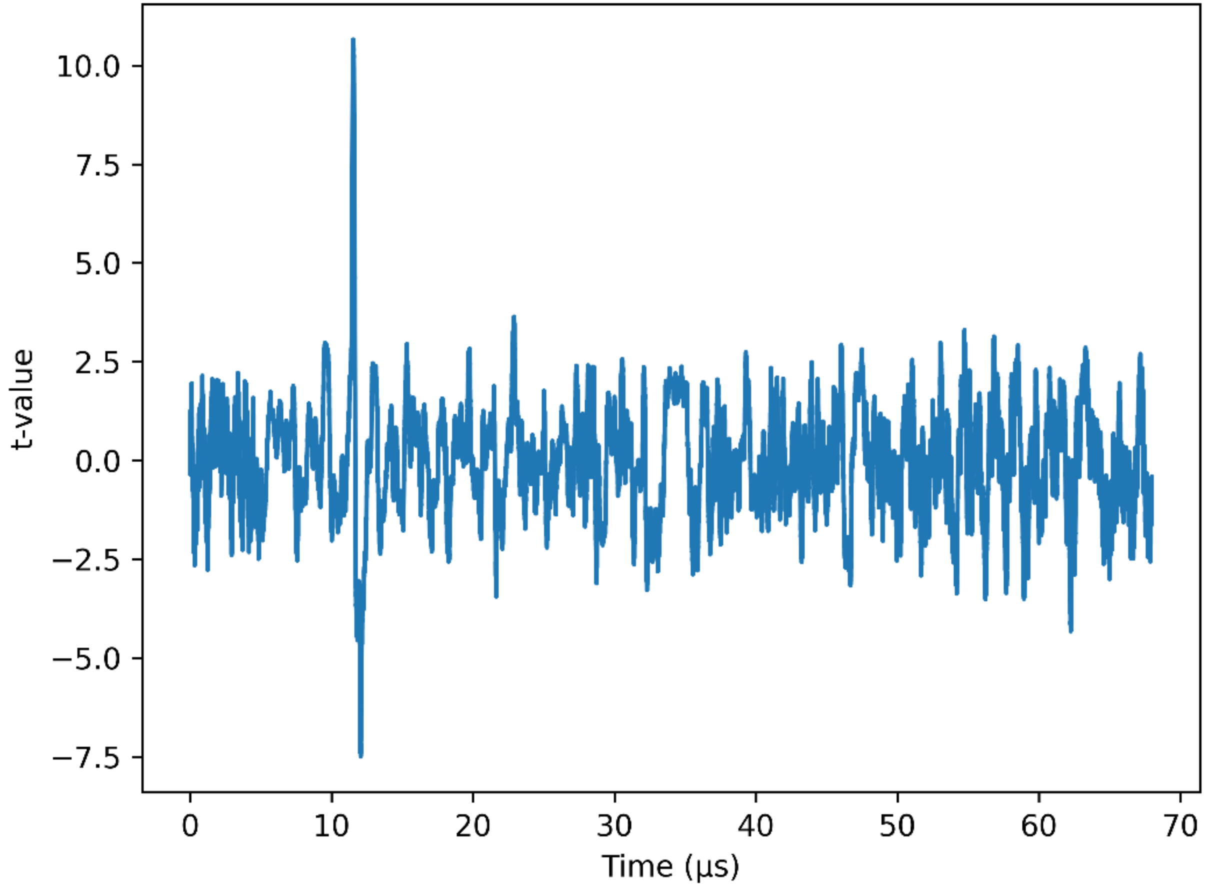
Biasing: Stage 2 with HD == 1055



SEMI-FIXED VS RANDOM TVLA (FIXING 4 BUTTERFLIES)



SEMI-FIXED VS RANDOM TVLA (FIXING 1 COEFFICIENT)



CONCLUSIONS

- NTT implementation leaks
- Leakage is mainly from memory operations
- Leakage from computing the buffterflies are negligible.
- Montgomery multiplication makes it harder to generate biased vectors.

Questions?

KEYSIGHT

Thank you