WrapQ:

Side-Channel Secure Key Management for Post-Quantum Cryptography

Markku-Juhani O. Saarinen

PQShield, UK and Tampere University, Finland

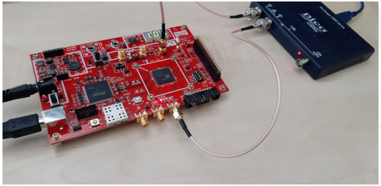
18 August 2023
PQCrypto 2023 – College Park, MD, USA

Outline of the Talk

1. **Intro: Side-Channels, Kyber, Dilithium, and Masking**
2. The "WrapQ Trick" and Secret Key Encoding Formats
3. Implementation and Leakage Assessment

Side-Channel Attacks

- **Side-Channel Attacks (SCA)** use external measurements such as latency (TA), power consumption (SPA/DPA), or electromagnetic emissions ([S/D]EMA) to extract secrets.
- *SCA resistance is important for PC, IoT, and mobile device "platform security" (secure boot, firmware updates, attestation), authentication tokens, smart cards, HSMs / secure elements..*
- Common compliance & market requirement for hardware (Common Criteria / AVA_VAN, FIPS 140-3 / ISO 17825).



- **Post-Quantum Cryptography (PQC)** implementations – e.g. lattice-based schemes **Dilithium** and **Kyber** inherit all of the security and compliance requirements of Elliptic Curve or RSA based solutions in applications.

Masking: Non-Invasive SCA Security

- **Masking:** Secret data $[\![\mathbf{s}]\!]$ is processed in d randomized shares $\mathbf{s}_i$.

$$\text{Boolean Masking:} \quad [\![\mathbf{s}]\!] = \mathbf{s}_1 \oplus \mathbf{s}_2 \oplus \cdots \oplus \mathbf{s}_d$$
$$\text{Arithmetic Masking:} \quad [\![\mathbf{s}]\!] = \mathbf{s}_1 + \mathbf{s}_2 + \cdots + \mathbf{s}_d \pmod{q}.$$

- Individually each share $\mathbf{s}_i$ is uniformly random, as is any combination if $d-1$ shares.

- A bit like d-of-d secret sharing: Even full knowledge of $d-1$ shares $\sum_{i=1}^{d-1} \mathbf{s}_i$ reveals nothing about $[\![\mathbf{s}]\!] = \sum_{i=1}^{d} \mathbf{s}_i$. You need all d shares. We call $d-1=t$ the *masking order*.

- If you only have partial or "noisy" measurements (traces), it has been shown that the number of such observations required to learn $[\![\mathbf{s}]\!]$ grows *exponentially* with d.
  (Chari et al. 1999 – a lot of subsequent theoretical and experimental work supports this.)

Masking Gadgets, Models, and Proofs

Computation on masked shares must be arranged so intermediate variables have no statistical correlation with the actual secret variables. They need to appear random too.

- **Gadgets:** Common approach is to first design a set of "gadgets" for simple operations (logical AND, selection, bit shift, etc.) and compose larger algorithms from them.
- **Refreshing:** Masking security generally requires that a particular secret sharing of variable $[\![\mathbf{s}]\!]$ can only be used once; after that, it needs to be *refreshed* (re-randomized).
- **Proofs:** The proofs can be made in several models; the Ishai-Sahai-Wagner (ISW) <u>*t*-probing security</u> requires that any *t* internal intermediate values don't reveal secrets.

  The <u>noisy leakage model</u> is an alternative; links have been proven between *t*-probing security, noisy leakage model, and information-theoretic attack complexity bounds.

Masking and Secret Keys in Practice

Masking is the best known method to secure Kyber and Dilithium. Practical impact on keys:

1. **The representation is now completely different!** It's in arithmetic and Boolean shares.
   You can't use the regular "packed" secret key formats – the keys must be masked all the time.

2. **Every time the key is used, we need to refresh the masking and overwrite the old key.**
   Without refresh, the shares are effectively just a bigger representation of an unmasked key!

3. **Masked keys are much bigger.** For example, Kyber's $\hat{\mathbf{s}}$ is 12288 bits and Dilithium's $(\mathbf{s}_1, \mathbf{s}_2)$ is 88320 bits at Level 5. Multiply that with d; first order $2\times$, second order $3\times$...

We have a trick for (2) and (3), but can't use the same format (1). We need key encryption and will add integrity as well – wrapped keys can then be stored on an untrusted medium.

Outline of the Talk

1. Intro: Side-Channels, Kyber, Dilithium, and Masking
2. **The “WrapQ Trick” and Secret Key Encoding Formats**
3. Implementation and Leakage Assessment

Decrypting Ciphertext C into Shares $[\![P]\!]$

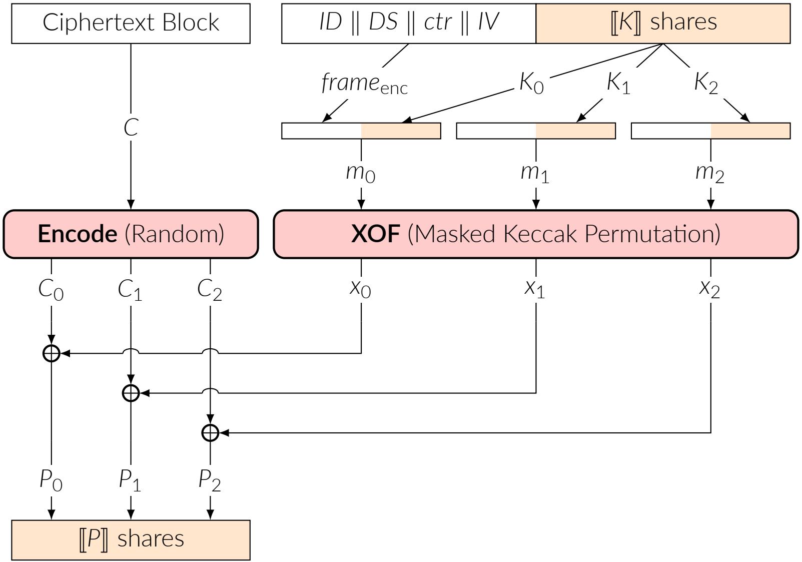
- Having 1 plaintext share leaks, but we can encrypt it. Leaking ciphertext is okay!
- But we need to decrypt directly into d randomized shares, different every time. We can do this by having a "stream cipher" that produces "masked keystream".
- Masked Kyber and Dilithium needs a masked SHAKE anyway (masks in – masks out.)

DecBlock$(C, [[K]], ID, ctr, IV)$

Input: C, Ciphertext block (Unmasked), $[[K]]$, Key Encryption Key (Boolean Masked).
Input: ID, ctr, IV: Used by the counter mode to construct unique $frame_{enc}$ (unmasked).
Output: $[[P]]$: Decrypted key material payload (Boolean masked.)

1: $[[C]] \leftarrow \text{Encode}(C)$ ▷ It's still ciphertext, but randomized into shares.
2: $[[x]] \leftarrow \text{XOF}_{|P|}(\ frame_{enc} \parallel [[K]]\ )$ ▷ Masked XOF in counter mode: Masked keystream.
3: $[[P]] \leftarrow [[C]] \oplus [[x]]$ ▷ Masked stream cipher into masked plaintext.
4: $[[K]] \leftarrow \text{Refresh}([[K]])$ ▷ Key Encryption Key needs a refresh.
5: **return** $[[P]] = \text{Refresh}([[P]])$

Decrypting Ciphertext C into Shares $[\![P]\!]$ (Three Shares)



Encrypting Masked Shares $[\![P]\!]$ into Ciphertext C

- Encrypt has the same steps in reverse; XOR masked plaintext with masked keystream.
- Note: Most masked AES modules only mask the key; not the plaintext or ciphertext.
- The final (collapsed) ciphertext is C is equivalent to having used an unmasked XOF.

$C = \mathsf{EncBlock}([[P]], [[K]], ID, ctr, IV)$

Input: $[[P]]$, Payload block (Boolean masked, $[[K]]$, Key Encryption Key (Boolean Masked).
Input: ID, ctr, IV: Used to construct header $frame_{\mathrm{enc}}$.
Output: C, Resulting ciphertext block.

1: $[[x]] \leftarrow \mathsf{XOF}_{|P|}(\ frame_{\mathrm{enc}} \parallel [[K]]\ )$ ▷ Generate a block of masked keystream.
2: $[[C]] \leftarrow [[P]] \oplus [[x]]$ ▷ "Masked Stream cipher Encryption."
3: $[[K]] \leftarrow \mathsf{Refresh}([[K]])$ ▷ Referesh the Key Encryption Key (KEK).
4: $[[P]] \leftarrow \mathsf{Refresh}([[P]])$ ▷ We can also just discard/zeroize plaintext here.
5: **return** $C = \mathsf{Decode}([[C]])$ ▷ It's encrypted; we can safely collapse the shares.

We need integrity too

- WrapQ is Encrypt-then-MAC (EtM); ciphertext is authenticated rather than plaintext.
- We can use a faster non-masked hash to process the ciphertext and associated data.
- Use the masked XOF only to bind it with the integrity key (process one block).
- WrapQ block can be in untrusted memory. Only KEK (256 bits) needs secure storage.

$T = \mathsf{AuthTag}(\ A, [[K]], ID, ctr, IV\ )$

Input: A, Authenticated data, including ciphertext.
Input: $[[K]]$, Message Integrity Key (Boolean masked.)
Input: ID, ctr, IV: Used to construct $frame_{DS}$ headers for domain separation.
Output: T, Resulting authentication tag/code.

1: $h \leftarrow \mathsf{Hash}(\ frame_{\mathrm{hash}} \parallel A\ )$ ▷ HAsh ciphertext and associated data.
2: $[[T]] \leftarrow \mathsf{XOF}_{|T|}(\ frame_{\mathrm{mac}} \parallel [[K]] \parallel h\ )$ ▷ Masked: Bind hash with secret integrity key.
3: $[[K]] \leftarrow \mathsf{Refresh}([[K]])$ ▷ Refresh the integrity key.
4: **return** $T = \mathsf{Decode}([[T]])$ ▷ Authentication tag is public.

Standard Dilithium Key Format & Classification

CRYSTALS-Dilithium	Public Key	Secret Key
Standard encoding	$pk = (\rho, \mathbf{t}_1)$	$sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$

Field	Size (bits)	Classification / Description
ρ	256	PSP: Seed for public $\mathbf{A}$.
$\mathbf{t}_1$	$k \times 10 \times 256$	PSP: Upper half of public $\mathbf{t}$.
K	256	CSP: Seed for deterministic signing.
tr	256^*	PSP: Hash of public key $tr = H(\rho \parallel \mathbf{t}_1)$.
$\mathbf{s}_1$	$\ell \times d_\eta \times 256$	CSP: Secret vector 1, coefficients $[-\eta, \eta]$.
$\mathbf{s}_2$	$k \times d_\eta \times 256$	CSP: Secret vector 2, coefficients $[-\eta, \eta]$.
$\mathbf{t}_0$	$k \times 13 \times 256$	PSP: Lower half of public $\mathbf{t}$.

*: tr may be increased to 512 bits.

Dilithium WrapQ: Encrypt Secrets, Authenticate All

WrapQ Dilithium Key: $sk_{wq} = (ID, T, IV, \rho, K, tr, \mathbf{s}_1, \mathbf{s}_2)$

Field	**Size (bits)**	**Description**
ID	32	Algorithm and serialization type identifier.
T	256	Authentication tag for integrity.
IV	256	Random nonce.
ρ	256	Authenticated: Public seed for $\mathbf{A}$.
K	256	Encrypted: Seed for deterministic signing.
tr	256^*	Authenticated: Hash $tr = \mathsf{SHAKE256}(pk)$.
$\mathbf{t}_0$	$k \times 13 \times 256$	Authenticated: Lower half of public $\mathbf{t}$.
$\mathbf{s}_1$	$\ell \times 4 \times 256$	Encrypted: Secret vector 1.
$\mathbf{s}_2$	$k \times 4 \times 256$	Encrypted: Secret vector 2.

*: tr may be increased to 512 bits.

Standard Kyber Key Format & Classification

CRYSTALS-Kyber	**Public Key**	**Secret Key**
Standard encoding:	$pk = (\hat{\mathbf{t}}, \rho)$	$sk = (\hat{\mathbf{s}}, pk, pkh, z)$

Field	**Size (bits)**	**Classification / Description**
$\hat{\mathbf{t}}$	$k \times 12 \times 256$	PSP: Public vector, NTT domain.
ρ	256	PSP: Seed for public $\mathbf{A}$.
$\hat{\mathbf{s}}$	$k \times 12 \times 256$	CSP: Secret vector, NTT domain.
pk	$\lvert\hat{\mathbf{t}}\rvert + 256$	PSP: Full public key.
pkh	256	PSP: Hash of the public key $\mathbf{SHA3}(pk)$.
z	256	CSP: Fujisaki-Okamoto rejection secret.

→ Full public key *pk* (and its hash *pkh*) is contained in the standard-format secret key.

Kyber WrapQ: Encrypt Secrets, Authenticate All

WrapQ Secret Key: $sk_{wq} = (ID, T, IV, pkh, z, \mathbf{s})$

Field	**Size (bits)**	**Description**
ID	32	Algorithm and serialization type identifier.
T	256	Authentication tag for integrity.
IV	256	Random nonce.
pkh	256	Authenticated: Public key hash SHA3(pk).
z	256	Encrypted: FO Transform secret.
$\mathbf{s}$	$k \times 4 \times 256$	Encrypted: Secret key polynomials.

→ The WrapQ blob doesn't contain a copy of the public key $pk = (\hat{\mathbf{t}}, \rho)$. It is needed for decapsulation and must be provided separately. Public key is authenticated with pkh.

Secret Key Sizes: Standard vs. WrapQ

- WrapQ encoding is identical (size) for all masking orders. XOF order matters.
- There is no need to refresh the encrypted blob (write back after decaps or signing.)
- Kyber WrapQ is smaller; only auth data for public key, compact encoding for **s**.
- Dilithium WrapQ is not much bigger even with the IV and Authentication Tag.

Kyber and Dilithium – Encoding Sizes in Bytes

Algorithm			**Masking**	**Std. Encoding**		**WrapQ**
Variant Name	k	ℓ	Share	$\lvert pk \rvert$	$\lvert sk \rvert$	$\lvert sk_{wq} \rvert$
Kyber512	2		768	800	1,632	**388**
Kyber768	3		1,152	1,184	2,400	**516**
Kyber1024	4		1,536	1,568	3,168	**644**
Dilithium2	4	4	5,888	1,312	2,528	**2,852**
Dilithium3	6	5	8,096	1,952	4,000	**4,068**
Dilithium5	8	7	11,040	2,592	4,864	**5,412**

Outline of the Talk

1. Intro: Side-Channels, Kyber, Dilithium, and Masking
2. The "WrapQ Trick" and Secret Key Encoding Formats
3. **Implementation and Leakage Assessment**

Hardware Platform Overview

The target "chip" implements masked Kyber & Dilithium (all parameters) with first-order masking and some other SCA countermeasures. A version of a commercial ASIC IP.

Unmasked secret key formats "always" leak – secure key management is needed.

- Small RV64 CPU. No ISA extensions used: Memory-mapped HW control registers.
- Lattice accelerator for Kyber and Dilithium $\mathbb{Z}_q$ polynomials and NTT operations. It can also perform bit-vector manipulation for tasks such as masking conversions (A2B, B2A).
- Ascon-based random mask generator. Used by the lattice unit for refreshing Boolean and Arithmetic $(\bmod\ q)$ shares. It can be continuously seeded from an entropy source.
- XOF: Compact three-share Threshold Implementation (TI) of the Keccak Permutation.
- A faster, non-masked 1600-bit Keccak permutation used for public $\mathbf{A}$ matrix generation and also to compute PSP hashes (e.g., the h value in $\mathsf{AuthTag}$.).

WrapQ Key Import/Export was integrated and tested like the other components.

TVLA Sign-Off: Trace Acquisition

Traces acquired from the implementation on an XC7A100T2FTG256 Artix 7 FPGA chip on a ChipWhisperer CW305-A100 board, clocked at 50 MHz. Picoscope 6434E oscilloscopes with a 156.25 MHz sampling rate connected to the SMA connectors on the CW305 board.

Key Import and Export Tests

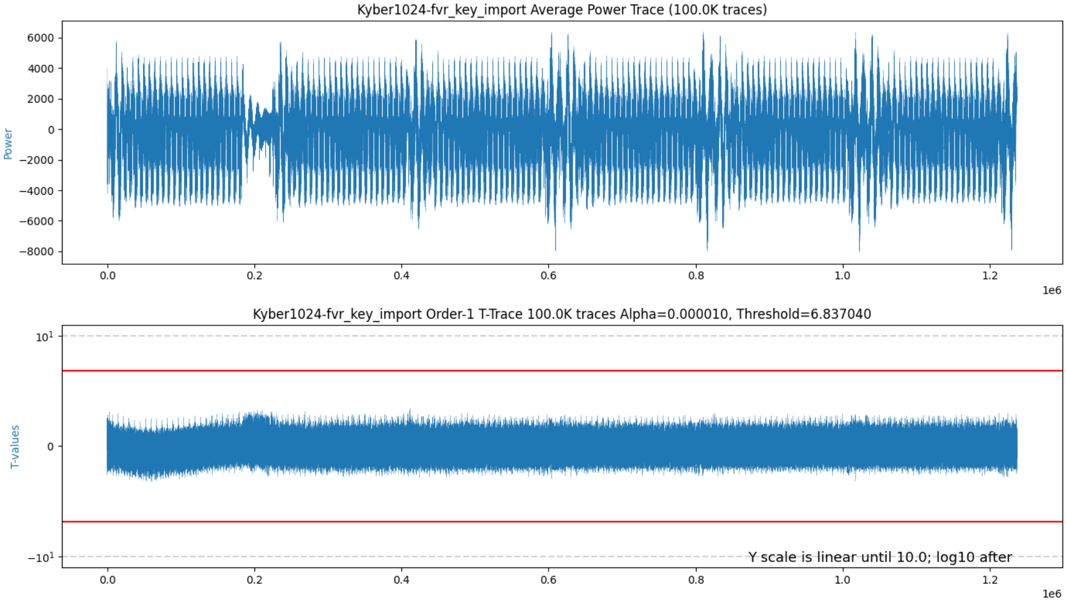
The ISO 17825 / TVLA type tests were designed to detect leakage from the KEK (Key-Encrypting Key) and the payload CSPs (Wrapped PQC Secret Keys.)

Summary of Random-vs-Fixed key import and export test types.

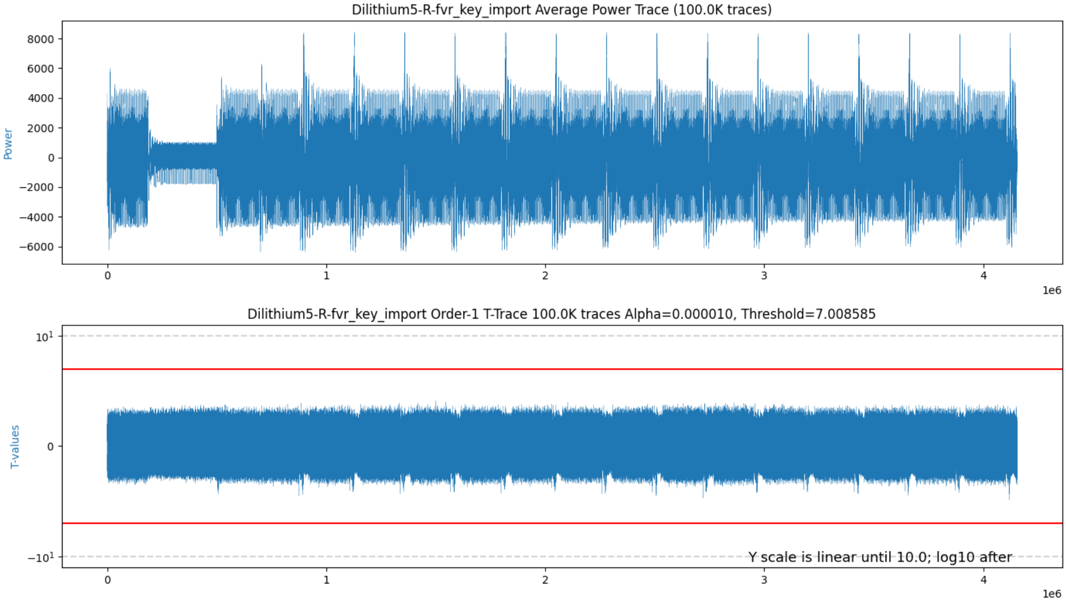
Test	Function	Set A	Set B	Both A&B
#1	Kyber Import	Fix CSP	Rand CSP	Fix KEK
#2	Kyber Import	Fix KEK	Rand KEK	Rand CSP
#3	Dilithium Import	Fix CSP	Rand CSP	Fix KEK
#4	Dilithium Import	Fix KEK	Rand KEK	Rand CSP
#5	Kyber Export	Fix CSP	Rand CSP	Fix KEK
#6	Kyber Export	Fix KEK	Rand KEK	Rand CSP
#7	Dilithium Export	Fix CSP	Rand CSP	Fix KEK
#8	Dilithium Export	Fix KEK	Rand KEK	Rand CSP

Industry-standard critical value calculation and calibration methods used. In Continuous Integration, the IUT passes the tests with $N = 100,000$ traces at all security levels.

Example 1: Kyber1024 WrapQ Key Import RvF CSP (#1)



Example 2: Dilithium5 WrapQ Key Import RvF CSP (#3)



Thank You! Conclusions

Motivating Research Problems

- Loading the secret key is "Step 0" of any private key op. It also needs to be secure!
- How to manage the "secret key write-back" refresh required in masking?
- PQC Secret Keys are big and don't easily fit into non-volatile secure storage.

Observations and Contributions in this Work:

- You can keep the secret key in compact 1-share encoding – if it's encrypted.
- There is a way to decrypt (unwrap) a key directly into randomized shares.
- PQC needs frequent Key Wrapping: Confidentiality and Integrity allow one to keep the big key material in less secure storage; just put the short KEK in secure storage.

Practical Level / Proof-of-Concept

- Secure Kyber and Dilithium have masked Keccak – use it as a masked stream cipher!
- Presented key variable sensitivity analysis, described the secret key formats.
- Implementation, leakage assessment of the import and export functions.