WrapQ: Side-Channel Secure Key Management for Post-Quantum Cryptography

Markku-Juhani O. Saarinen
PQShield Ltd.
Oxford, United Kingdom
mjos@pqshield.com

***Abstract*—Side-channel secure implementations of public-key cryptography algorithms must be able to load and store their secret keys safely. We describe WrapQ, a masking-friendly key management technique and encoding format for Kyber and Dilithium Critical Security Parameters (CSPs). WrapQ protects secret key integrity and confidentiality with a Key-Encrypting Key (KEK) and allows the keys to be stored on an untrusted medium. Importantly, its encryption and decryption processes avoid temporarily collapsing the masked asymmetric secret keys (which are plaintext payloads from the viewpoint of the wrapping primitive) into an unmasked format. We demonstrate that a masked Kyber or Dilithium private key can be loaded in a leakage-free fashion from a compact WrapQ format without updating the encoding in non-volatile memory. WrapQ has been implemented in a side-channel secure hardware module. Kyber and Dilithium wrapping and unwrapping functions were validated with 100K traces of TVLA-type leakage assessment.**

***Index Terms*—Side-Channel Security, Masking Countermeasures, Key Wrapping, Post-Quantum Cryptography, Kyber, and Dilithium**

I. Introduction

With the standardization of CRYSTALS suite algorithms Kyber [1] and Dilithium [2] as the preferred NIST Post-Quantum Cryptography (PQC) algorithms for key agreement and digital signatures [3], their secure and efficient implementation has become one of the most important engineering challenges in cryptography. NSA has also selected these two algorithms for the CNSA 2.0 suite for protecting classified information in National Security Systems [4].

Among other applications, Kyber and Dilithium will be gradually replacing older RSA and Elliptic Curve systems in systems where it is a requirement that the secret information held by a device (such as a mobile phone, payment card, or an authentication token) does not leak even if an adversary has access to the physical device (or its close proximity.) A related application is *platform security*, where cryptographic methods protect system integrity and system (firmware) updates against unauthorized modification and other attacks.

Side-Channel Attacks (SCA) use external physical measurements to derive information about the data being processed. Some of the most important ones are Timing Attacks (TA) [5], Differential Power Analysis (DPA) [6], and Differential Electromagnetic Analysis (DEMA) [7]. The attacks are powerful, and almost any implementation can be rapidly attacked if appropriate countermeasures are not in place.

A. Side-Channel Countermeasures for Lattice Cryptography

Masking [8] countermeasures have emerged as the most prominent and effective way to secure lattice-based cryptography against side-channel attacks. Masking is based on randomly splitting all secret variables into two or more shares.

Definition 1: Order-d masked encoding $[[x]]$ of a group element $x \in G$ consists of a tuple of $d+1$ shares $(x_0, x_1, \cdots, x_d), x_i \in G$ with $x_0 + x_1 + \cdots + x_d \equiv x$.

The addition operation can be defined in an arbitrary finite group G; Boolean masking uses the exclusive-or operation $\oplus$, while arithmetic masking uses modular addition. Vectors, matrices, and polynomials can also be used as shares.

A fundamental security requirement is that the shares are randomized so that all $d+1$ shares are required to reconstruct x, and any subset of only d shares reveals no statistical information about x itself. There are $|G|^d$ possible representations $[[x]]$ for x; *Mask refreshing* refers to a re-randomization procedure that maps $[[x]]$ to another encoding $[[x]]'$ of x.

Computation of cryptographic functions $[[y]] = f([[x]])$ is organized to maintain this poperty, avoiding leakage of x or y. It can be shown that the amount of side channel information required to learn x or y can be limited to grow exponentially in $d+1$, the number of shares. Any circuit can be transformed to use masking with at most $O(d^2)$ overhead [8], [9].

Several abstract models have been proposed for the purpose of providing theoretical proofs of security for masked implementations, including the Ishai-Sahai-Wagner probing model [9] and Prouff-Rivain noisy leakage model [10], [11]. Duc et al. provide a reduction from the latter to the first [12].

In addition to theoretical soundness, an essential advantage of masking countermeasures for PQC is that they are defined at a higher algorithmic level and are generally less dependent on the physical details of the implementation when compared to logic-level techniques such as dual-rail countermeasures [13]. However, it is essential to experimentally verify the leakage properties. There are standard approaches to leakage assessment of physical implementations [14]–[16].

Designers often proceed by describing a set of generic "gadgets" that make up the secured portion of the algorithm and then providing analysis for the composition. There already exists a large body of works discussing the masking of lattice cryptography schemes, including GLP [17], Dilithium [18] and Kyber [19], [20]. The issue of key management is generally not addressed in these works.

B. Private Keys and Secret Variables

Side-channel leakage can be exploited in any component that handles secret key material. Hence the key management processes must meet the same security requirements as cryptographic private key operations. Often the "zeroth" step of a private-key operation is "load private key." Clearly, the key can't be stored in non-masked plaintext format.

Masking generally requires that the shares are refreshed (re-randomized) every time they are used. A trivial solution is to write back the refreshed keys to non-volatile memory after each usage, which risks corruption of the keys.

Furthermore, masked representations significantly increase the secret key storage requirement. Secure, non-volatile key storage is a limited resource. Standard-format Kyber1024 private keys are 25,344 bits, while Dilithium5 secret keys are 38,912 bits (Table III.) This is an order of magnitude more than typical RSA keys and two orders of magnitude more than the keys of Elliptic Curve Cryptography schemes.

Key Wrapping [21], [22] refers to a process where Authenticated Encryption (AE) is used to protect the confidentiality and integrity of other key material, such as asymmetric keys. Key wrapping reduces much of the problem of secure key storage to that of protecting the shorter AE keys. Furthermore, leakage of encrypted data is not a security issue.

C. Outline of this work and Our Contributions

We define the *side-channel secure key wrapping* problem and outline the WrapQ key import and export methods. WrapQ performs a simultaneous unwrapping (symmetric decryption) and refreshing of PQC private keys. The technique enables compact storage of (relatively long) PQC secret keys on an untrusted medium and their side-channel secure use.

We describe a real-life implementation of WrapQ for Kyber and Dilithium. This requires an analysis and classification of their Critical Security Parameters (CSPs) so that each variable is appropriately handled. We present data from an FPGA implementation and "TVLA" validation on importing and exporting Kyber and Dilithium keys using WrapQ. The module is found not to leak in 100K traces.

II. Masked Key Wrapping

Most works on side-channel secure implementations of symmetric ciphers (such as AES) focus on protecting the symmetric key; in a standard model, the attacker can observe and even choose both plaintext and ciphertext. For Key Wrapping, we have an additional goal: its "plaintext" (i.e., the wrapped asymmetric key payload) also remains invisible to side-channel measurements.

For lattice-based secret keys, an approach that first decrypts a standard serialization of a secret key and only then splits it into randomized shares (Definition 1) will leak information in repeat observations; even partial information about coefficients can be used to accelerate attacks. One can also consider encrypting the individual masked shares, which significantly increases the size of the key blob. However, when importing the same static key blob multiple times, the decrypted masked key is also static: Not a unique, random representation as required. From the attackers' viewpoint, a secret key in static shares is not much different from an unmasked key; it is just a longer "expanded key". A potential solution would be to write a refreshed, re-encrypted secret key back every time the key is used, but this approach has severe practical disadvantages in addition to a much larger key blob, such as reliability risks.

A. High level interface

WrapQ implements masked Key Wrapping (protection of the confidentiality and integrity of cryptographic keys [21]) for lattice cryptography with a special type of Authenticated Encryption with Associated Data (AEAD) [23] mechanism. An abstract high-level interface for a masked key wrapping and unwrapping is:

$$C \leftarrow \mathsf{WrapQ}(\ [[K]], [[P]], AD\ ) \tag{1}$$

$$\{\ [[P]], \mathsf{FAIL}\ \} \leftarrow \mathsf{WrapQ}^{-1}(\ [[K]], C, AD\ ). \tag{2}$$

Double square brackets $[[\cdot]]$ denote masked variables:

$[[\mathbf{K}]]$	Symmetric key(s) for integrity and confidentiality protection. Supplied as Boolean shares.
$[[\mathbf{P}]]$	Payload: Asymmetric key material to be encrypted. A set of masked (Boolean/Arithmetic) quantities.
AD	Authenticated Associated Data: Public values that only require integrity protection.
C	Resulting wrapped key blob containing encrypted P, authentication information for AD and P, and internal auxiliary information such as nonces.

Each unwrapping call WrapQ^{-1} produces a fresh, randomized masking representation for $[[P]]$ variables, or FAIL in case of authentication (integrity) failure. In addition to standard AEAD security goals, the primitives guarantee that long-term secrets $\mathbf{K}$ or $\mathbf{P}$ do not leak while handling $[[\mathbf{K}]]$ and $[[\mathbf{P}]]$.

III. WrapQ 1.0 Desing Outline

Our solution makes several design choices motivated by its particular use case; a side-channel secure hardware module that implements lattice-based cryptography. It is hardware-oriented and not intended as an "universal" format.

A. Design Choices

1) Key Import and Export: Importing may occur during device start-up or if there is a change of keys. Key export is required when new keys are generated or if KEK changes. Side-channel considerations are equally important in both use cases. The term "import" does not necessarily imply interaction with external devices. The import function simply prepares and loads a private key from static storage to be used by a cryptographic processor.

2) Key Encryption Key: We primarily want to secure the process of local, automatic, unsupervised loading of secret keys for immediate use. For example, some hardware devices may use a device-unique key or a Physically Unclonable Function (PUF) to derive the KEK, with the idea that keys exported to a less trusted storage can only be imported back into the same physical module [24]. Since the main goal is side-channel security, the storage format may be modified to accommodate implementation-specific requirements.

3) Non-Determinism is Preferable: Rogaway and Shrimpton [22] argue that a key wrapping operation should be fully deterministic; the inputs K, P, A fully determine C without randomization. Their motivation is that removing the randomization nonce from C will save some bandwidth. We prioritize side-channel security and observe that randomization helps to eliminate leakage in the export function.

4) Secondary Encryption: WrapQ only encrypts critical portions of the key material. It is a "feature" that algorithm identifiers and the public key hash are unencrypted; this makes it possible to retrieve a matching public key before validating the secret key blob. WrapQ key blobs do not have complete confidentiality properties, such as indistinguishability from random. However, the resulting blob is much safer to handle as critical variables are encrypted; a secondary confidentiality step can use arbitrary mechanisms to re-encrypt it.

5) Not (necessarily) a key interchange format: Export can also occur between devices; sometimes, the term "Key Exchange Key" is used to export a key from one HSM to another or from an on-premises system to the cloud [25]. In such "one-off" manual use cases, side-channel protections may be less critical, and mechanisms such as PKCS #12 [26] can be used (after additional authorization).

B. Masked XOF and Domain Separation

WrapQ uses a masked XOF (extensible output function [27]) as a building block for all of its side-channel secure cryptographic functionality.

Definition 2: An Order-d masked extensible output function $[[h]] \leftarrow \mathsf{XOF}_n([[m]])$ processes an arbitrary-length masked input $[[m]]$ into n-byte output shares $[[h]]$ while maintaining Order-d security (under some applicable definition.)

We construct a non-secret frame header for all XOF inputs from four fixed-length components:

$$\mathit{frame} = (\mathit{ID} \parallel \mathit{DS} \parallel \mathit{ctr} \parallel \mathit{IV}) \tag{3}$$

ID	32-bit identifier for algorithm type, parameter set, authentication frame structure, key blob structure, WrapQ version; all serialization details.
DS	8-bit Domain Separation identifier. This specifies frame purpose: hash, keyed MAC, encryption, etc.
ctr	A 24-bit block index $0, 1, 2, \ldots$ for encrypting multi-block material. Set to 0 for authentication (unless the authentication process is parallelized).
IV	Nonce: a 256-bit Initialization Vector, chosen randomly for the key blob. Its frames share the IV.

The main security property of the frame header is that it creates non-repeating, domain-separated inputs for the XOF.

- For a fixed secret key protecting many key blobs, this is due to the randomization of IV. There is a birthday bound of 2^{128} wrapping operations for a given key.
- Within a key blob (fixed IV, key), frames are made unique thanks to (DS, ctr) being different.
- Across versions. Any functional change in WrapQ serialization requires a new ID. This identifier unambiguously defines the structure of the key blob, the interpretation of the contents, the frame header, etc.

There are predefined domain separation bytes; DS_{hash} and DS_{mac} for authentication (Algorithm 1) and DS_{enc} for encryption/decryption (Algorithms 2 and 3.) Frame headers with these domain separation fields are denoted $\mathit{frame}_{\text{hash}}$, $\mathit{frame}_{\text{mac}}$ and $\mathit{frame}_{\text{enc}}$.

C. Integrity Protection: Masked MAC Computation

Algorithm 1 describes the authentication tag computation process. The authentication tag is always checked before any decryption is performed.

```
Algorithm 1: $T = \mathsf{AuthTag}(\ A, [[K]], ID, ctr, IV\ )$
Input: $A$, Authenticated data, including ciphertext.
Input: $[[K]]$, Message Integrity Key (Boolean masked.)
Input: $ID, ctr, IV$: Used to construct frame headers.
Output: $T$, Resulting authentication tag/code.
1: $h \leftarrow \mathsf{Hash}(\ frame_{\text{hash}} \parallel A\ )$
2: $[[T]] \leftarrow \mathsf{XOF}_{|T|}(\ frame_{\text{mac}} \parallel [[K]] \parallel h\ )$
3: $[[K]] \leftarrow \mathsf{Refresh}([[K]])$
4: return $T = \mathsf{Decode}([[T]])$
```

For performance reasons, we first use a non-masked hash function $\mathsf{Hash}()$ to process A (Step 1), and only use a masked XOF to bind the hash result h with the (masked) authentication key $[[K]]$ and other variables (Step 2.) Furthermore, randomized hashing [28] with a frame header containing the IV is used to make the security of h more resilient to collision attacks. The random prefix IV is included in the *frame* construction (Eq. (3)) and used again in the masked key binding step. It is domain-separated via DS from encryption/decryption frames in case the same $[[K]]$ is used. After this single masked step, $[[K]]$ is refreshed, and the authentication tag $[[T]]$ can be unmasked (collapsed) into T.

Cryptographic security notes. In the terminology of [29], WrapQ is an Encrypt-then-MAC (EtM) scheme; ciphertext is authenticated rather than plaintext. Upon a mismatch between the calculated T' and the tag T, a FAIL is returned – no partial decrypted payload. Since WrapQ is an Authenticated Encryption with Associated Data (AEAD) [23] scheme, A includes data items that do not need to be decrypted in addition to ciphertext C. Unambiguous serialization is used to guarantee domain separation between data items. The ID identifier in *frame* defines the contents and ordering of fixed-length fields in A.

D. *Confidentiality Protection: Encrypting Masked Plaintext*

We use the masked XOF in "counter mode" to encrypt/decrypt data. Data is processed in blocks. For Sponge-based primitives such as SHA3/SHAKE [27] the appropriate block size is related to the "rate" parameter, which depends on the security level. Generally, one wants to minimize the number of permutation invocations. SHAKE256 has a data rate of $(1600 - 2 * 256)/8 = 136$ bytes for each permutation, while SHAKE128 has a 168-byte rate.

Algorithm 2 outlines the process of encrypting a single block; using stream cipher terminology, it uses the masked XOF to produce a block of keystream shares (Step 1), which are exclusive-ored with the plaintext to produce ciphertext (Step 2). Key blocks must be used only once before being refreshed (Step 3.) Plaintext must also be refreshed unless it is discarded (Step 4.) The ciphertext is no longer sensitive, so it can be decoded back into unmasked format (Step 5.)

Algorithm 2: $C = \mathsf{EncBlock}([[P]], [[K]], ID, ctr, IV)$

```
Input: $[[P]]$, Payload block (Boolean masked.)
Input: $[[K]]$, Key Encryption Key (Boolean Masked).
Input: $ID, ctr, IV$: Used to construct header $frame_{\text{enc}}$.
Output: $C$, Resulting ciphertext block.
1: $[[x]] \leftarrow \mathsf{XOF}_{|P|}(\ frame_{\text{enc}} \parallel [[K]]\ )$
2: $[[C]] \leftarrow [[P]] \ \oplus\ [[x]]$          ▷ "Stream cipher."
3: $[[K]] \leftarrow \mathsf{Refresh}([[K]])$
4: $[[P]] \leftarrow \mathsf{Refresh}([[P]])$          ▷ (Unless discarded.)
5: return $C = \mathsf{Decode}([[C]])$
```

Algorithm 3 describes the decryption process, which is also illustrated in Fig. 1. A necessary feature of the block decryption (import) function (Algorithm 3) is that the ciphertext C is first converted into masked encoding (Step 1). The secret cover $[[x]]$ is also in randomized shares (Step 2). Hence decryption occurs in masked form (Step 3), avoiding collapsing $[[P]]$.

Algorithm 3: $[[P]] = \mathsf{DecBlock}(C, [[K]], ID, ctr, IV)$

```
Input: $C$, Ciphertext block.
Input: $[[K]]$, Key Encryption Key (Boolean Masked).
Input: $ID, ctr, IV$: Used to construct header frames.
Output: $[[P]]$, key material payload (Boolean masked.)
1: $[[C]] \leftarrow \mathsf{Encode}(C)$
2: $[[x]] \leftarrow \mathsf{XOF}_{|P|}(\ frame_{\text{enc}} \parallel [[K]]\ )$
3: $[[P]] \leftarrow [[C]] \oplus [[x]]$          ▷ "Strem cipher."
4: $[[K]] \leftarrow \mathsf{Refresh}([[K]])$
5: return $[[P]] = \mathsf{Refresh}([[P]])$
```

Cryptographic security notes. Confidentiality of C follows from the random-indistinguishability and one-wayness of the XOF function (as it would without masking), assuming that the frame identifiers never repeat for the same secret key K.

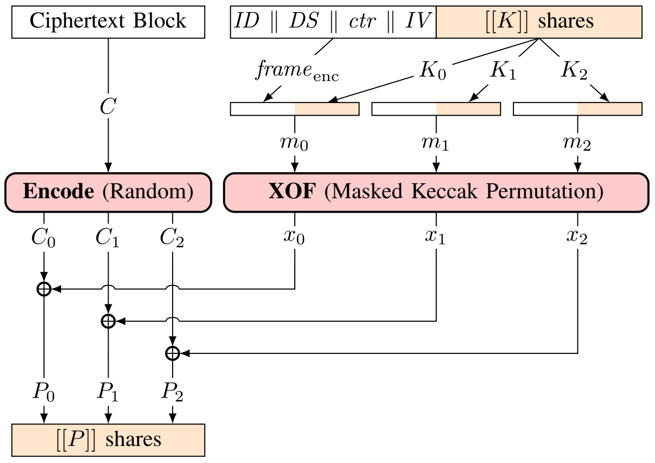

Fig. 1. WrapQ import uses a masked XOF in counter mode to decrypt ciphertext blocks C into randomized Boolean shares $[[P]]$. A masked Keccak Permutation (pictured here with three shares) is also required for masked implementations of Dilithium and Kyber private key operations themselves, so we may assume that this primitive is available.

E. *XOF and Gadget Instantation*

The XOF (Definition 2) is instantiated with a masked Keccak[1600] [27] permutation. Note that a masked SHA3/SHAKE (and hence a masked Keccak permutation) is required to process secret variables in Kyber (G, PRF, KDF) and in Dilithium (H, ExpandS, ExpandMask). Hence this masked primitive can be expected to be available in masked Kyber and Dilithium implementations.

For first-order security one can use the Threshold Implementation (TI) approach of [30], [31] to implement Keccak. Note that a TI Keccak internally uses three shares to obtain first-order resistance, but the technique reduces the amount of randomness required. For higher-order XOF, one can use higher-order TI [32] or the techniques of [33].

For first-order security, we use trivial refresh gadgets $\mathsf{Refresh}([[x]]) = (x_0 \oplus r, x_1 \oplus r)$ with $r = \mathsf{Random}()$ and $\mathsf{Encode}(x) = (x \oplus r, r)$ with $r = \mathsf{Random}()$. For higher-order refresh gadgets, see [34], [35]. The function $\mathsf{Decode}([[x]]) = x_0 \oplus x_1 \oplus \cdots x_d = x$ simply unmasks x.

IV. Kyber and Dilithium Private Keys

Cryptographic module security standards (FIPS 140-3 [36] / ISO 19790 [37]) expect that implementors classify all variables based on the impact of their potential compromise.

- **CSP** (Critical Security Parameter): Security-related information whose disclosure or modification can compromise the security of a cryptographic module. CSPs require both integrity and confidentiality protection.
- **PSP** (Public Security Parameter): Security-related public information whose modification can compromise the security of a cryptographic module. PSPs require only integrity protection (authentication).
- **SSP** (Sensitive Security Parameter): Either a CSP or PSP, or a mixture of both. Essentially all variables in a cryptographic module are SSPs.

The parts of secret key material whose disclosure can compromise cryptographic security are CSPs. Additionally, all internally derived or temporary variables whose leakage will compromise security are CSPs. In the FIPS 140-3 / ISO 19790 context, the (non-invasive) side-channel leakage protection requirement only applies to CSPs, not PSPs.

A. CRYSTALS-Kyber

Table I contains a classification of Kyber key variables. WrapQ encrypts and authenticates masked CSPs $(\mathbf{s}, z)$ and only authenticates the rest of the parameters. For the underlying MLWE problem $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$ the public key consists of $\mathbf{t}$ and the secret key is $\mathbf{s}$. In Kyber, the $\mathbf{A}$ matrix is represented by a SHAKE128 seed ρ that deterministically generates it.

TABLE I
KYBER PUBLIC AND SECRET KEY COMPONENT VARIABLE CLASSIFICATION AND WRAPQ ENCODING FOR SECRET KEYS.

CRYSTALS-Kyber		**Public Key**	**Secret Key**
Standard encoding [1]		$pk = (\hat{\mathbf{t}}, \rho)$	$sk = (\hat{\mathbf{s}}, pk, pkh), z)$
Field	**Size (bits)**	**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 $\mathsf{SHA3}(pk)$.	
z	256	CSP: Fujisaki-Okamoto rejection secret.	

WrapQ Key / Kyber		$sk_{wq} = (ID, T, IV, pkh, z, \mathbf{s})$
Field	**Size (bits)**	**Description**
ID	32	Algorithm and serialization type identifier.
T	256	Authentication tag (Algorithm 1).
IV	256	Random nonce.
pkh	256	Authenticated: Public key hash $\mathsf{SHA3}(pk)$.
z	256	Encrypted: FO Transform secret.
$\mathbf{s}$	$k \times 4 \times 256$	Encrypted: Secret key polynomials.

Kyber standard secret key standard encoding stores $\mathbf{s}$ in the NTT-domain representation $\hat{\mathbf{s}}$. To conserve storage space and also Boolean-to-Arithmetic transformation effort, we instead store normal-domain $\mathbf{s}$, where coefficients are in the range $[-\eta, \eta]$ and would fit into 3 bits (in Kyber, we have $\eta \in \{2, 3\}$, depending on the security level.) However, WrapQ uses 4 bits per coefficient for decoding convenience.

The z variable is a secret quantity used to generate a deterministic response to an invalid ciphertext in the Fujisaki-Okamoto transform. The security proofs assume it to be secret (we implement the entire FO transform as masked); hence, this 256-bit quantity is handled as a Boolean masked secret.

The standard encoding secret key contains a full copy of the public key, likely due to the reference API [38], which did not allow the passing of the public key to signature decapsulation functions. It also contains $H(pk)$, purely as a performance optimization. We also retain and authenticate the $H(pk)$ quantity, but for a different reason: it can be used to authenticate a separately supplied public key.

B. CRYSTALS-Dilithium

Table II contains a classification of Dilithium key variables. WrapQ encrypts CSPs $(K, \mathbf{s}_1, \mathbf{s}_2)$ and only authenticates the rest of the parameters. For the underlying equation $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ the public key is $(\mathbf{A}, \mathbf{t})$ and the secret key is $(\mathbf{s}_1, \mathbf{s}_2)$. In Dilithium, the $\mathbf{A}$ matrix is represented with a short SHAKE128 seed ρ.

Note that Dilithium's public variable $\mathbf{t}$ is split into two halves to minimize the size of the public key, with $\mathbf{t}_1$ placed in the public key and the $\mathbf{t}_0$ in private key (as high bits are sufficient for verification.) The underlying hard problems and security proofs treat the entire $\mathbf{t}$ as a public variable. Hence $\mathbf{t}_0$ is placed within the secret key blob, but as a PSP, there is no need to encrypt it; we just authenticate it.

TABLE II
DILITHIUM PUBLIC AND SECRET KEY COMPONENT VARIABLE CLASSIFICATION AND WRAPQ ENCODING FOR SECRET KEYS.

CRYSTALS-Dilithium		**Public Key**	**Secret Key**
Standard encdoding [2]		$pk = (\rho, \mathbf{t}_1)$	$sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$
Field	**Size (bits)**	**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 $\mathbf{t}$.	

WrapQ / Dilithium		$sk_{wq} = (ID, T, IV, \rho, K, tr, \mathbf{s}_1, \mathbf{s}_2)$
Field	**Size (bits)**	**Description**
ID	32	Algorithm and serialization type identifier.
T	256	Authentication tag (Algorithm 1).
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.

The tr quantity is a 256-bit hash of the public key $tr = \mathsf{SHAKE256}(\rho \parallel \mathsf{t}_1)$. Only the hash is required for signature generation, but since it is an authenticated part of the key blob, we use this quantity to verify that a supplied public key matches the secret key.

The distribution of both $\mathbf{s}_1$ and $\mathbf{s}_2$ is uniform in $[-\eta, +\eta]$. Depending on the security parameters, we have $\eta \in \{2, 4\}$, resulting in 5 or 9 distinct values. While the standard encoding uses bit-packed $d_\eta = \lceil \log_2(2\eta + 1) \rceil$ bits (either 3 or 4). WrapQ uses 4 bits per coefficient in both cases.

The K variable is a secret "seed" value used in deterministic signing (when a signature should be a deterministic, non-randomized function of the private key and the message to be signed). We treat K as a Boolean-masked quantity. However, from a side-channel security perspective, it is preferable to randomize the signing process, in which case K is not used.

V. Parameter Selection and Algorithm Analysis

WrapQ is entirely built from SHA3 / Keccak components; the $\mathsf{XOF}()$ masked permutation (Definition 2) and its non-masked counterpart $\mathsf{Hash}()$ (Section III-C). A straightforward first-order threshold implementation of masked Keccak is roughly three times larger [30] than the unmasked one, and the complexity grows quadratically with the masking order [33]. Other operations in the process are related to mask refreshing or trivial ones such as linear XORs, packing of bits, etc.

Algorithm 1 requires $\lceil (|frame| + |A| + |\text{padding}|)/r \rceil$ unmasked Keccak permutations to compute h with $\mathsf{Hash}()$, where r is the block rate. For SHAKE256, we have $r = 136$ bytes. Additionally, there is a single invocation of masked $\mathsf{XOF}()$ permutation to compute $[[T]]$.

Algorithms 2 and 3 require $\lceil |P|/r \rceil$ invocations of the masked permutation in $\mathsf{XOF}()$. This is also the minimum when computation is organized in a "counter mode" fashion where $[[P]]$ is split into block-sized chunks and ctr is used as an input index. In a way, the encryption process is "format-preserving" as the block size is arbitrary. It is not economical to encrypt blocks substantially smaller than r, as that will result in an increased number of permutations. However, for some parameters, we sacrifice optimality for the logical separation of data items, simplifying implementation.

A. Wrapping Process

In the implementation of the key wrapping operation WrapQ (Eq. (1)) all CSPs are converted to Boolean shares. For $\mathbf{s}$ shares, this involves Inverse-NTT operations since 4-bit packing is used, followed by an Arithmetic-to-Boolean conversion.

After conversions required for the construction of $[[P]]$ we choose a random IV for the entire key blob. The $[[P]]$ input, comprising of CSP data, is divided into blocks and fed to Algorithm 2 to produce ciphertext C.

For Dilithium and Kyber, we process one polynomial at a time since the resulting $(4 \times 256)/8 = 128$-byte block fits the 136-byte data rate of SHAKE256. This has the advantage of "random access" – each secret polynomial can be decrypted only when needed, reducing the RAM requirement. The 4-bit encoding is not optimal of all $[-\eta, +\eta]$ ranges present in these algorithms but is simpler to decode.

The Boolean CSPs (K or z) have $ctr = 0$, block and polynomial CSPs are $1 \leq ctr \leq k$ with Kyber and $1 \leq ctr \leq k+\ell$ with Dilithium. The ciphertext blocks and the PSP data items are then combined into blob A; its serialization is the same as in Tables I and II, although ID, T, IV are omitted from A.

Finally, A is passed to Algorithm 1 to produce T; then the final WrapQ key blob is combined from (ID, T, IV, A).

B. Unwrapping Process

The unwrapping operation WrapQ^{-1} (Eq. (2) starts with consistency checks; we parse ID from the beginning of the blob and see if the size of the blob matches with it. We also check that the pkh (Kyber) or tr (Dilithium) fields match with a hash of the public key that is separately provided.

The rest of unwrapping proceeds in inverse order from wrapping; authentication first, then decryption. We extract IV and A (the remaining part after IV in the blob), and pass those to Algorithm 1 to obtain a check value T'. If we have a mismatch $T \neq T'$ then we return FAIL and abort.

Upon success, we proceed to decrypt CSP fields into payload shares $[[P]]$ using Algorithm 3. The conversion of arithmetic CSPs also follows an inverse route; Boolean-to-Arithmetic conversion, followed by an NTT transform as the implementation keeps secret keys "ready" in the NTT domain.

C. Size Metrics

Table Table III summarizes the sizes of both standard encodings for Kyber and Dilithium keypairs. We observe that even a single randomized arithmetic CSP share would be larger than the WrapQ format. For several parameter sizes, the WrapQ size could be further reduced by encoding the $[-\eta, +\eta]$ coefficients in less than 4 bits, but this would complicate implementation somewhat.

TABLE III
Kyber 3.02 [1] and Dilithium 3.1 [2] "standard serialization" public and secret key sizes in bytes, and the size of the WrapQ secure secret key blob (Tables I and II.)

Algorithm			**Masking**	**Std. Encoding**		**WrapQ**
Parameters	k	ℓ	Per Share	$\|pk\|$	$\|sk\|$	$\|sk_{wq}\|$
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**

The NIST standardization process will likely bring some changes to Kyber 3.02 [1] and Dilithium 3.1 [2]. Furthermore, WrapQ is not necessarily an "interchange" key format; details are subject to change from one instantiation to another.

VI. Implementation and Leakage Assessment

WrapQ grew out of a need to be able to manage Kyber and Dilithium private keys in a commercial side-channel secure hardware module. For leakage testing, the hardware platform was instantiated on an FPGA target. A simple conversion program was written in Python for interoperability testing.

A. FPGA Platform Overview

A first-order implementation of WrapQ was tested with an FPGA module that also implements first-order masked Dilithium and Kyber. We outline its relevant components.

- A simple 64-bit RISC-V control processor.
- Lattice accelerator that can support Kyber and Dilithium $\mathbb{Z}_q$ polynomials and NTT ring arithmetic. The unit can also perform vectorized bit manipulation operations for tasks such as masking conversions (A2B, B2A).
- Ascon-based [39] random mask generator. This is used by the lattice unit for refreshing Boolean and Arithmetic $(\bmod\ q)$ shares. The unit can be continuously seeded from an entropy source.

- A compact first-order, three-share Threshold Implementation [30], [31] of the masked Keccak permutation. See discussion in Section III.
- 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 Algorithm 1).

B. Implementation Overview

The implementation supported all main versions of Kyber and Dilithium (Table III). In the internal representation, the algorithms hold two copies of the variables in Tables I and II either in compressed or uncompressed format. Kyber polynomials were decoded into 16 bits per coefficient for arithmetic operations, while Dilithium polynomials used 32 bits. Hence a two-share unpacked Kyber1024 $[[\mathbf{s}]]$ requires 4 kB of internal storage while Dilithium5 $([[\mathbf{s}_1]], [[\mathbf{s}_2]])$ needs 30 kB. These polynomials are handled using $\pmod q$ arithmetic masking. The 256-bit quantities z (Kyber) and K (Dilithium) were Boolean masked.

Key generation for Kyber (CBD functions) and Dilithium (small-range uniform rejection sampling) was implemented in the bitwise Boolean-masked domain. Hence the wrapping operation does not necessarily require an Arithmetic-to-Boolean (A2B) transform – if invoked from the key generation. The implementation had an A2B function for this case, however.

First-order Boolean-to-Arithmetic (B2A) transform in the unwrapping function was based on the efficient method of Goubin [40], with additional masked manipulation steps to transform secret key quantities from $\pmod{2^n}$ arithmetic masking to $(\bmod q)$ arithmetic masking.

Conversion from the two-share representation to the three-share input required for the TI Keccak was done with the help of the fast masking random generator. Two random vectors were used. Conversion into another direction involves collapsing two of the three shares together (keeping the third intact) and then refreshing the result.

The confidentiality algorithm used in the test target matches the details of Algorithms 2 and 3 in Section Section II-A. Authentication was enabled in the import and export functions, but the tests were performed using a "platform security" parameterization; 128-bit IV and T fields, and a slightly different arrangement of hashes is Algorithm 1.

C. Leakage Assessment: Fixed-vs-Random Experiments

Our methodology broadly follows the ISO/IEC WD 17825:2021(E) "General Testing Procedure," [16, Figure 7] with statistical corrections [15], [41]. This, in turn, was based on Test Vector Leakage Assessment (TVLA) proposed by CRI / Rambus in 2011 [42] and refined in [14], [15].

In TVLA testing, two sets of trace waveforms, A and B, each with L synchronized time points, are compared. The tests we use are of the "Fixed-vs-Random" type, where set A has a CSP (such as a secret key set) to a fixed value, while set B has that CSP set to random values. Welch's t-test is applied to each time point to see if the averages of A and B sets differ significantly. A significant difference is a distinguishing feature between the sets, implying leakage from the CSP.

The first step is determining the required sample size $N = N_A + N_B$ and t-test threshold C from the experiment parameters.

α: Significance level of false positives / type I error.
β: Significance level of false negatives / type II error.

Traditionally ([43]–[45]) a critical value C of ± 4.5 has been used for $L = 1$, which matches an $\alpha < 10^{-5}$ in that case. Since we have long traces (large L), this choice would cause false positives. We adjust the critical value C based on L using the Mini-p procedure from Zhang et al. [15]. Let $\alpha_L = 1-(1-\alpha)^{(1/L)}$ be the adjusted significance level. Since the degrees of freedom are very large, we can approximate using the normal distribution: $C = \mathsf{CDF}^{-1}(1 - \frac{\alpha_L}{2})$.

Older (2016) versions of ISO 17825 [45] set the number of traces to $N = 10,000$ at FIPS 140-3 security level 3 and $N = 100,000$ at the highest level 4. Newer draft versions [16] derive N using an experimentally-derived Cohen's statistical effect size d, an approach suggested in [41]. The $d \in \{0.01, 0.04\}$ values were based on AES key recovery experiments. We don't have experimental key-extraction data to justify a specific d selection, so we collect as many traces as practically possible (in a day or two.) We adopt a best-effort approach to leakage detection and attempt to minimize physical or methodological sources of interference that might negatively impact the leakage assessment process.

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 is not crossed, the test is a PASS.

D. KEK Leakage Testing

A (relatively) straightforward fixed-vs-random test is used in relation to the symmetric Key Encryption Key (KEK) K. This 256-bit secret variable is used for decryption in import (Algorithm 3), encryption in export (Algorithm 3), and to compute integrity check values (Algorithm 1) in both cases.

The test aims to find leakage from the key K itself, and its set-up is similar to "fixed-vs-random key" TVLA tests performed on block ciphers such as AES [16], [44]. Set A has a fixed K, while set B has a random K. Note that the plaintext payload data (i.e., Kyber and Dilithium keys) is randomized in this test; only the symmetric keys are manipulated.

E. CSP Leakage Testing

For fixed-vs-random testing, confidentiality (encryption) is only provided in WrapQ for CSP (actually non-public) variables. Kyber has two CSPs: ring vector $\mathbf{s}$ (decryption key), and FO secret z (Table I) while Dilithium's CSPs are the ring vectors $\mathbf{s}_1, \mathbf{s}_2$ (signing key) and the deterministic seed K (Table II.) All other variables are PSPs (public.)

Since public components are not protected, they would, of course “leak.” In order to capture leakage from these specific CSP variables during the import/export function, we construct *synthetic keys* [46] where CSPs have been randomized, but other components (such as public seed ρ) are unmodified. Such masked keys $[[P]]$ would not be valid for encapsulation/decapsulation or signing/verifying; they are merely artifacts used in side-channel testing of key import and export functions.

F. Trace Acquisition and Results

The experiments were performed with XC7A100T2FTG256 Artix 7 FPGA chip on a ChipWhisperer CW305-A100 board, clocked at 50 Mhz. The processor and coprocessor bitstreams were synthesized with Xilinx Vivado 2021.2. All firmware was in C and complied GCC, under `-Os` size optimization and `-mabi=lp64 -march=rv64imac` architectural flags.

Signal acquisition was performed with Picoscope 6434E oscilloscopes with a 156.25 MHz sampling rate connected to the SMA connectors on the CW305 board. The DUT generated a cycle-precise trigger.

Table IV summarizes the various Fixed-vs-Random tests performed on the implementation. The tests were carried out on all three proposed security levels of Kyber and Dilithium, but we only include graphs for the highest Category 5 proposals, Kyber1024 and Dilithium5.

The functions passed the tests with 100,000 traces. Even though the critical value C has been adjusted for long traces (as discussed above), by looking at the figures referenced in Table IV, we can see that the t values are generally bound at a much smaller range. The target unit also performs side-channel secure Kyber and Dilithium operations, but those tests are out of scope for the present work.

TABLE IV
SUMMARY OF RANDOM-VS-FIXED TESTS ON WRAPQ KEY IMPORT AND EXPORT FUNCTIONS. THE TESTS WERE DESIGNED TO TEST LEAKAGE FROM THE KEK (KEY) AND THE PAYLOAD CSPS.

Function Under Test	Set A	Set B	Both A&B
Kyber Import (Fig. 2)	Fix CSP	Rand CSP	Fix KEK
Kyber Export (Fig. 3)	Fix CSP	Rand CSP	Fix KEK
Kyber Import (Fig. 4)	Fix KEK	Rand KEK	Rand CSP
Kyber Export (Fig. 5)	Fix KEK	Rand KEK	Rand CSP
Dilithium Import (Fig. 6)	Fix CSP	Rand CSP	Fix KEK
Dilithium Export (Fig. 7)	Fix CSP	Rand CSP	Fix KEK
Dilithium Import (Fig. 8)	Fix KEK	Rand KEK	Rand CSP
Dilithium Export (Fig. 9)	Fix KEK	Rand KEK	Rand CSP

VII. CONCLUSIONS AND FUTURE WORK

When building side-channel secure implementations of asymmetric algorithms, it is easy to sidestep the key management problem. Academic works have generally focused on protecting the private key operations, assuming that refreshed key shares can be kept in working memory. However, many real-life devices do not have the option of having refreshable non-volatile memory for keys.

WrapQ is a method for handling masked secret key material between a hardware security module and potentially untrusted storage. Its encryption, decryption, and authentication modes can manage wrapped key material in masked format, significantly increasing resilience to side-channel attacks.

We describe a version of WrapQ that supports CRYSTALS-Kyber 3.02 Key Encapsulation Mechanism and CRYSTALS-Dilithium 3.1 signature scheme. The implementation leverages a masked implementation of FIPS 202 / SHAKE256 (the Keccak permutation) in a mode that prevents leakage even when an attacker can acquire thousands of side-channel measurements from importing and exporting secret keys and also access the resulting WrapQ data itself.

We have performed a TVLA leakage assessment and validation of a WrapQ implementation for Kyber and Dilithium. The leakage of payload CSP variables and the KEK (key-encryption key) was tested. Import and export functions for both algorithms pass TVLA testing for up to 100K traces.

Our experimental work has focused on first-order protections. However, the file format works also with higher-order masking. As the masking order grows, so does the complexity of all nonlinear operations and refresh gadgets. We acknowledge that the construction of higher-order gadgets for WrapQ (Section II-A) requires further investigation.

ACKNOWLEDGMENTS

The author wishes to thank Ben Marshall for running the leakage assessment tests and Oussama Danba and Kevin Law for helping to make the FPGA test target operational. Further thanks to Thomas Prest, Rafael del Pino, and Melissa Rossi for the technical and theoretical discussions. The author is to blame for all errors and omissions.

APPENDIX

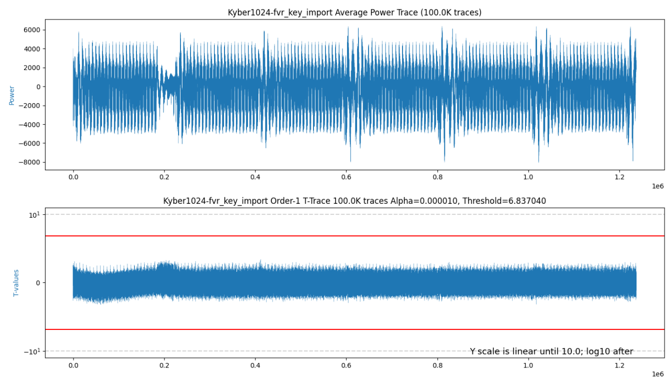

Fig. 2. Kyber1024 WrapQ Import Random-vs-Fixed CSP, 100K Traces.

REFERENCES

[1] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, “CRYSTALS-kyber: Algorithm specifications and supporting documentation (version 3.02),” NIST PQC Project, 3rd Round Submission Update, August 2021. [Online]. Available: https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf

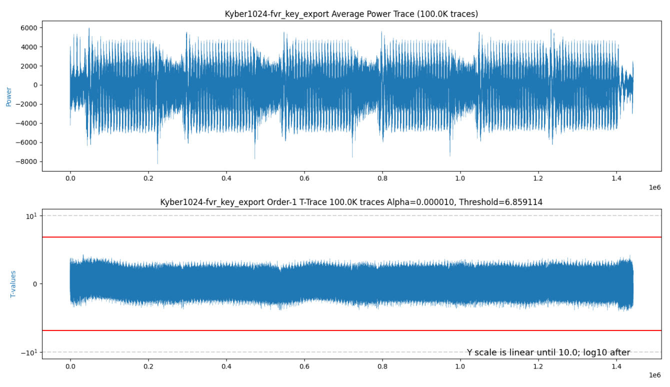

Fig. 3. Kyber1024 WrapQ Export Random-vs-Fixed CSP, 100K Traces.

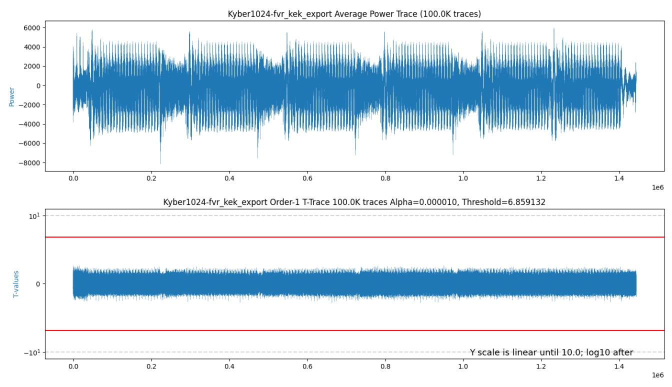

Fig. 5. Kyber1024 WrapQ Export Random-vs-Fixed KEK, 100K Traces.

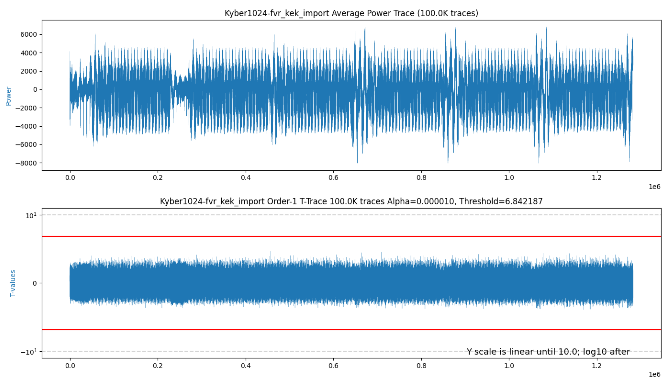

Fig. 4. Kyber1024 WrapQ Import Random-vs-Fixed KEK, 100K Traces.

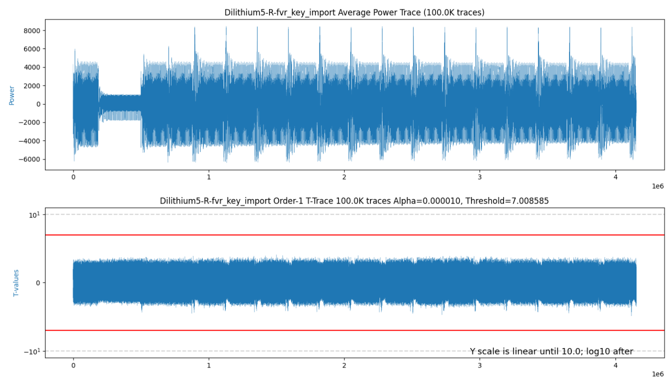

Fig. 6. Dilithium5 WrapQ Import Random-vs-Fixed CSP, 100K Traces.

[2] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-dilithium: Algorithm specifications and supporting documentation (version 3.1)," NIST PQC Project, 3rd Round Submission Update, February 2021. [Online]. Available: https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf

[3] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, "Status report on the third round of the NIST post-quantum cryptography standardization process," NISTIR 8413-upd1, National Institute of Standards and Technology, Interagency or Internal Report, September 2022. [Online]. Available: https://csrc.nist.gov/publications/detail/nistir/8413/final

[4] NSA, "Announcing the commercial national security algorithm suite 2.0," National Security Agency, Cybersecurity Advisory, September 2022. [Online]. Available: https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF

[5] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, ser. Lecture Notes in Computer Science, N. Koblitz, Ed., vol. 1109. Springer, 1996, pp. 104–113.

[6] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, ser. Lecture Notes in Computer Science, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 388–397.

[7] J. Quisquater and D. Samyde, "Electromagnetic analysis (EMA): measures and counter-measures for smart cards," in *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*, ser. Lecture Notes in Computer Science, I. Attali and T. P. Jensen, Eds., vol. 2140. Springer, 2001, pp. 200–210.

[8] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, ser. Lecture Notes in Computer Science, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 398–412.

[9] Y. Ishai, A. Sahai, and D. A. Wagner, "Private circuits: Securing hardware against probing attacks," in *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, ser. Lecture Notes in Computer Science, D. Boneh, Ed., vol. 2729. Springer, 2003, pp. 463–481.

[10] M. Rivain and E. Prouff, "Provably secure higher-order masking of AES," in *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, ser. Lecture Notes in Computer Science, S. Mangard and F. Standaert, Eds., vol. 6225. Springer, 2010, pp. 413–427.

[11] E. Prouff and M. Rivain, "Masking against side-channel attacks: A formal security proof," in *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, ser. Lecture Notes in Computer Science, T. Johansson and P. Q. Nguyen, Eds., vol. 7881. Springer, 2013, pp. 142–159.

[12] A. Duc, S. Dziembowski, and S. Faust, "Unifying leakage models: From probing attacks to noisy leakage," *J. Cryptol.*, vol. 32, no. 1, pp. 151–177, 2019.

[13] M. Alioto, S. Bongiovanni, M. Djukanovic, G. Scotti, and A. Trifiletti, "Effectiveness of leakage power analysis attacks on DPA-resistant logic styles under process variations," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 2, pp. 429–442, 2014.

[14] T. Schneider and A. Moradi, "Leakage assessment methodology - A clear roadmap for side-channel evaluations," in *Cryptographic Hardware*

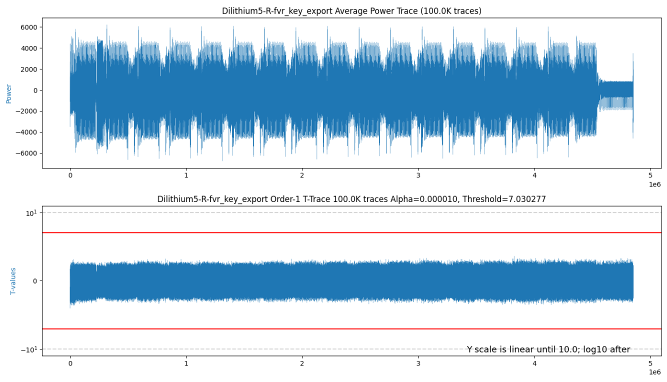

Fig. 7. Dilithium5 WrapQ Export Random-vs-Fixed CSP, 100K Traces.

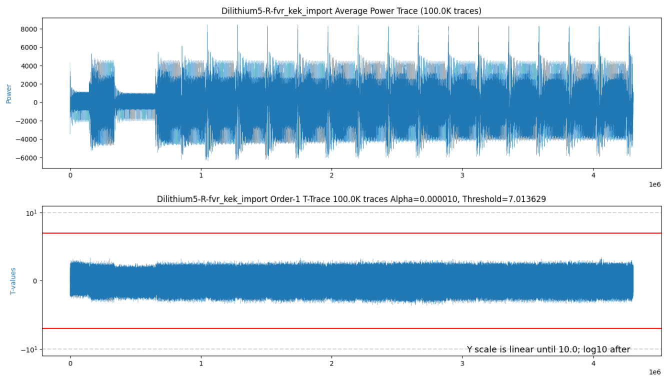

Fig. 8. Dilithium5 WrapQ Import Random-vs-Fixed KEK, 100K Traces.

and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings, ser. Lecture Notes in Computer Science, T. Güneysu and H. Handschuh, Eds., vol. 9293. Springer, 2015, pp. 495–513.

[15] A. A. Ding, L. Zhang, F. Durvaux, F. Standaert, and Y. Fei, "Towards sound and optimal leakage detection procedure," in *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*, ser. Lecture Notes in Computer Science, T. Eisenbarth and Y. Teglia, Eds., vol. 10728. Springer, 2017, pp. 105–122.

[16] ISO, "Information technology – security techniques – testing methods for the mitigation of non-invasive attack classes against cryptographic modules," International Organization for Standardization, Working Draft ISO/IEC WD 17825:2021(E), 2021.

[17] G. Barthe, S. Belaïd, T. Espitau, P. Fouque, B. Grégoire, M. Rossi, and M. Tibouchi, "Masking the GLP lattice-based signature scheme at any order," in *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, ser. Lecture Notes in Computer Science, J. B. Nielsen and V. Rijmen, Eds., vol. 10821. Springer, 2018, pp. 354–384. [Online]. Available: https://eprint.iacr.org/2018/381

[18] V. Migliore, B. Gérard, M. Tibouchi, and P. Fouque, "Masking Dilithium - efficient implementation and side-channel evaluation," in *Applied Cryptography and Network Security - 17th International Conference, ACNS 2019, Bogota, Colombia, June 5-7, 2019, Proceedings*, ser. Lecture Notes in Computer Science, R. H. Deng, V. Gauthier-Umaña, M. Ochoa, and M. Yung, Eds., vol. 11464. Springer, 2019, pp. 344–362.

[19] J. W. Bos, M. Gourjon, J. Renes, T. Schneider, and C. van Vredendaal, "Masking kyber: First- and higher-order implementations," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 4, pp. 173–214, 2021.

[20] D. Heinz, M. J. Kannwischer, G. Land, T. Pöppelmann, P. Schwabe, and D. Sprenkels, "First-order masked Kyber on ARM Cortex-

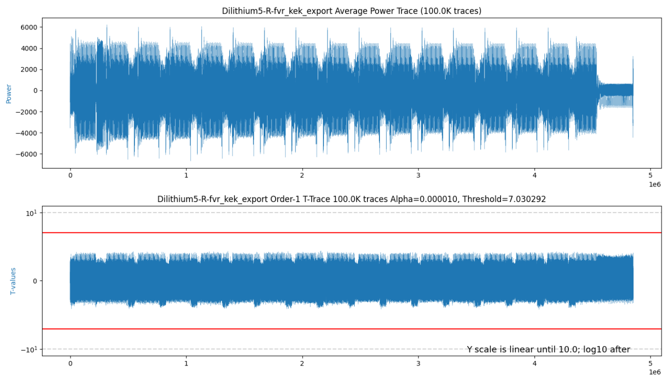

Fig. 9. Dilithium5 WrapQ Export Random-vs-Fixed KEK, 100K Traces.

M4," IACR ePrint 2022/058, 2022. [Online]. Available: https://eprint.iacr.org/2022/058

[21] M. Dworkin, "Recommendation for block cipher modes of operation: Methods for key wrapping," NIST Special Publication SP 800-38F, December 2012.

[22] P. Rogaway and T. Shrimpton, "A provable-security treatment of the key-wrap problem," in *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, ser. Lecture Notes in Computer Science, S. Vaudenay, Ed., vol. 4004. Springer, 2006, pp. 373–390.

[23] P. Rogaway, "Authenticated-encryption with associated-data," in *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, V. Atluri, Ed. ACM, 2002, pp. 98–107. [Online]. Available: http://dl.acm.org/citation.cfm?id=586110

[24] N. Menhorn, "External secure storage using the PUF," Application Note: Zynq UltraScale+ Devices, XAPP1333 (v1.2), April 2022. [Online]. Available: https://docs.xilinx.com/r/en-US/xapp1333-external-storage-puf

[25] Microsoft, "Bring your own key specification," Online documentation: Azure Key Vault / Microsoft Learn. Accessed 2022-Oct-12, February 2022. [Online]. Available: https://learn.microsoft.com/en-us/azure/key-vault/keys/byok-specification

[26] K. M. Moriarty, M. Nystrom, S. Parkinson, A. Rusch, and M. Scott, "PKCS #12: Personal information exchange syntax v1.1," IETF RFC 7292, July 2014.

[27] NIST, "SHA-3 standard: Permutation-based hash and extendable-output functions," Federal Information Processing Standards Publication FIPS 202, August 2015.

[28] S. Halevi and H. Krawczyk, "Strengthening digital signatures via randomized hashing," in *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, ser. Lecture Notes in Computer Science, C. Dwork, Ed., vol. 4117. Springer, 2006, pp. 41–59.

[29] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," *J. Cryptol.*, vol. 21, no. 4, pp. 469–491, 2008.

[30] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Building power analysis resistant implementations of Keccak," August 2010. [Online]. Available: https://csrc.nist.gov/Events/2010/The-Second-SHA-3-Candidate-Conference

[31] J. Daemen, "Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing," in *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, ser. Lecture Notes in Computer Science, W. Fischer and N. Homma, Eds., vol. 10529. Springer, 2017, pp. 137–153.

[32] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, "Higher-order threshold implementations," in *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, ser. Lecture Notes

in Computer Science, P. Sarkar and T. Iwata, Eds., vol. 8874. Springer, 2014, pp. 326–343.

[33] G. Barthe, S. Belaïd, F. Dupressoir, P. Fouque, B. Grégoire, P. Strub, and R. Zucchini, "Strong non-interference and type-directed higher-order masking," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 116–129. [Online]. Available: http://dl.acm.org/citation.cfm?id=2976749

[34] A. Mathieu-Mahias, "Securisation of implementations of cryptographic algorithms in the context of embedded systems," Ph.D. dissertation, Université Paris-Saclay, 2021. [Online]. Available: https://tel.archives-ouvertes.fr/tel-03537322

[35] D. Goudarzi, T. Prest, M. Rivain, and D. Vergnaud, "Probing security through input-output separation and revisited quasilinear masking," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 3, pp. 599–640, 2021.

[36] NIST, "Security requirements for cryptographic modules," Federal Information Processing Standards Publication FIPS 140-3, March 2019.

[37] ISO, "Information technology – security techniques – security requirements for cryptographic modules," International Organization for Standardization, Standard ISO/IEC 19790:2012(E), 2012.

[38] NIST, "PQC – API notes," Example Files, Official Call for Proposals, National Institute for Standards and Technology, September 2017. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/example-files/api-notes.pdf

[39] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2," Submission to NIST (Lightweight Cryptography Project), May 2021. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf

[40] L. Goubin, "A sound method for switching between boolean and arithmetic masking," in *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, ser. Lecture Notes in Computer Science, Ç. K. Koç, D. Naccache, and C. Paar, Eds., vol. 2162. Springer, 2001, pp. 3–15.

[41] C. Whitnall and E. Oswald, "A critical analysis of ISO 17825 ('testing methods for the mitigation of non-invasive attack classes against cryptographic modules')," in *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, ser. Lecture Notes in Computer Science, S. D. Galbraith and S. Moriai, Eds., vol. 11923. Springer, 2019, pp. 256–284.

[42] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for sidechannel resistance validation," CMVP & AIST Non-Invasive Attack Testing Workshop (NIAT 2011), September 2011. [Online]. Available: https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/08_goodwill.pdf

[43] G. Becker, J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, T. Kouzminov, A. Leiserson, M. Marson, P. Rohatgi, and S. Saab, "Test vector leakage assessment (TVLA) methodology in practice," 2013, presented at International Cryptography Module Conference – ICMC 2013.

[44] Rambus, "Test vector leakage assessment (TVLA) derived test requirements (DTR) with AES," Rambus CRI Technical Note, February 2015. [Online]. Available: https://www.rambus.com/wp-content/uploads/2015/08/TVLA-DTR-with-AES.pdf

[45] ISO, "Information technology – security techniques – testing methods for the mitigation of non-invasive attack classes against cryptographic modules," International Organization for Standardization, Standard ISO/IEC 17825:2016, 2016. [Online]. Available: https://www.iso.org/standard/82422.html

[46] M.-J. O. Saarinen, "WiP: Applicability of ISO standard side-channel leakage tests to NIST post-quantum cryptography," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST). June 27–30, 2022 Washington DC, USA*. IEEE, 2022, pp. 69–72. [Online]. Available: https://eprint.iacr.org/2022/229

[47] M. J. Wiener, Ed., *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, ser. Lecture Notes in Computer Science, vol. 1666. Springer, 1999.