

Post-Quantum vs. AVA_VAN

Dr. Markku-Juhani Saarinen
ICCC 2023 – November 1, 2023

Motivation: Securing Dilithium for AVA_VAN.3+

- **AVA_VAN: Common Criteria / Vulnerability Analysis**
 - **Algorithm-agnostic**, but realistic (costing the best of attack paths.)
 - Requirement for Secure elements, Smart Cards, Root of Trust, etc.
- **Dilithium (ML-DSA): Replacing ECDSA and RSA**
 - **PQC Transition:** Fitting Dilithium into high assurance applications.
 - Countermeasures are more complex than those of ECDSA and RSA.

DISCLAIMER: *This talk reflects only my personal opinions and experiences from engineering and attacking PQC modules and my current understanding of the research literature and the certification processes. No claims about specific commercial products or their security are made.*

Special thanks to Timo Zijlstra (PQShield). Errors and omissions are all mine.

Standards: Post-Quantum Crypto (FIPS IPD)



- **FIPS 203 ML-KEM (Kyber)**
Replaces EC(DH), RSA in key establishment.
- **FIPS 204 ML-DSA (Dilithium)**
Replaces EC(DSA), RSA signatures (primary signature algorithm.)
- **FIPS 205 SLH-DSA (SPHINCS+)**
Stateless hash-based signatures (not “drop-in” for RSA, ECDSA.)
- Older: **NIST SP 800-208** Stateful Hash Based Signatures **XMSS** and **LMS**.
- Work in Progress: **FIPS 206 FN-DSA (Falcon)**

Dilithium

August 24, 2023

ML-DSA ~ “Dilithium”

Initial Public Draft Standard

Dilithium has been in public evaluation since 2017, selected as standard in 2022.

Based on structured lattices (MLWE and SelfTargetMSIS.)

FIPS 204 (DRAFT)

MODULE-LATTICE-BASED DIGITAL SIGNATURE STANDARD

Federal Information Processing Standards Publication 204 Specification for the Module-Lattice-Based Digital Signature Standard

Table of Contents

1	Introduction	1
1.1	Purpose and Scope	1
1.2	Context	1
1.3	Differences Between the ML-DSA Standard and CRYSTALS-DILITHIUM	2
1.3.1	Differences Between Version 3.1 and the Round 3 Version of CRYSTALS-DILITHIUM	2
1.3.2	Differences Between the ML-DSA Standard and Version 3.1 of CRYSTALS-DILITHIUM	2


Available: <https://doi.org/10.6028/NIST.FIPS.204.ipd>

Future: Additional PQC Signature On-Ramp

- Primarily for new non-lattice signature schemes. (Lattices are ok if the candidate is better than Dilithium or Falcon in some important aspect.)
- First round on-ramp candidates were released on July 17, 2023.

6 Code-Based, 1 Isogeny-based, 7 Lattice-based, 7 MPC-in-the-Head, 10 multivariate, 4 Symmetric, 5 Other. Currently about 12 “broken/revised.”

<https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures>

- One scheme designed for side-channel security: **RACCOON** .
[del Pino, Espitau, Katsumata, Maller, Mouhartem, Prest, Rossi, Saarinen]

Defence & IC: CNSA 2.0 and NIAP

“Note that this will effectively deprecate [in NSS] the use of RSA, Diffie-Hellman (DH), and elliptic curve cryptography (ECDH and ECDSA) when mandated.”

Table III: CNSA 2.0 quantum-resistant public-key algorithms

Algorithm	Function	Specification	Parameters
CRYSTALS-Kyber	Asymmetric algorithm for key establishment	TBD	Use Level V parameters for all classification levels.
CRYSTALS-Dilithium	Asymmetric algorithm for digital signatures	TBD	Use Level V parameters for all classification levels.

(The NIAP talk will discuss their Protection Profiles in relation to PQC.)

Implication: Changing Secure Element Hardware

Generic Secure Element in -2020

Control Unit <i>ARM MCU</i>	Big Integer <i>RSA / ECC</i>
Simple DMA <i>basically copy</i>	SHA-2 <i>HMAC + Hash</i>
DRBG <i>SP 800-90A</i>	"TRNG" <i>AIS-20/31</i>
.. a lot more stuff ..	

Generic Secure Element in 2025-

Control Unit <i>RISC-V MCU</i>	Rings / NTT <i>Poly x^n+1</i>
Bit vector ops <i>A2B/B2A etc</i>	SHA3/SHAKE <i>Keccak f1600</i>
Fast Random <i>for masking</i>	ES + RBGs <i>SP 800-90ABC</i>
.. even more little stuff ..	

AVA_VAN: Testing with Actual Attacks

- **High assurance** level (EUCC: equivalent to AVA_VAN.3 or above) is a requirement for Root of Trust IP, Smart Cards, Secure elements, many types of IoT (SESIP).
 - While FIPS testing focuses on “checklist compliance”, AVA_VAN checks real-life security via a “penetration test.”
 - Possible FIPS 140-3 “non-invasive” (ISO 17825) leakage assessment ignores many practical physical attacks, such as fault injection attacks.
- “Evaluators must have knowledge and experience of [...] side channel attacks (SCA) such as Timing Analysis, Machine Learning based SCA, Simple Power Analysis (SPA), Differential Power Analysis (DPA), Differential EM radiation Analysis (DEMA), Template Attacks (TA); fault injection attacks such as DFA [...]”*
- **EUCC v1.1.1** (also SOG-IS Documents)

AVA_VAN: Common Criteria Vulnerability Analysis

Attack Potential is evaluated with a score-based system that roughly maps to the “**cost of attack.**” (think \$ or €)

Considers attack **Identification + exploitation**, with many factors:

- Elapsed time (hours-months)
- Attacker Expertise (multiple)
- Knowledge (how restricted)
- Access to the TOE (samples)
- Equipment (common/bespoke)

See EUCC v1.1.1 annexes or SOG-IS “Application of Attack Potential” docs.

AVA_VAN.1 Vulnerability Survey

- TOE resistance against BASIC Attack Potential (0-15)

AVA_VAN.2 (Unstructured) Vuln. Analysis

- TOE resistance against BASIC Attack Potential (16-20)

AVA_VAN.3 Focused (Unstructured) Vuln. Analysis

- TOE resistance against ENHANCED-BASIC AP (21-24)

AVA_VAN.4 Methodical Vuln. Analysis

- TOE resistance against MODERATE AP (25-30)

AVA_VAN.5 Advanced Methodical Vuln. Analysis

- TOE resistance against HIGH Attack Potential (31-)

AVA_VAN: Example Protection Profiles

AVA_VAN.3+ is a requirement for Root of Trust and Security IC products. We assume that this will not change with Post-Quantum Cryptography.

[JSADEN011] “**SESIP Profile for PSA Certified™ Level 3**”

Root of Trust (PSA-RoT): 35 person-days of AVA_VAN.3 activities.

[PP-0084] "**Security IC Platform Protection Profile**"

EAL 4 augmented by AVA_VAN.5 and ALC_DVS.2

[PP-0117] "**Secure Sub-System in System-on-Chip (3S in SoC)**"

EAL 4 augmented by ATE_DPT.2, AVA_VAN.5, ALC_DVS.2 and ALC_FLR.2

What are we protecting: CSPs and Verification

Classification in Crypto Module world:

- Public Security Parameter (**PSP**) needs integrity only: Can't be modified.
- Critical Security Parameter (**CSP**) needs confidentiality and integrity.
- Together: Sensitive Security Parameters (**SSP** \cong All variables in crypto!)

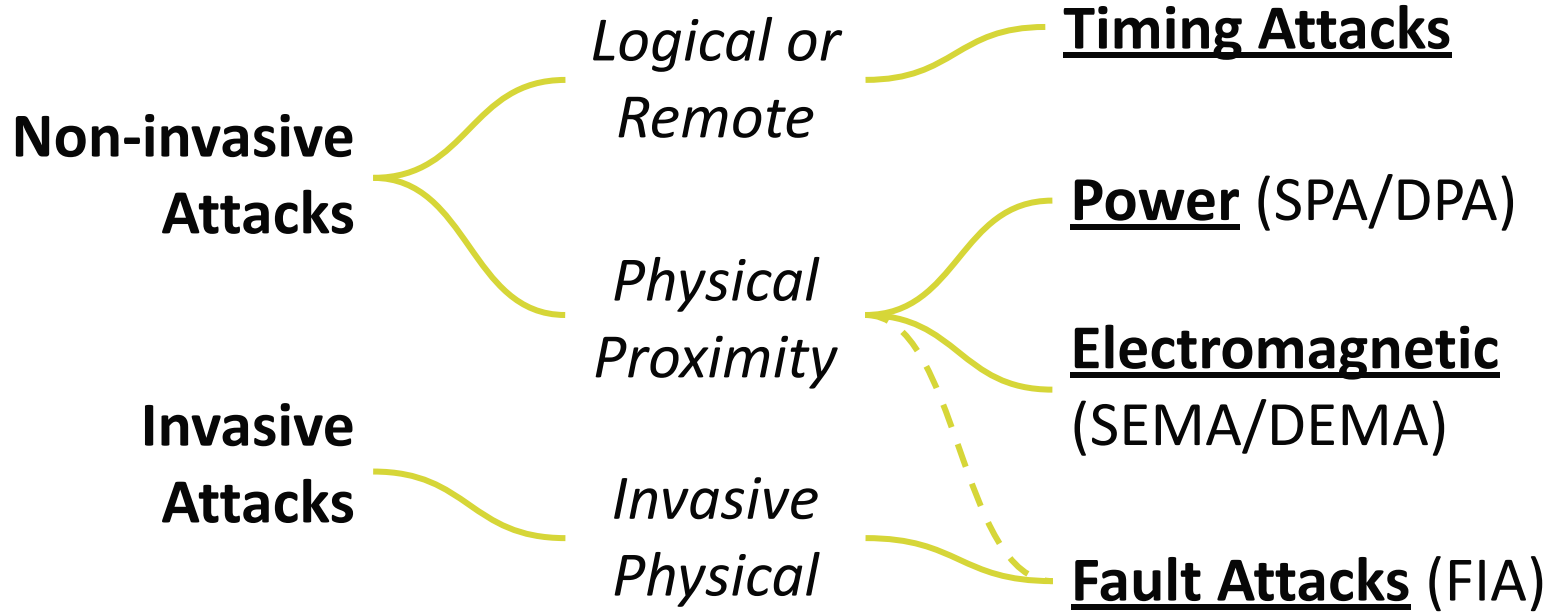
For us, a **CSP** is any information that helps directly or indirectly to:

1. Determine a shared secret in a key establishment scheme or
2. Forge a signature in a signature scheme.

Attacker who learns a secret (CSP**) variable or forges a signature “wins.”**

Attacker who bypasses signature verification / authentication “wins.”

Side-Channel Attacks



Remotely Exploitable: Timing Attacks 1996-2020s

- P.C. Kocher: "*Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems.*" (CRYPTO 1996. Target: RSAREF 2.0 running on MS-DOS.)
- D. Brumley and D. Boneh: "*Remote timing attacks are practical.*" (USENIX Security 2003. OpenSSL RSA remote key recovery, CVE-2003-0147.)
- B. Brumley and N. Toveri: "*Remote Timing Attacks Are Still Practical.*" (ESORICS 2011. OpenSSL ECDSA remote key recovery, CVE-2011-1945.)
- Q. Guo, T. Johansson, A. Nilsson, "*A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM.*" (Crypto 2020.)

Eliminate all **secret**-dependant latency

1. Secret-controlled branches and loops:

```
if <secret> then { delay1(); } else { delay2(); }
```

2. Memory accesses (cache timing attacks). Can be a load or a store.

```
ct = SBox[pt ^ key]; // cache latency leaks key.
```

3. Arithmetic operations whose processing time depends on inputs:

```
x = y % q; // division/remainder are rarely constant-time.
```

Rejection Samplers: Not literal “Constant-Time”

- You have a fair **6-sided dice** and want to have random numbers 1 to 5:
- Just ignore/reject sixes. The remaining **1..5 are uniform** in that range.
- Number of rolls (time) and even the pattern of rejects can be public.
- But it **is secure!** does not leak 1,2,3,4,5.
- The same Rejection Sampling idea extends to arbitrary distributions.

(Dilithium has R.S. in four places..)



Basics of Constant-time Programming

- Transform conditionals to straight-line code using Boolean operations?
 $s ? a : b$; vs. $b \wedge ((\neg(s \ \& \ 1)) \ \& \ (a \ \wedge \ b))$; 🤔
- Table lookups: Bit-slicing (transform to Boolean circuit), “scan / collect”.
- No division instructions in modular arithmetic (Montgomery, Barrett)
- Have a “library” of solid CT replacements for memcmp() and similar
- Test with symbolic execution (e.g. valgrind) or at instruction level..

.. etc .. these are core cryptography programming skills!

Portability of “constant-time” code ?

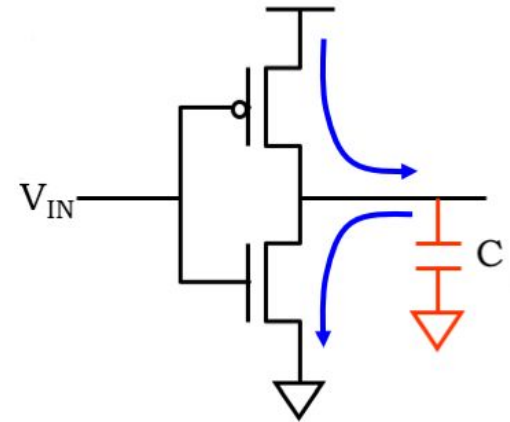
- Use static analysis or dynamic variable tainting (in emulator) to verify that compiled code is using only the right instructions to handle secrets.
- ”Constant-timeness” of **Intel** and **ARM**: Often from experiments.



- **RISC-V** codified data-independent latency as the **Zkt** instruction list for scalar, **Zvkt** for vector. <https://github.com/riscv/riscv-crypto/releases>
- RISC-V **PQC ISA** Starting: <https://lists.riscv.org/g/tech-pqc-cryptography>

SCA, DEMA: Sources of Power and EM leakage

- Logic changes ($0 \rightarrow 1$ or $1 \rightarrow 0$) consume **dynamic power**.
(There is also static power, but resting bits generally don't leak.)
- State changes also emanate on the **electromagnetic spectrum**.
- Programmer-visible CPU registers and memory are just one part of the circuitry where your secret bits pass through.
- Need to consider all data paths whose toggling can correlate with secret data.



RSA and ECC: “Algebraic countermeasures”

Well-established techniques:

Basically 1 step – ModExp (RSA) or Scalar Mult (ECC) – to protect

RSA: Randomization / “blinding” (D. Chaum 1982, P. Kocher 1996+)

- Message: Pick random r , compute blinded $c' = cr^e \pmod n$, decrypt/sign c' instead of c : $m' = c'^d \pmod n$, normalize by $m = m'r^{-1}$.
- Exponent: use $d' = (p-1)(q-1)r + d$ instead of d to randomize exponent.

ECC: J.-S. Coron, “Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems.” Proc. CHES’99, pp. 292–302, 1999

- For 20+ years: Randomization of the Private Exponent [Scalar], Blinding the [Base] Point P, Randomized Projective Coordinates, + misc.

PQC: Need masking (+ shuffling, delays..)

- Masking splits secrets \mathbf{X} into \mathbf{d} randomized “shares” $\mathbf{X}_1 + \mathbf{X}_2 + \dots + \mathbf{X}_d = \mathbf{X}$. Disclosure of a subset of $\mathbf{d}-1$ shares does not allow one to determine \mathbf{X} .

Arith. $\mathbf{X} = \mathbf{X}_1 + \mathbf{X}_2 \pmod{q}$; q is 3329 (Kyber) or 8380417 (Dilithium).
 Boolean $\mathbf{X} = \mathbf{X}_1 \oplus \mathbf{X}_2$ Exclusive or: SHA3/SHAKE, shifts, etc.

- Masking and other attack mitigation techniques needed for PQC algorithms are technically much more complex than for legacy crypto.
- Why? The algorithms are not homogenous like RSA or ECC but contain a large number of dissimilar steps. Secrets may leak out at any stage. One may have to design dozen different masking gadgets for one algorithm.

[Identify CSPs] ML-DSA.Sign(sk, M) part 1/3

- 1: $(\rho, K, \text{tr}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(\mathbf{sk})$ // Not all variables are actually secret!
 - 2: $\hat{\mathbf{s}}_1 \leftarrow \text{NTT}(\mathbf{s}_1)$ // The main secret key is $(\mathbf{s}_1, \mathbf{s}_2)$.
 - 3: $\hat{\mathbf{s}}_2 \leftarrow \text{NTT}(\mathbf{s}_2)$
 - 4: $\hat{\mathbf{t}}_0 \leftarrow \text{NTT}(\mathbf{t}_0)$ // The \mathbf{t}_0 and ρ variables = the public key.
 - 5: $\hat{\mathbf{A}} \leftarrow \text{ExpandA}(\rho)$ // Public “lattice” matrix $\hat{\mathbf{A}}$.
 - 6: $\mu \leftarrow \text{H}(\text{tr} \parallel M, 512)$ // Message M not secret (public verify).
 - 7: $\text{rnd} \leftarrow \{0, 1\}^{256}$ // Secret randomizer (recommended).
 - 8: $\rho' \leftarrow \text{H}(K \parallel \text{rnd} \parallel \mu, 512)$ // Ephemeral random signature seed.
 - 9: $K \leftarrow 0$ // Public counter (timing leak okay.)
 - 10: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$ // Signature result: public.
- $\hat{\mathbf{s}}_1, \hat{\mathbf{s}}_2$: Secret Key. ρ' : Secret seed. // Masked variables after steps 1-10.

[Identify CSPs] ML-DSA.Sign(sk, M) part 2/3

```

11:  while (z, h) =  $\perp$  do // Signature generation trial.
12:  |   y            $\leftarrow$  ExpandMask( $\rho, \mathbf{k}$ ) // Ephemeral secret.
13:  |   w            $\leftarrow$  NTT-1( $\hat{A} \circ$  NTT(y)) // Still sensitive.
14:  |   w1         $\leftarrow$  HighBits(w) // Sensitive gadget (with A2B.)
15:  |    $\tilde{c} \in \{0, 1\}^{2\lambda} \leftarrow$  H( $\mu \parallel$  w1Encode(w1), 2 $\lambda$ ) // Commitment hash: safe.
16:  |   ( $\tilde{c}_1, \tilde{c}_2$ )  $\in \{0, 1\}^{256} \times \{0, 1\}^{2\lambda-256} \leftarrow$   $\tilde{c}$  // First 256 bits to  $\tilde{c}_1$ .
17:  |   c            $\leftarrow$  SampleInBall( $\tilde{c}_1$ ) // Sample "ternary."
18:  |    $\hat{c}$           $\leftarrow$  NTT(c) // NTT Domain.
19:  |    $\langle\langle cs_1 \rangle\rangle$   $\leftarrow$  NTT-1( $\hat{c} \circ \hat{s}_1$ ) // Reversible op; sensitive.
20:  |    $\langle\langle cs_2 \rangle\rangle$   $\leftarrow$  NTT-1( $\hat{c} \circ \hat{s}_2$ )
21:  |   z            $\leftarrow$  y +  $\langle\langle cs_1 \rangle\rangle$  // Signature: "Semi-public."
22:  |   r0         $\leftarrow$  LowBits(w -  $\langle\langle cs_2 \rangle\rangle$ ) // MLWE protected.

```

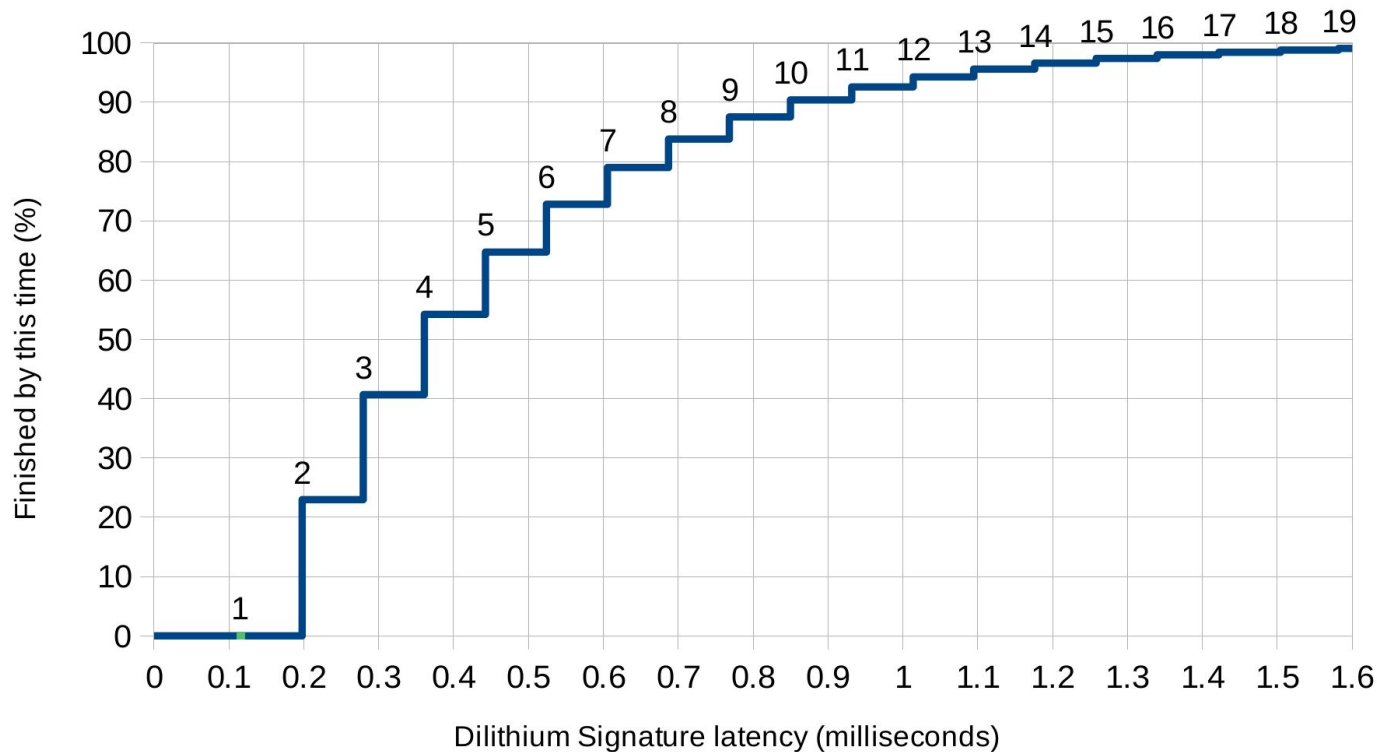
[Identify CSPs] ML-DSA.Sign(sk, M) part 3/3

```

23: |   if  $\|z\|_\infty \geq \gamma_1 - \beta$  or  $\|r_0\|_\infty \geq \gamma_2 - \beta$  then  $(z, h) \leftarrow \perp$            // Norm checks.
24: |   else
25: |   |    $\langle\langle ct_0 \rangle\rangle \leftarrow NTT^{-1}(\hat{c} \circ \hat{t}_0)$ 
26: |   |    $h \leftarrow \text{MakeHint}(-\langle\langle ct_0 \rangle\rangle, w - \langle\langle cs_2 \rangle\rangle + \langle\langle ct_0 \rangle\rangle)$  // Compute hint.
27: |   |   if  $\|\langle\langle ct_0 \rangle\rangle\|_\infty \geq \gamma_2$  or the number of 1's in  $h$  is greater than  $\omega$ ,
   |   |   |   then  $(z, h) \leftarrow \perp$  // Reject.
28: |   |   end if
29: |   end if
30: |    $K \leftarrow K + \text{ell}$  // Increment counter.
31: end while // Success!
32:  $\sigma \leftarrow \text{sigEncode}(\tilde{c}, z \bmod \pm q, h)$  // signature is public
33: return  $\sigma$ 

```

Sidenote: Variable signing time



How to address SCA security in Design?

- 1. Design secure gadgets.** The gadgets for all “CSP” algorithm steps should be provably secure in appropriate model: t-probing model, noisy leakage model, SNI (Strong Non-Interference) composability.
- 2. Leakage simulation** in microcontroller and Pre-Silicon for hardware. The models range from very simple & fast – based on bit toggling – to extremely advanced “3D” physical models.
- 3. Physical verification / Sign-Off.** Continuing adversarial analysis using physical tools. Also running automated leakage assessment tests like TVLA in Continuous Integration.

Automation: Continuous Integration on FPGAs



Automation: Test Vector Leakage Assessments

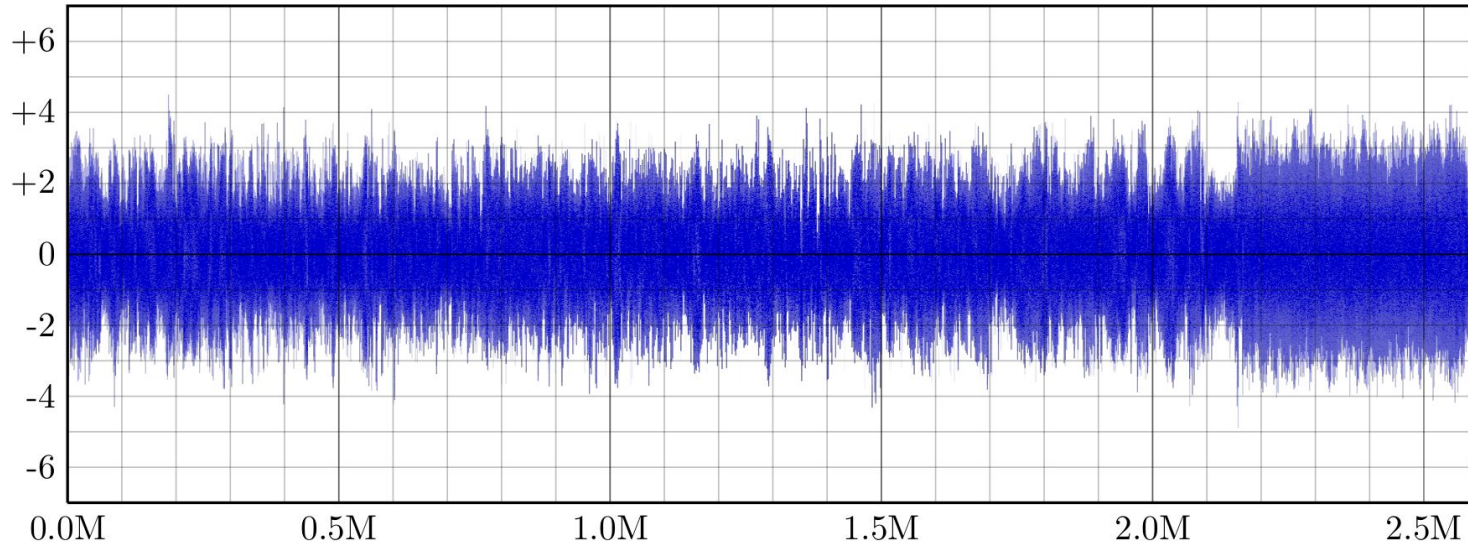


Figure 1: No secret key leakage was detected in a 200,000-trace leakage assessment of Raccoon-128 ($d = 2$) signature function. Each trace had 2.59×10^6 time points (one for each cycle – horizontal axis); a total of 518 billion measurements. The vertical axis has the t -statistic, which was bound by $|t| < 4.89$. This is well under the critical value $C = 6.94$.

Dilithium: On SCA and FIA Countermeasures

- **Constant time code** and **randomized signing** are absolutely needed. Randomization helps with security; ML-DSA IPD allows both options. (See <https://ia.cr/2022/1406> - along with additional CSP analysis.)
- **DPA & DEMA:** Dilithium requires complex countermeasures due to a large number of dissimilar algorithm steps (~12 different masking gadgets.)
- **Fault Attacks;** countermeasures often focused on fault **detection**. Error, glitch detectors, “check & recheck” everywhere, especially in verification.
- **Key management** (and KeyGen) also needs security: See WrapQ techniques: <https://ia.cr/2022/1499>

Attack Potential: Mock Calculation (Example)

<u>AP Component</u>	<u>Identification</u>	<u>Exploitation</u>
Elapsed time	2 (< one week)	6 (< one month)
Expertise	5 (expert)	4 (expert)
Knowledge of the TOE	4 (sensitive)	0 (public)
Access to the TOE	0 (< 10 samples)	0 (< 10 samples)
Equipment	3 (specialized)	4 (specialized)
Open Samples	0 (public)	0 (public)
Total	28 (AVA_VAN.4 / moderate AP range)	

SOG-IS: “Application of Attack Potential to Smartcards and Similar Devices”

Conclusion: Countermeasure Design Loop

Side-channel security development is an iterative process.

1. Identify all critical variables and computations / points of attack.
2. Design consistent countermeasures for each (at least masking).
3. Validate countermeasures. Apply adversarial analysis, testing.
4. Based on analysis and new research: Improve, loop to Step 3.

State of the Art: We and a few others are in the continuous iteration phase 3-4 with **Kyber** and **Dilithium**. (AVA_VAN.3+ certification is possible.)

Note: Thus far no one (to our knowledge) has completed Steps 2-4 for **Falcon**. Available countermeasures may remain to be limited to timing attacks.

A large, circular visualization of a particle detector, likely a bubble chamber or cloud chamber, showing a complex network of tracks and vertices. The tracks are composed of numerous small, glowing orange-yellow dots, creating a dense, intricate pattern against a dark background. The overall shape is roughly circular, with some tracks extending towards the edges.

Thank You
Questions?