Post-Quantum Crypto Side-Channel Tests (and a CSP Walk-Through)

Think openly, build securely

International Cryptographic Module Conference
September 15, 2022 - Washington DC

Dr. Markku-Juhani O. Saarinen
Staff Cryptography Architect, PQShield Ltd.

Outline

Post-Quantum Crypto Security Engineering & Validation

Note: *This is a large slideset; I will be be skimming over some parts.*

1. **Motivation: NIST PQC and "Non-Invasive" in FIPS 140-3.**
2. CRYSTALS-Kyber: Key Establishment.
3. CRYSTALS-Dilithium: Signatures.
4. ISO 17825 / "FIPS 140-3" TVLA in Side-Channel Testing of PQC.

Motivation: NIST PQC Means FIPS 140-3 PQC

Post-Quantum Crypto transition is driven by NIST/FIPS

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

NIST/FIPS Post-Quantum Crypto: Selection July 2022, Standards 2024.

Replacement for ECC, RSA key establishment and ECDSA, RSA signatures.

Especially for U.S. Government Entities:

- Active transition effort expected (presidential directives NSM-08, NSM-10).
- Regulations mandate FIPS 140-3 cryptography -> also for PQC modules.

Motivation: (Sept 2022) CNSA 2.0 / NIAP

Public-key
CRYSTALS-Dilithium
CRYSTALS-Kyber

Symmetric-key
Advanced Encryption Standard (AES)
Secure Hash Algorithm (SHA)

Software and Firmware Updates
Xtended Merkle Signature Scheme (XMSS)
Leighton-Micali Signature (LMS)

Transition 2025-2030-2035:

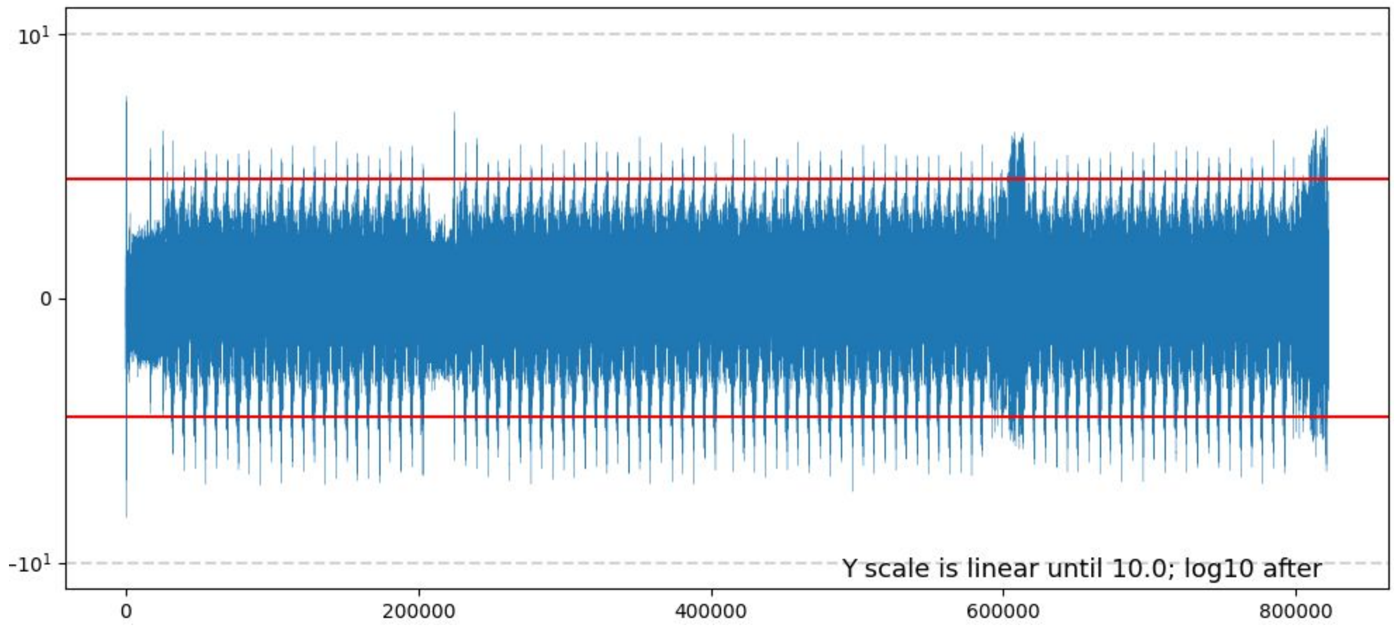
“Note that this will effectively deprecate [in NSS] the use of RSA, Diffie-Hellman (DH), and elliptic curve cryptography (ECDH and ECDSA) when mandated.”

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.

Motivation: FIPS 140-3 Non-Invasive Security

Requires Side-Channel Mitigations at High Security Levels



Introduced as a major change in FIPS 140-3 in relation to FIPS 140-2:

- Side-Channel Attacks (Power, Emissions, Timing) are in 140-3 scope.
- Documentation required for Levels 1 and 2. Mitigation Testing at Levels 3 and 4.

What are the CSPs, PSPs, SSPs of PQCs !

Designer classifies all variables and wires into CSPs and PSPs

Critical Security Parameter (CSP) requires require both confidentiality (secrecy) and integrity (no modification) protection.

Examples: Secret and private keys, passwords, temporary tokens, and derived temporary quantities whose disclosure would compromise the security of the cryptographic system.

Public Security Parameters (PSPs) do not need confidentiality but need Integrity.

Examples: A public key needs to be handled in a way that prevents it from being changed or replaced. A digital signature or ciphertext is usually a PSP, not a CSP. Any component variable of a secret key that can be easily derived from the public key is a PSP.

**Sensitive Security Parameters (SSPs): ** Together, CSPs and PSPs constitute SSPs.

Examples: Most inputs and outputs of a cryptographic module are SSPs. A public-private key pair is an SSP. FIPS Zeroization requirements apply to all SSPs, including PSPs.)

What needs to be protected?

Only CSPs are in Scope of non-invasive (and need masking)

Section 7.8 of ISO/IEC 19790:2012(E), unmodified in ISO/IEC WD 19790:2022(E):

``Non-invasive attacks attempt to compromise a cryptographic module by acquiring knowledge of the module's CSPs without physically modifying or invading the module. Modules may implement various techniques to mitigate against these types of attacks.''

- Only leakage of CSPs is relevant for FIPS 140-3. Public key leakage is a *false positive*.
- For us, this CSP is primarily information that (1) can be used to determine a shared secret in a key establishment scheme or (2) forge a signature in a signature scheme.
- Invasive physical attacks (that modify the state) are out of scope for ISO 17825. FIPS 140-3 has "fault induction mitigation" at Level 4. Faults are a part of CC assessments.

Outline

Post-Quantum Crypto Security Engineering & Validation

***Note:** This is a large slideset; I will be be skimming over some parts.*

1. Motivation: NIST PQC and "Non-Invasive" in FIPS 140-3.
2. **CRYSTALS-Kyber: Key Establishment.**
3. CRYSTALS-Dilithium: Signatures.
4. ISO 17825 / "FIPS 140-3" TVLA in Side-Channel Testing of PQC.

CRYSTALS-Kyber

NIST's Preferred PQC Key-Establishment Scheme

"The public-key encryption and key-establishment algorithm that will be standardized is ***CRYSTALS–KYBER****."* – NIST IR 8413, July 2022

- Designed by a team of academic, industry researchers (mainly in Europe). Building on ~25 years of research.

Kyber is a Quantum-secure counterpart to NIST SP 800-56A/B, *Pairwise Key-Establishment.* Bulk data encryption is with symmetric crypto (AES / AEADs).

- Establishing shared session keys in security protocols (TLS, IPSec, ..).

- Establishing confidentiality and integrity keys for secure messaging / e-mail.

Kyber: Basic Facts

A drop-in replacement to ECDH, RSA Encryption.. Almost

- A Key Establishment Mechanism (KEM). Can be used to replace Key Exchange ([Elliptic Curve] Diffie-Hellman), Public-Key Encryption (RSA).

- Significantly faster (than ECDH / NIST-P256, P384) on common CPUs.

- Sizes below are in bytes. ~3x larger than RSA, 25x larger than ECDH.

Parameters	**PQ Cat**	**Ciphertext**	**Public Key**	**Secret Key**
Kyber-512	**1 (128)**	768	800	(1632)
Kyber-768	**3 (192)**	1088	1184	(2400)
Kyber-1024	**5 (256)**	1568	1568	(3168)

Kyber Encapsulation ("Alice")

Encapsulation wraps Encryption inside a *a lot of* hashing

(**CT**, **SS**) = **Kyber.CCAKEM.Enc**(**PK**):

1.	**seed**	←	random 32 bytes	// *Seed: Unique every time.*
2.	**M**	←	SHA3-256(**seed**)	// *Hash it, "just to be sure.."*
3.	(**K**, **R**)	←	SHA3-512(**M** ‖ SHA3-256(PK))	// *Shared secret K and seed R.*
4.	CT	←	Kyber.CPAPKE.Enc(PK, **M**, **R**)	// *Encrypt to create ciphertext.*
5.	**SS**	←	SHAKE-256(**K** ‖ SHA3-256(CT))	// *Shared Secret (256 bits).*

- CSP variables are marked in **RED**. Ciphertext CT is public, session key **SS** secret, etc..

The wrapper is known as the "Fujisaki-Okamoto Transform." It is essential to protect against Chosen Ciphertext Attacks (CCA) if the secret key is fixed (not ephemeral).

Kyber Decapsulation ("Bob")

Decapsulation wraps & tests Decryption. Pretends to never fail!

SS = **<u>Kyber.CCAKEM.Dec</u>(CT, SK**):

1-4.	(**S**, PK, h, **Z**) $\leftarrow$ **SK**	//	*Decode secret key.*
4.	**M'** $\leftarrow$ <u>Kyber.CPAPKE.Dec</u>(**S**, CT)	//	*Encrypt to create ciphertext.*
5.	(**K'**,**R'**) $\leftarrow$ SHA3-512(**M'** ‖ SHA3-256(PK))	//	*Hash of PK is cached in "h."*
6.	**CT'** $\leftarrow$ <u>Kyber.CPAPKE.Enc</u>(PK, **M'**, **R'**)	//	*Simulated encryption.*
7.	If CT ≠ **CT'** then:	//	*If re-encryption* different,
10.	| **K'** $\leftarrow$ **Z**	//	*.. replace key with a "fake."*
12.	**SS'** $\leftarrow$ SHAKE-256(**K'** ‖ SHA3-256(CT))	//	*Shared Secret.*

If CT is valid, one can get **SS'** without steps 6-10 – and perhaps make the decapsulation twice as fast – but this won't be secure against (adaptive) CCA attacks. **Known Attacks!**

Kyber: TLS 1.3 Integration

New IETF Internet Drafts Underway

Initial support in cloud, browsers, handsets will probably be a Kyber + ECDH hybrid key exchange in TLS 1.3 – to deter "record now, decrypt later" attacks.

- P. Schwabe, B. Westerbaan: **"Kyber Post-Quantum KEM."**
  https://datatracker.ietf.org/doc/html/draft-cfrg-schwabe-kyber

- D. Stebila, S. Fluhrer, S. Gueron: **"Hybrid key exchange in TLS 1.3."**
  https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design

Currently proposes four hybrids for TLS 1.3 key exchange: *x25519+kyber768, secp384r1 + kyber768, x25519 + kyber512, secp256r1 + kyber512.*

[DEEP] **Kyber Algorithm Parameters**

A Module-LWE Based KEM - Popular Successor to NewHope

- Coefficients / elements are mod prime $q = 3329 = 2^8 \cdot 13 + 1$, fitting 12 bits.
- Structured lattice polynomial rings are $R = \mathbb{Z}_N [x]/(x^n + 1)$ with degree $\mathbf{n}=256$.
- Polynomial multiplication is via (negacyclic) Number Theoretic Transforms (NTT).
- Also module. Rank is denoted by **k** in Kyber. Lattice dimension is $\mathbf{k} \times \mathbf{n}$.
- In Learning With Errors (LWE) the "error" that that makes the inverse problem hard is explicitly added from a distribution. Bit shifting parameters d_u, d_v are for compression.
- Uses both uniform distribution and the Centered Binomial Distribution (CBD) with parameter $\eta_1, \eta_2 \in \{ 2, 3 \}$. The numbers are $-\eta \leq x \leq +\eta$ from a pop count of 2η bits.

Parameter Set	Rank	η_1	η_2	d_u	d_v	Failure	Classic	Quantum
Cat 1: *"Kyber512"*	k=2	3	2	10	4	2^{-139}	2^{118}	2^{107}
Cat 3: *"Kyber768"*	k=3	2	2	10	4	2^{-164}	2^{183}	2^{166}
Cat 5: *"Kyber1024"*	k=4	2	2	11	5	2^{-174}	2^{256}	2^{232}

[DEEP] Kyber's Polynomial Ring

Summary: Your big integer unit is useless. SIMD rules.

- Kyber uses **k** $\in \{ 2, 3, 4 \}$ rings $R_q = \mathbb{Z}_q [x]/(x^n + 1)$ with degree **n=256** and **q=3329**.

- Multiplication is implemented via Number-Theoretic Transform (NTT), which is analogous to FFT but uses n:th roots of unity $\zeta^n = 1$ ("zetas") in the finite field GF(q) a.k.a. $\mathbb{Z}_q$ instead of FFT's $\omega^n = 1$ in complex field $\mathbb{C}$. Generator used is $\zeta^1=17$.

- Kyber uses NTT-domain values "on the wire", so representation must be the same!

- As with FFT, $A*B = NTT^{-1}(NTT(A) \circ NTT(B))$. NTT multiplication is $O(n \log_2 n)$, traditional is $O(n^2)$. In Kyber the "$\circ$ basecase" op is with pairs of coefficients, not point-by-point.

- Normal FFT/NTT would allow wrap-around convolution (mod $x^n - 1$), but to do $x^n + 1$ with length-n NTT we need Nussbaumer's negacyclic transform a.k.a. "tweaks".

[DEEP] **CPA Kyber Keypair Generation**

Algorithm “KYBER.CPAPKE.KeyGen”

1. $\rho, \sigma \leftarrow$ 256 random bits // *Public and secret seed values.*
4. $\hat{A} \leftarrow \text{gen}(\rho)$ // *Public value $\hat{A}$ is derived from ρ.*

“$\hat{A}$” is a $k \times k$ matrix of vectors/polynomials. Computed on the fly from ρ and (i, j) as inputs SHAKE-128. XOF output filtered with rejection sampling to be uniform mod q. As it is uniform, it can be directly interpreted to be uniform in the NTT domain too.

9. $s \leftarrow \text{CBD}(\eta_1, \sigma, 0,1,..k\text{-}1)$ // *Weights of $2 \times \eta_1$ from SHAKE-256 output.*
13. $e \leftarrow \text{CBD}(\eta_1, \sigma, k,..,2k\text{-}1)$ // *Error the same, last parameter is a counter.*

*Both s and e consist of k polynomials, each of k*n coefficients in $-\eta_1 \leq x \leq +\eta_1$.*

17. $\hat{s} \leftarrow \text{NTT}(s)$, $\hat{e} \leftarrow \text{NTT}(e)$ // *Transform both the secret key and error.*
19. $\hat{t} \leftarrow \hat{A} \circ \hat{s} + \hat{e}$ // *Public key $\hat{t} = \text{NTT}(A \cdot s + e)$ – in NTT domain.*
20. return $PK = (\hat{t}, \rho)$, $S = \hat{s}$

[DEEP] **Kyber's "Compression"**

A little bit cumbersome bit-dropping optimization

The serialization methods mostly involve bit field packing (ignoring those for now)

Kyber also does lossy scaling to 1 ("message") and d_u, d_v bits: $d \in \{ 1, 4, 5, 10, 11 \}$.

Compress$_q$: Scales a number from mod-q range $[0, q-1]$ to d-bit range $[0, 2^d-1]$.

$$\mathrm{Compress}_q(x, d) = \lceil (2^d / q) \cdot x \rfloor \bmod 2^d.$$

Decompress$_q$: Scales a number from d-bit range $[0, 2^d-1]$ to mod-q range $[0, q-1]$.

$$\mathrm{Decompress}_q(x, d) = \lceil (q / 2^d) \cdot x \rfloor.$$

<u>Note</u>: $\lceil x \rfloor = \mathrm{floor}(x + ½)$ is rounding to closest integer, with ties rounded up.

[DEEP] **Kyber Encryption (CPA)**

A subroutine for both Encapsulation and Decapsulation

CT = <u>Kyber.CPAPKE.Enc</u>(PK, M, R):

1.	$(\hat{t}, \rho) \leftarrow$ PK	//	*Deserialize $\hat{t}$ and ρ from the public key.*
4.	$\hat{A} \leftarrow \mathrm{gen}(\rho)$	//	*(Actually compute $\hat{A}$ on the fly from seed ρ.)*
9.	$r \leftarrow \mathrm{CBD}(\eta_1, R, 0,1,..k\text{-}1)$	//	*Weights of $2{\times}\eta_1$ segments of SHAKE-256 output.*
13.	$e_1 \leftarrow \mathrm{CBD}(\eta_2, R, k,..,2k\text{-}1)$	//	*Error 1 is the same, but uses distribution η_2.*
17.	$e_2 \leftarrow \mathrm{CBD}(\eta_2, R, 2k)$	//	*Error 2 is a single (n=256) ring element.*
18.	$\hat{r} \leftarrow \mathrm{NTT}(r)$	//	*Transform ephemeral secret.*
19.	$u \leftarrow \mathrm{NTT}^{-1}(\hat{A}^T \circ \hat{r}) + e_1$	//	*First part of ciphertext: $u = A^T \cdot r + e_1$.*
20.	$m \leftarrow \mathrm{Decompress}_q(M, 1)$	//	*"One time pad" bits as { 0, ceil(q/2) }.*
	$v \leftarrow \mathrm{NTT}^{-1}(\hat{t}^T \circ \hat{r}) + e_2 + m$	//	*Second, shorter part of ciphertext: $t^T \cdot r + e_2 + m$.*
21.	return **CT** = $(\mathrm{Compress}_q(u, d_u), \mathrm{Compress}_q(v, d_v))$		

[DEEP] **Kyber Decryption (CPA)**

CPA Decryption is just a subroutine for CCA Decapsulation

M = **<u>Kyber.CPAPKE.Dec</u>(CT, s)**:

0. $(CT_u, CT_u) \leftarrow CT$ // *Split the ciphertext into u and v halves.*
1. $u \leftarrow \mathrm{Decompress}_q(CT_u, d_u)$ // *Scale coefficients of u from d_u bits to [0, q-1].*
2. $v \leftarrow \mathrm{Decompress}_q(CT_v, d_v)$ // *Scale coefficients of v from d_v bits to [0, q-1].*
3. $s \leftarrow S$ // *Load (and remask) the secret key.*
4. $m \leftarrow v - \mathrm{NTT}^{-1}(\hat{s}^T \circ \mathrm{NTT}(u))$ // *NTT arithmetic for: $m = v - s^T \cdot u$.*
5. return $M = \mathrm{Compress}_q(m, 1)$ // *Extract message $M \in \{0,1\}^{256}$ from high part.*

Why does it work? 🤔 Let's substitute and the expand equation (ignoring transpositions):

<u>Public key</u>: $PK = t = A \cdot s + e$. <u>Ciphertext</u>: $CT=(u, v)$, $u = A \cdot r + e_1$, $v = t \cdot r + e_2 + m$.

<u>Decryption</u>: $v - s \cdot u = (A \cdot s \cdot r) - (A \cdot s \cdot r) + (e \cdot r) + e_1 + e_2 + m \approx m$ + "small" values!

Outline

Post-Quantum Crypto Security Engineering & Validation

Note: *This is a large slideset; I will be be skimming over some parts.*

1. Motivation: NIST PQC and “Non-Invasive” in FIPS 140-3.
2. CRYSTALS-Kyber: Key Establishment.
3. **CRYSTALS-Dilithium: Signatures.**
4. ISO 17825 / “FIPS 140-3” TVLA in Side-Channel Testing of PQC.

CRYSTALS-Dilithium

NIST's Preferred PQC Signature Scheme

*"While there are multiple signature algorithms selected, NIST recommends **CRYSTALS-Dilithium** as the primary algorithm to be implemented."*

– NIST IR 8413, July 2022

- From academics + industry, building on about 25 years of research.
- Quantum-secure counterpart for FIPS 186-4, Digital Signature Standard (RSA, ECDSA). High-security variants expected to be chosen for NSS.
- Main use cases: PKI certificates / message authentication and end-point authentication (TLS, IPSec). *IETF specs and interoperability testing needed.*

Dilithium: Basic Facts

A drop-in replacement to ECDSA, RSA Signatures.. Almost

- Signs message M with a “hash prefix”: $\mathbf{H_2(H_1(PK) \,||\, M)}$.
- Generally faster than ECDSA (NIST-P256, P384) on common CPUs.
- Sizes below are in bytes: Signatures are about 10x larger than RSA.

PQ Security	**Signature**	**Public Key**	**(Secret Key)**
Dilithium2	2420	1312	(2528)
Dilithium3	3293	1952	(4000)
Dilithium5	4595	2592	(4964)

Dilithium Versions (pre-standard danger!)

We're talking about Version 3.1. Three Security Levels 2, 3, 5.

- Parameter sets **Dilithium2**, **Dilithium3**, and **Dilithium5** correspond to security levels of SHA-256, AES-192, and AES-256 against quantum adversaries (no "1"!).

- A potential security issue in the the third-round submission (3.0) was noted by NIST and fixed by the Dilithium team for version 3.1 in February 2021. This change impacts the key sizes slightly and breaks (KAT) compatibility.

- As of 2022-Sep-01, the version on NIST web site does **not** have this fix.

Get the latest spec: https://pq-crystals.org/dilithium/resources.shtml
Reference code / KATs: https://github.com/pq-crystals/dilithium

Dilithium Signatures & Real Time Systems

Bernoulli trials: Technically no upper bound

- The norms checks check that the problem is "hard enough", that there is no accidental leakage, and that the signature fits into the fixed-length format.

- Each one of the iterations is an independent "Bernoulli trial" (of a random **y**) with probability p of passing; passing in n iterations (or less) is $1 - (1-p)^n$.

- There is technically no upper bound and the signature time is not Gaussian.

Rep	**1**	**2**	**3**	**4**	**5**	**10**	**15**
Dilithium 2	23.4%	41.3%	55.0%	65.6%	73.6%	93.0%	98.2%
Dilithium 3	19.9%	35.9%	48.6%	58.9%	67.0%	89.1%	96.4%
Dilithium 4	26.4%	45.8%	60.0%	70.6%	78.3%	95.3%	99.0%

Dilithium: Upper-bounding Signature Time

Determine two implementation characteristics C1 and C2

Both the set-up time **C1** and per-iteration latency **C2** have relatively low variance.

Latency: $\mathbf{t = C1 + n * C2}$.

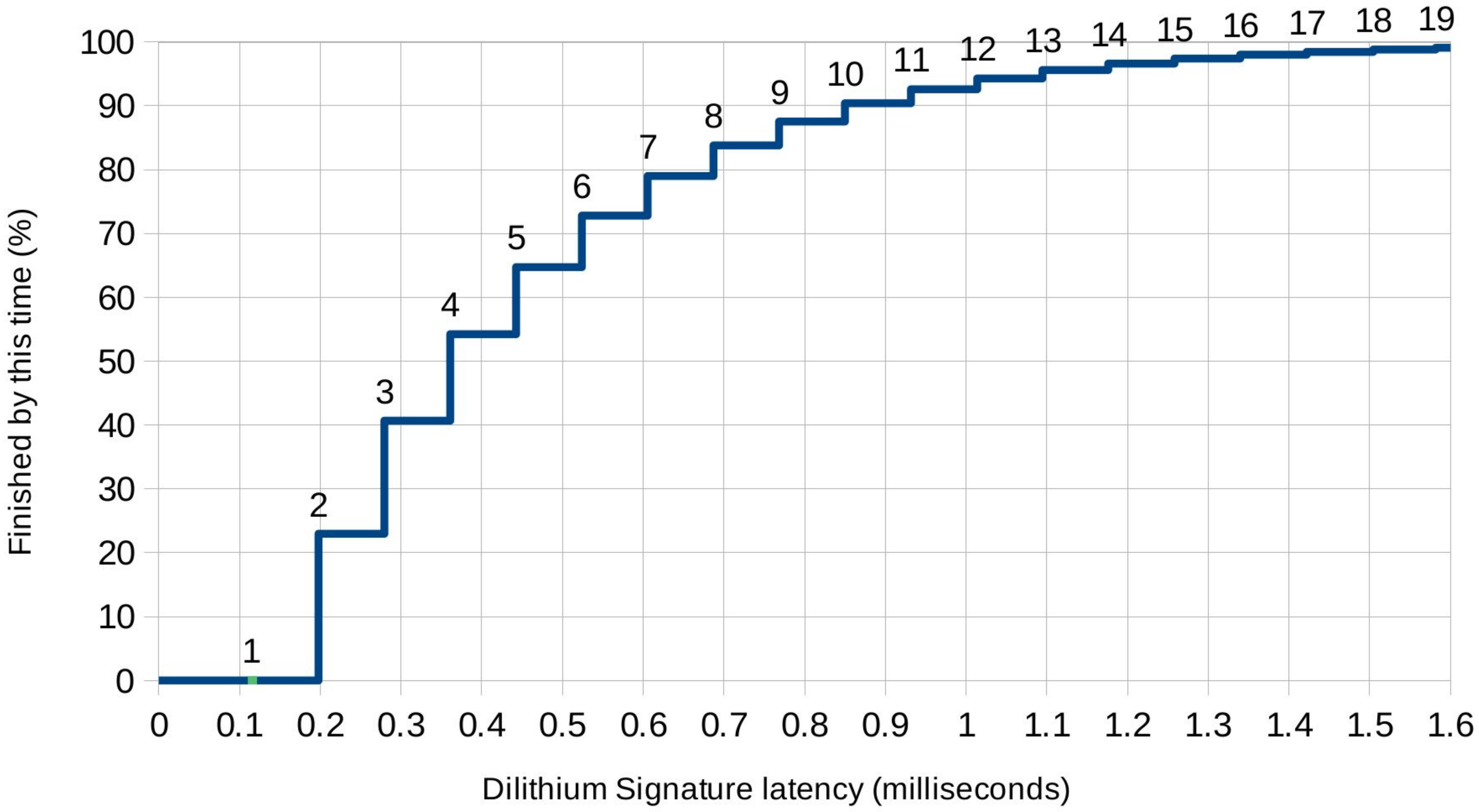
The per-iteration success probability **p** depends the Dilithium algorithm itself:

$\mathbf{p \in \{ 0.23, 0.20, 0.26 \}}$ at security levels 2, 3, and 5, respectively.

The probability of success before $t < C1+C2$ is zero (you need at least 1 iteration.)

Iterations at time **t**: $\mathbf{n = \lfloor (t - c1) / c2 \rfloor}$. Success rate after n: $\mathbf{1 - (1 - p)^n}$.

Dilithium2 Latency vs Success % (example)



Dilithium: PKI Integration

Work going on in IETF LAMPS WG and some other places

J. Massimo, P. Kampanakis, S. Turner, B. Westerbaan. “**Algorithms and Identifiers for Post-Quantum Algorithms in the Internet X.509 Public Key Infrastructure.**”
https://datatracker.ietf.org/doc/draft-massimo-lamps-pq-sig-certificates/

One may want to transition via a hybrid solution. There are two main hybridization proposals, offering different trade-offs in system integration complexity:

- Composite: Combine a classical (ECDSA) and PQC signature of the same data into a single hybrid signature. Both signatures need to check as valid.

- Non-composite (NSA): Effectively two independent certificate chains, PKIs.

[DEEP] **Dilithium Algorithm Parameters**

A Signature Algorithm based on MLWE and SIS

- Coefficients / elements work in $\mathbb{Z}_q$ with q = 8380417 = $2^{23} - 2^{13} + 1$ fitting a 23 bits.
- Ring again is of type $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ with n=256. NTT arithmetic is used.
- **A** has two dimensions: **k** and **l**, so the total dimension is **k** × **l** × **n.**
- Public key compression (bit dropping): d = 13 bits.
- Challenge distribution has τ non-zero ±1 coefficients and (n-τ) zero coefficients.
- The secret key distribution is *uniform* but in very short range [-η, +η].
- Uniform **y** sampling range $[-\gamma_1, +\gamma_1]$ and low-order rounding range is $[-\gamma_2, +\gamma_2]$.
- Furthermore we have rejection bounds β (for signature) and ω (for carry hint **h**).

Parameter Set	(k, l)	τ	η	γ_1	$(q-1)/\gamma_2$	β	ω	Reps	Classic	Quant
Dilithium 2:	(4, 4)	39	2	2^{17}	88	78	88	4.3	2^{123}	2^{112}
Dilithium 3:	(6, 5)	49	4	2^{19}	32	196	55	5.1	2^{182}	2^{165}
Dilithium 5:	(8, 7)	60	2	2^{19}	32	120	75	4.0	2^{252}	2^{229}

[DEEP] **Dilithium Keypair Generation**

Simplest and Fastest Operation in Dilithium

02. ρ, ρ', $K \leftarrow$ random or H(Seed) *// Public and secret seed values.*
03. $\hat{\mathbf{A}} \leftarrow$ ExpandA(ρ) *// Public $\hat{\mathbf{A}}$ has size $k \times l \times R_q$, derived from ρ.*
04. $\mathbf{s}_1 \leftarrow$ ExpandS(ρ', 0,2,..,l-1) *// Secret $\mathbf{s}_1$ has size $l \times R_q$, distribution $[-\eta, +\eta]$.*
    $\mathbf{s}_2 \leftarrow$ ExpandS(ρ', l, ..,l+k-1) *// Secret $\mathbf{s}_2$ has size $k \times R_q$, distribution $[-\eta, +\eta]$.*
05. $\mathbf{t} \leftarrow \mathbf{A} \cdot \mathbf{s}_1 + \mathbf{s}_2$ *// All of $\mathbf{t}$ is secure. $\mathbf{A} \cdot \mathbf{s}_1 = NTT^{-1}(\hat{\mathbf{A}} \circ NTT(\mathbf{s}_1))$.*
06. $(\mathbf{t}_1, \mathbf{t}_0) \leftarrow$ Power2Round($\mathbf{t}$, d) *// Split $\mathbf{t}$; $\mathbf{t}_1$ high 13 bits, $\mathbf{t}_0$ low 10 bits.*
07. tr $\leftarrow$ H(ρ, $\mathbf{t}_1$) *// tr = SHAKE256(PK).*
08. return **PK** = (ρ, $\mathbf{t}_1$), **SK** = (ρ, K, tr, $\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{t}_0$)

- The actual secret key is just ($\mathbf{s}_1$, $\mathbf{s}_2$). The K variable is only used in non-randomized signing (where the same message and **SK** always give the same sig.)
- Note that ExpandS(ρ') deterministic sampling is only useful in testing. If one can get uniform $[-\eta, +\eta]$ numbers (basically $\mathbb{Z}_5$ and $\mathbb{Z}_9$) directly in shares, this is better.

[DEEP] Dilithium Signature Generation *(1 of 2)*

Create a randomized "challenge" based on the message

09.	$\hat{\mathbf{A}}$	$\leftarrow$	ExpandA(ρ)	// *A has size $k \times l \times R_q$, derived from ρ.*
10.	μ	$\leftarrow$	H(tr || M)	// *512-bit message hash with H(PK) prefix.*
11.	κ	$\leftarrow$	0, $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$	// *Iteration counter κ, Iteration result.*
12.	ρ'	$\leftarrow$	random [or H(K, μ)]	// *[Use hash in deterministic signing.]*
13.	while $(\mathbf{z}, \mathbf{h}) = \perp$ do:			// — REJECTION LOOP —
14.	| $\mathbf{y}$	$\leftarrow$	ExpandMask(ρ', κ..)	// *$\mathbf{y}$ is $l \times R_q$ sampled from $[-\gamma_1, +\gamma_1]$.*
15.	| $\mathbf{w}$	$\leftarrow$	$\mathbf{A}*\mathbf{y}$	// *Compute as $\mathbf{w} = NTT^{-1}(\hat{\mathbf{A}} \circ NTT(\mathbf{y}))$.*
16.	| $\mathbf{w}_1$	$\leftarrow$	$\text{HighBits}_q(\mathbf{w}, 2\gamma_2)$	// *$\mathbf{w}_1$ range is $(q-1)/2\gamma_2$ so [0,15] or [0,43].*
17.	| $\tilde{c}$	$\leftarrow$	H($\boldsymbol{\mu}$, $\mathbf{w}_1$)	// *$\tilde{c}$ is derived from message and public key.*
18.	| $\mathbf{c}$	$\leftarrow$	SampleInBall($\tilde{c}$)	// *$\mathbf{c}$ is in R_q, has τ non-zero (± 1) coefficients.*
19.	| $\mathbf{z}$	$\leftarrow$	$\mathbf{y} + \mathbf{c}*\mathbf{s}_1$	// *It's better to store $NTT(\mathbf{s}_1)$ – as shares.*

*That's the arithmetic for $\tilde{c}$ and **z**. We must reject them and "goto 14" if some checks fail..*

[DEEP] **Dilithium Signature Generation** ***(2 of 2)***

Based on "Fiat-Shamir with Aborts" - Rejection Iteration

20. | $\mathbf{r_0} \leftarrow$ LowBits($\mathbf{w} - \mathbf{c} * \mathbf{s_2}$, $2\gamma_2$) *// Range is basically $\pm 2\gamma_2$*
21. | if MaxAbs($\mathbf{z}$) $\geq \gamma_1 - \beta$ or MaxAbs($\mathbf{r_0}$) $\geq \gamma_2 - \beta$ then: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$ *// reject*
22. | else:
23. | $\mathbf{h} \leftarrow$ MakeHint($- \mathbf{c} * \mathbf{t_0}$, $\mathbf{w} - \mathbf{c} * \mathbf{s_2} - \mathbf{c} * \mathbf{t_0}$, $2\gamma_2$) *// $\mathbf{h} \in \{0,1\}^{kN}$*
24. | if MaxAbs($\mathbf{c} * \mathbf{t_0}$) $> \gamma_2$ or CountOnes($\mathbf{h}$) $> \omega$ then: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$ *// reject*
25. | $\kappa \leftarrow \kappa + l$ *// For creating fresh **y** in next iteration*

end while

26. return Sig = ($\tilde{c}$, $\mathbf{z}$, $\mathbf{h}$) *// no longer secret*

- Protecting just the ($\mathbf{s_1}$, $\mathbf{s_2}$) secret itself via masking is easy; NTT in shares.
- Leaking the one-time secret $\mathbf{y}$ also breaks things; use masked arithmetic.
- MaxAbs and SampleInBall are very tricky to implement in masked format.
- The protected variables become non-secret (signature) after passing the check.

[DEEP] **Dilithium Signature Verification**

For completeness – Luckily doesn't involve secrets

{ T, F } = **Verify(** Sig, M, PK):

	$(\tilde{c}, \mathbf{z}, \mathbf{h}) \leftarrow$ Sig	// *Deserialize signature.*
	$(\rho, \mathbf{t}_1) \leftarrow$ PK	// *Deserialize public key.*
27.	$\hat{\mathbf{A}} \leftarrow \mathrm{ExpandA}(\rho)$	// *"Lattice" in NTT transformed domain.*
28.	$\mu \leftarrow H(\, H(PK), M \,)$	// *Prefix the message hash with H(PK).*
29.	$\mathbf{c} \leftarrow \mathrm{SampleInBall}(\tilde{c})$	// *Hash to τ non-zero (± 1) coefficients.*
30.	$\mathbf{w'}_1 \leftarrow \mathrm{UseHint}_q(\, \mathbf{h}, \mathbf{A}{*}\mathbf{z} - \mathbf{c}{*}\mathbf{t}_1 \cdot 2^d, 2\gamma_2 \,)$	// *Hint helps make w'_1 exactly matching.*

31. if $\mathrm{MaxAbs}(\, \mathbf{z} \,) < \gamma_1 - \beta$ and $\tilde{c} = H(\, \mu \,||\, \mathbf{w'}_1)$ and $\mathrm{CountOnes}(\mathbf{h}) \leq \omega$ then:

| return T 👍 "Good signature"

else:

| return F 👎 "Fail!"

Outline

Post-Quantum Crypto Security Engineering & Validation

***Note:** This is a large slideset; I will be be skimming over some parts.*

1. Motivation: NIST PQC and “Non-Invasive” in FIPS 140-3.

2. CRYSTALS-Kyber: Key Establishment.

3. CRYSTALS-Dilithium: Signatures.

4. **ISO 17825 / “FIPS 140-3” TVLA in Side-Channel Testing of PQC.**

FIPS 140-3 Non-Invasive Security

Also Known as Side-Channel Testing

Introduced as a major change in FIPS 140-3 in relation to FIPS 140-2:

- Side-Channel Attacks (Power, Emissions, Timing) are in 140-3 scope.
- Documentation required for Levels 1 and 2. Mitigation Testing at Levels 3 and 4.

But how?

- Initially (when FIPS 140-3 started): not tested (only "if claimed by a vendor".)
- Annex F of ISO 19790:2012 had no test metrics, but the *draft* SP 800-140F Rev 1 had ISO 17825, "Testing methods for mitigation of non-invasive attack classes."
- Updated Annex F of ISO 19790:2022 will reference ISO 17825 **directly**.

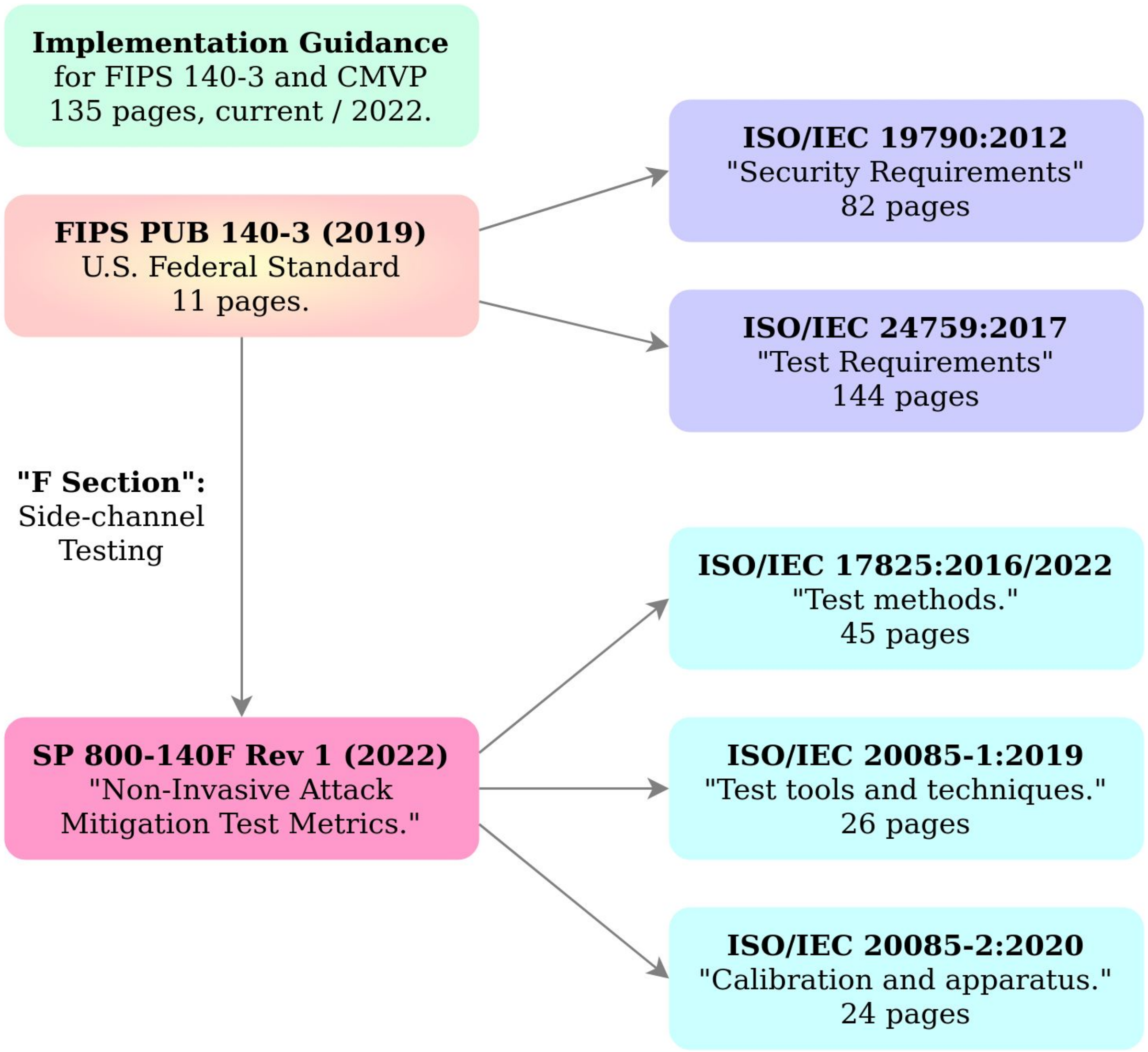
Complicated? "Non-Invasive" and FIPS 140-3

NIST Special Publication		ISO/IEC 19790:2012(E)	ISO/IEC 24759:2017(E)
SP 800-140		--	§6.1 through §6.12
SP 800-140A		Annex A	§6.13
SP 800-140B		Annex B	§6.14
SP 800-140C	modifies	Annex C	§6.15
SP 800-140D		Annex D	§6.16
SP 800-140E		Annex E	§6.17
SP 800-140F		Annex F	§6.18

NIST SP 800-140F Rev. 1 (Draft) CMVP Approved Non-Invasive Attack Mitigation Test Metrics

Document Revisions

Edition	Date	Change
Revision 1	[date]	**§ 6.2 Approved non-invasive attack mitigation test metrics** Added: ISO/IEC 17825 and associated ISO/IEC 20085-1 and -2



Non-Invasive and Post-Quantum

“Testing methods for the mitigation of non-invasive attack classes”

ISO/IEC WD 17825:2021(E) is based on Test Vector Leakage Assessment (“TVLA.”)
It does not try measure the difficulty of attack (like CC AVA_VAN); just detect leakage.

The standard text starts out with: *The test approach employed in this International Standard is an efficient “push-button” approach: the tests are technically sound, repeatable and have moderate costs. [!]*

Reality:

- That’s for testing labs ~2025. A well-defined “push-button” does really exist yet.
- However, one of the things already vendors use internally for sign-off assurance.

ISO 17825 Leakage Analysis Scenario

DPA and DEMA: Power and Electromagnetic Emission Traces

- **Standard attack setting**: Tester can set inputs to the module at the IO boundary (API). Can choose inputs and synchronize to the start of the operation.

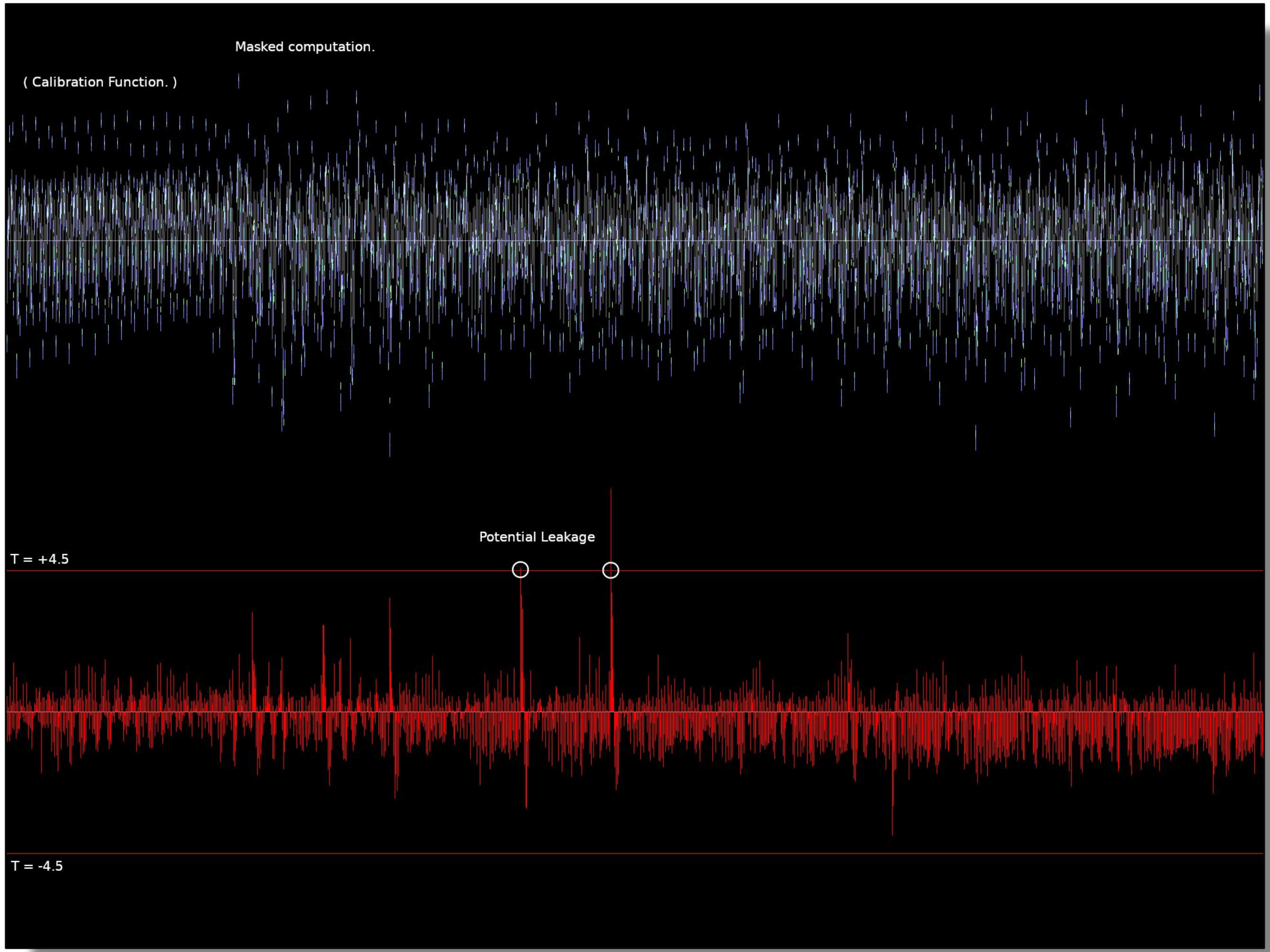
- **Oscilloscope measures power** (or electromagnetic emissions) at high precision, perhaps a couple samples per clock cycle. Measurement vectors are “traces”.

- **Traces are analyzed** to detect leakage. In leakage analysis the analyst can know or choose keys: Is looking for correlations between keys and and the traces.

- **Statistical analysis of significance.** PASS/FAIL metric (no key recovery).

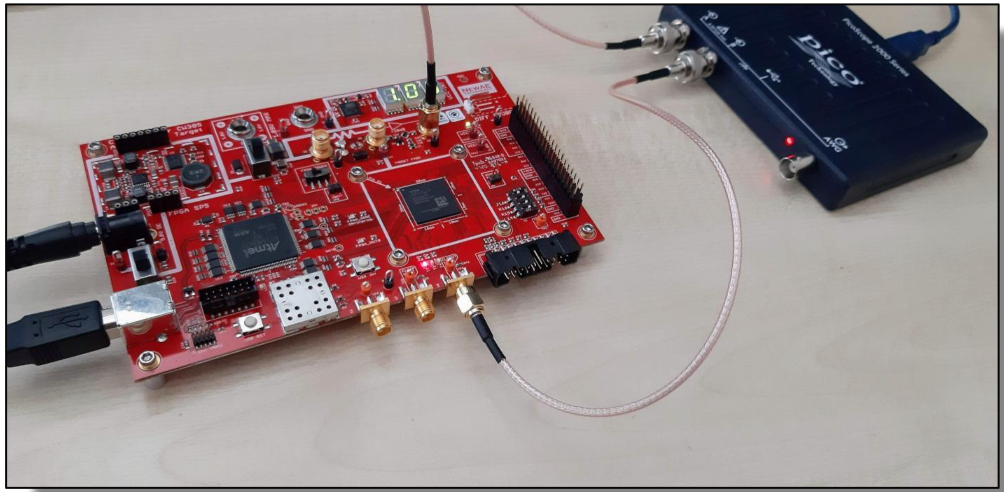
Side Channels: FPGA Leakage Emulation

ISO/IEC 17825 & 20085 - PQC Side-Channel Tests



👉 We use FPGA to emulate leakage of hardware post-quantum crypto modules. Try to apply ISO 17825.

👇 CW305 *"artefact"* as discussed in Annex C of ISO/IEC 20085-2:2020(E).



What are the “non-invasive mitigations” like?

Expect masking + ad hoc countermeasures

- Masking splits secrets into “shares.” Successful measurement of an individual share does not leak the secret itself. “Masking Gadgets” used to perform arithmetic steps.

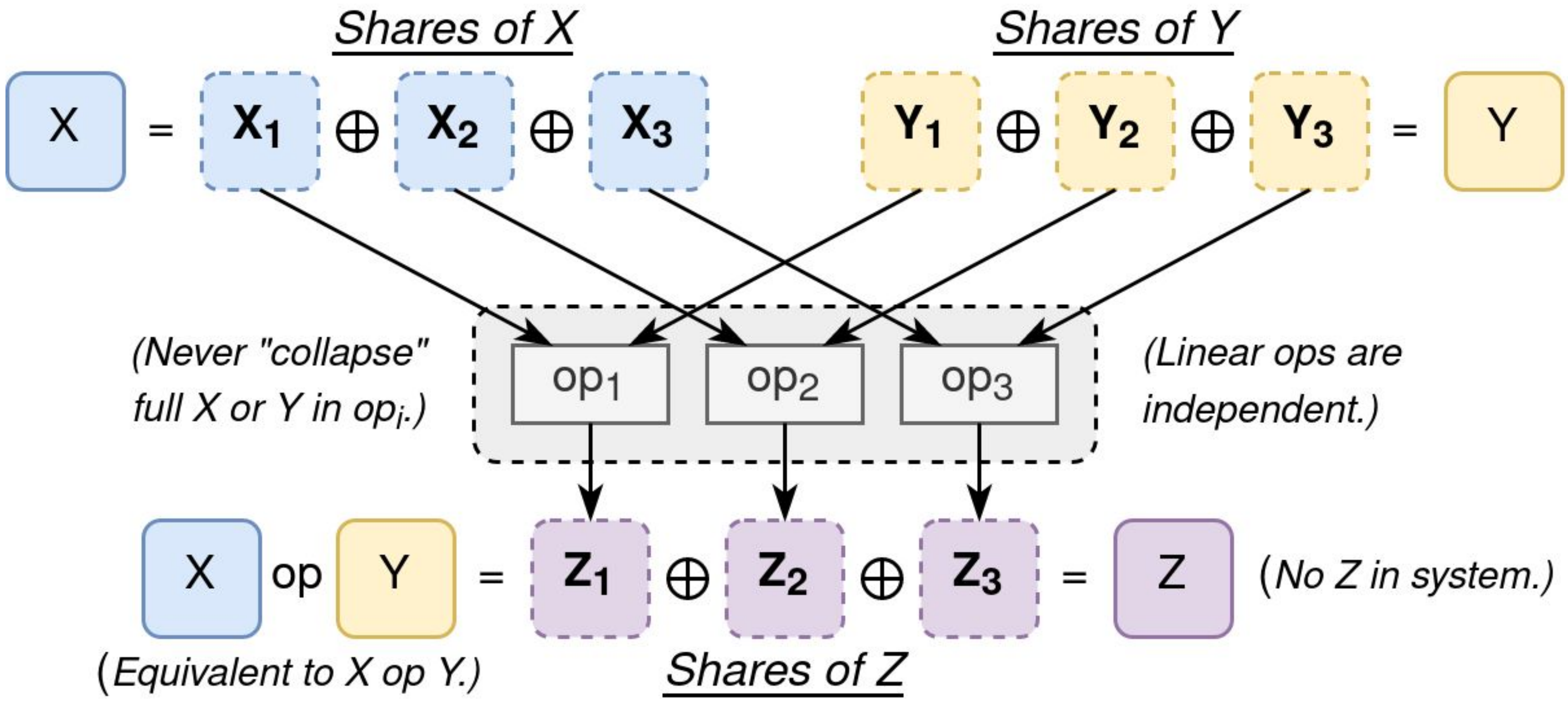
Type:	Relationship:	Algebraic object:
A/Q:	$X = X0 + X1 \pmod q$	Prime q is 3329 (Kyber) or 8380417 (Dilithium).
A/N:	$X = X0 + X1 \pmod{2^N}$	Size N is 16 or smaller (NTRU 11..14, Saber 14).
B:	$X = X0 \oplus X1$	Bit strings (managed e.g. as 64-bit words).

- **Most cryptographers agree**: Masking and other attack mitigation techniques for PQC algorithms are technically more complex than for older cryptography.

- **Why?** The algorithms are not homogenous like RSA or ECC but contain a number of dissimilar steps. One may have to design a dozen different gadgets for one algorithm.

The main countermeasure: Masking

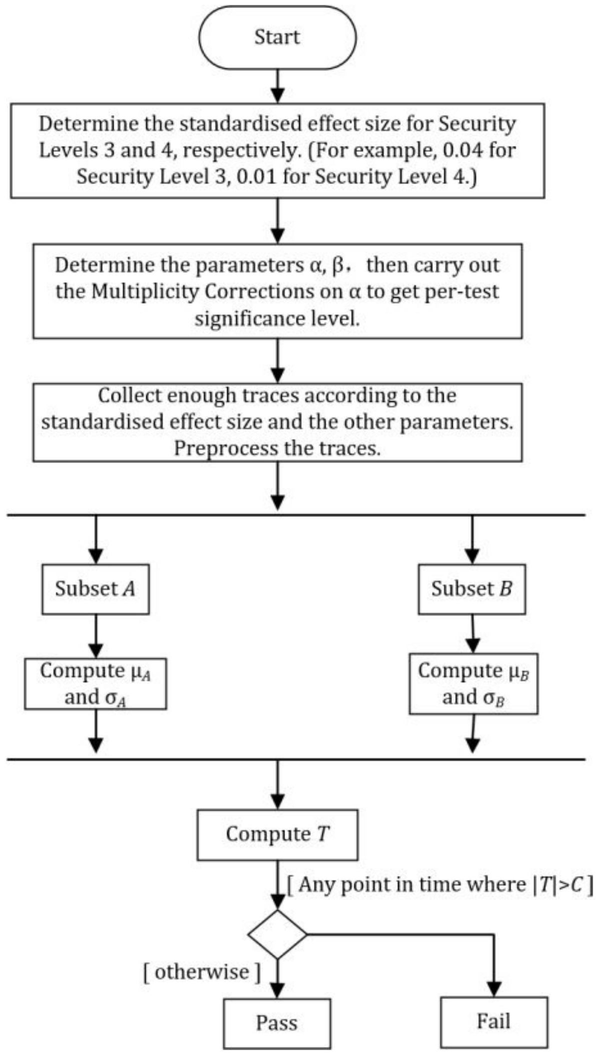
Limit leakage by breaking computation into randomized shares



Basic SCA Tests for Post-Quantum Crypto

Detects "leakage" – no key recovery (easily False Positives)

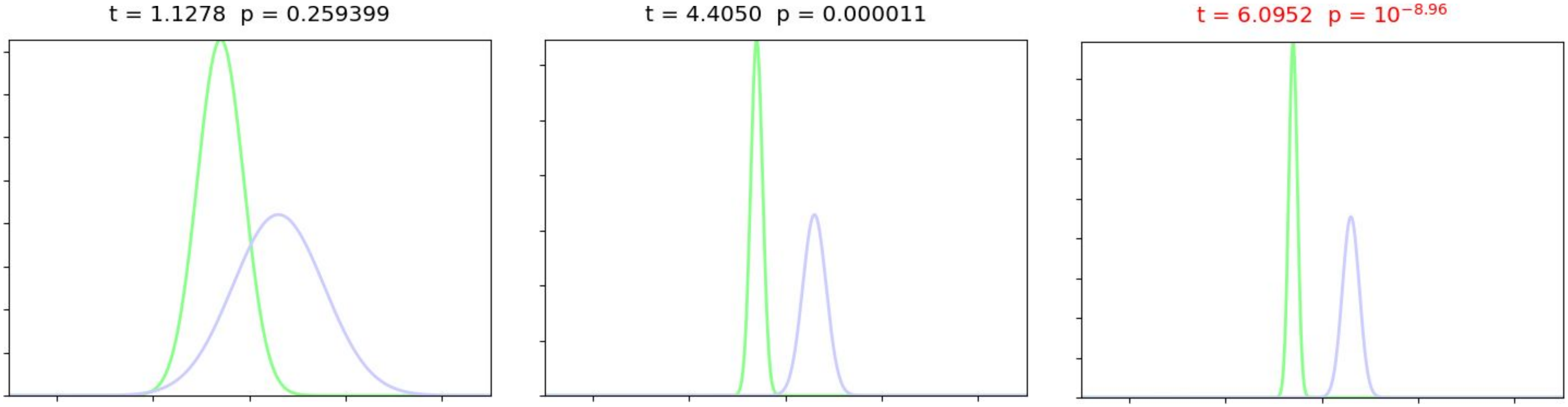
- ISO 17825 has a "general statistical test procedure."
- The current version of these tests create data subsets A and B of measurements (e.g., trace waveforms) with the IUT.
- But the trace sets A and B need input test vectors!
- **Example**: Set A may use a fixed bit value in a CSP, while measurements in set B use random CSP values.
- If the A/B measurement sets can be distinguished from each other – with the Welch t-test with high enough statistical confidence – this is taken as evidence of CSP leakage.



Simple math: Non-specific (Welch) t-test

Leakage is assumed when A and B don't have the same mean

- Subsets A and B are trigger - synchronized. Has sub-cycle precision (under 1ns).
- For each time sample, compute averages (μ_A, μ_B) and standard deviations (σ_A, σ_B).
- t-statistic relates to the certainty that the two sets are distinguishable.
- Confidence "probability" assumes Gaussians distribution (here normalized by $1/\sqrt{N}$).



ISO 17825 "General Statistical Test Procedure"

Outline of the General Statistical Test Procedure

0. Determine the required sample size $N = N_A + N_B$ and t-test threshold C from the experiment parameters.
1. Collect Subsets A and B and compute their pointwise averages (μ_A, μ_B) and standard deviations (σ_A, σ_B).
2. Compute the pointwise Welch t-test statistic vector

$$T = \frac{\mu_A - \mu_B}{\sqrt{\frac{\sigma_A^2}{N_A} + \frac{\sigma_B^2}{N_B}}}.$$

3. If at any point $|T| > C$, the test results in a FAIL. If the threshold was is not crossed, the test is a PASS.

External API Interfaces for SCA testing

Using handles: Testing just the core private key operation

Tester: Create inputs (load test vectors or compute them).

↓

key_handle = **key_import():** Deserializes CSPs into module's internal memory layout.

↓

——————— *Trigger Measurement Start.*———————

ss_handle = **decaps(** ct**,** key_handle **)**

——————— *Trigger Measurement End.* ———————

↓

ss_tv = **key_export(** ss_handle **):** Collapse shares and extract results from memory.

↓

Tester: Verify results, store measurement.

Also test secure CSP import and export

ISO 17825 Requires testing at "Module I/O Boundary."

Using secure import (and export for keygen, encaps, decaps)

Tester: Create inputs (load test vectors or compute them).

↓

——————— *Trigger Measurement Start.* ———————

signed = **sign(** wrapped_key, msg **):**

handle = **key_import(** wrapped_key **)**: Deserialize CSPs.

signed = **sign(** handle, msg **):** Private key operation.

——————— *Trigger Measurement End.* ———————

↓

Tester: Verify results, store measurement.

Goals of Automatic TVLA "Sign-Off"

Leakage tests should aim for widest possible coverage

1. **Try to have specific testing coverage over all CSPs in all relevant sub-algorithms.**

   (Key Generation, Key Export, Import, Encapsulation, Decapsulation, Signature.)

2. **Design the experiments and test vectors (input data) in a way that eliminates false positives to greatest extent possible.**

   (Hopefully no need to specify "areas of interest" in resulting traces.)

Opinion: Industry will need to agree on a standardized set of test vectors in order to have consistent results. These are dependant on details of each algorithm.

Two basic types of test vectors will get you far

Fixed vs Random (FIX) and A/B Classification (ABC)

1. **Fixed vs Random** (non-specific t-test) can be used in "live" testing:
   - Trace set A: Fixed CSP for every trace.
   - Trace set B: New random CSP secret for each trace.

2. **A/B Categorization** works with capture-then-analyze flow:
   - Records traces with detailed test vector metadata; CSPs are known in analysis.
   - Traces are categorized *after capture* to A and B sets based on CSP selection criteria,
     Examples: a specific internal CSP variable or secret key bit, "plaintext checking" bit.
   - The same trace data can be categorized to A and B in a number of different ways.

In both cases: Set A and Set B statistically differentiable with t-test = **FAIL**.

Example 1: Fixed-vs-Random

Fixed-vs-Random on Secret Key on Kyber Decrypt

M = **Kyber.CPAPKE.Dec**(**CT**, **S**):

0-2.	$(u, v) \leftarrow$ (decode) CT	*// Decode u and v from ciphertext.*
3.	$\hat{s}^T \leftarrow$ (decode) **S**	*// Decode (and refresh) secret key.*
4.	$m \leftarrow v - \mathrm{NTT}^{-1}(\hat{s}^T \circ \mathrm{NTT}(u))$	*// NTT arithmetic for:* $m = v - s^T \cdot u$.
5.	return $M = \mathrm{Compress}_q(m, 1)$	*// Extract the "signs" as* $M \in \{0,1\}^{256}$.

Subset **A**: Fixed secret key **S** but a fresh ciphertext **CT** for every trace.

Subset **B:** Random (**S**, **CT**) generated for each decryption trace.

If one can statistically distinguish **A** from **B**, then there is probably leakage from **S**.

CSP: Actual Kyber (CCA KEM) Private Key

More complex because of Fujisaki-Okamoto Transform

Recall that ISO 17825 tests are done at the "Module I/O" boundary, i.e. with CCA:

CCA: **CT** = (Encode(c_m) ǁ Encode(b'))
PK = (Encode(t) ǁ ρ)
SK = (Encode(**s**) ǁ Encode(t) ǁ ρ ǁ Hash(PK) ǁ **z**)

The CCA secret key contains the public key too – because of re-encryption!

- If we just pick a random SK (cca), then we' will getting irrelevant leakage indications (false positives) from the public parameters, as those are not masked at all.

- False positives are similar to "leaking" public modulus n in RSA, or public point in ECC.

Test Vector Creation

For all lattice schemes – Signature and CCA KEM

Standard format PQC secret keys are complex mixtures of secret and public information:

- Kyber (CCA) **SK** = (Encode(**s**) ‖ Encode(t) ‖ ρ ‖ Hash(PK) ‖ **z**)
- Dilithium **SK** = (ρ ‖ K ‖ Hash(ρ, t_1) ‖ Encode($\mathbf{s_1}$) ‖ Encode($\mathbf{s_2}$) ‖ t_0)

Avoiding false positives from non-CSPs; we’d want to keep the public (ρ or seed_A) values static and only manipulate private polynomial **s**. This is analogous to keeping the “curve” constant with elliptic curves and just looking for leakage in the scalars.

As with RSA and ECC, the procedure for high-level test vector generation depends on the algorithm structure. We’re proposing test vectors that “activate” CSP components only.

Example 2: Plaintext Checking Oracle

Re-encrypt & check in Fujisaki-Okamoto is Extremely Fragile

- A Plaintext Checking (PC) oracle leaks information about the **M** == **M**’ comparison.
- Leakage from steps 2-4 can do that.
- The PC oracle bit can be used to efficiently break Kyber (extract **S**) in adaptive attack.

Even though test vectors are not adaptive, we test *indirectly* for PC oracle in Decapsulation e.g. by mismatching secret key with ciphertext in Set B.

Subset **A**: CT = CT’. (“Valid ciphertext.”)

Subset **B**: CT ≠ CT’. (“Invalid ciphertext.”)

<u>**CCA.Decaps(CT, SK):**</u>

0. (H(PK), **Z**, **S**) ← **SK**
1. **M’** ← CPA.Decrypt(**S**, CT)
2. (**K’**, **R’**) ← H(**M’**, H(PK))
3. **CT’** ← CPA.Encrypt(PK, **M’**, **R’**)
4. if CT == **CT’** then:
5. | **SS’** ← H(**K’**, H(CT))
6. else:
7. | **SS’** ← H(**Z**, H(CT))
8. return **SS’**

ISO 17825 for NIST PQC: Conclusions

- ISO 17825 / TVLA leakage tests are useful as a sign-off and positive assurance. *No key recovery, attack potential grading – has different goals from AVA_VAN.*

- ISO 17825 being adopted FIPS 140-3 and can be used on Post-Quantum Crypto.

- Such testing should focus on *coverage;* aim to test all CSPs, everywhere. But care must be taken to avoid false positives (e.g. detection of PSP variables).

Big caveat: Do not let such testing replace security analysis in the design process!

"When a measure becomes a target, it ceases to be a good measure".

– Goodhart's law (of unintended consequences.)