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

Markku-Juhani O. Saarinen[0000−0002−2555−235X]

PQShield Ltd. Oxford, UK
mjos@pqshield.com

Abstract. Transition to PQC brings complex challenges to builders of secure cryptographic hardware. PQC keys usually need to be stored off-module and protected via symmetric encryption and message authentication codes. Only a short, symmetric Key-Encrypting Key (KEK) can be managed on-chip with trusted non-volatile key storage. For secure use, PQC key material is handled in masked format; as randomized shares. Due to the masked encoding of the key material, algorithm-specific techniques are needed to protect the side-channel security of the PQC key import and export processes.

In this work, we study key handling techniques used in real-life secure Kyber and Dilithium hardware. We describe WrapQ, a masking-friendly key-wrapping mechanism designed for lattice cryptography. On a high level, WrapQ protects the integrity and confidentiality of key material and allows keys to be stored outside the main security boundary of the module. Significantly, its wrapping and unwrapping processes minimize side-channel leakage from the KEK integrity/authentication keys as well as the masked Kyber or Dilithium key material payload.

We demonstrate that masked Kyber or Dilithium private keys can be managed in a leakage-free fashion from a compact WrapQ format without updating its encoding in non-volatile (or read-only) 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 ISO 17825 / TVLA-type leakage assessment.

Keywords: Side-Channel Security · Masking Countermeasures · Key Wrapping · Kyber · Dilithium

1 Introduction

With the standardization of CRYSTALS suite algorithms Kyber [3] and Dilithium [5] as the preferred NIST Post-Quantum Cryptography (PQC) methods for key agreement and digital signatures [1], 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 [30].

1.1 Side-Channel Countermeasures for Lattice Cryptography

Kyber and Dilithium are gradually replacing older RSA and Elliptic Curve Cryptography in systems where it is a requirement that a device (such as a mobile phone, authentication token, or a smart card) does not leak sensitive information even if an adversary has physical access to the device or its close proximity. A related (System-on-Chip) PQC use case is *platform security*, where cryptographic signatures and protocols are used to protect system firmware/bitstream integrity and 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 considerations are Timing Attacks (TA) [22], Differential Power Analysis (DPA) [23], and Differential Electromagnetic Analysis (DEMA) [33]. Almost any implementation can be rapidly attacked with these methods if appropriate countermeasures are not in place. Mitigations against TA, DPA, DEMA non-invasive attacks are required for FIPS 140-3/ISO 19790 certification [20, 21] at higher levels.

Masking [12] has 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 be represented 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 in a way that avoids directly combining the shares, thereby limiting leakage. Arbitrary circuits can be transformed to use masking with quadratic $O(d^2)$ overhead [19]. It has been shown that the amount of side channel information required to learn x or y grows exponentially in relation to masking order d [12]. Hence masking is asymptotically efficient.

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 [19] and Prouff-Rivain noisy leakage model [32,35]. 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. The SNI (Strong Non-Interference) [6] property allows better composability.

In addition to theoretical soundness, an essential advantage of masking countermeasures for PQC is that they are generally less dependent on the physical details of the implementation when compared to logic-level techniques such as

dual-rail countermeasures [2]. However, it is essential to verify the leakage properties experimentally. There are standard approaches to physical leakage assessment [14, 21, 38].

1.2 Sensitivity Analysis: Private Keys and Secret Variables

Side-channel leakage can be exploited in any component that handles secret key material. In a broader sensitivity analysis (such as the one performed on Dilithium in [18]), it is apparent that the key management processes must meet the same security requirements as the key generation or private key operations.

Often the "zeroth" step of an asymmetric private-key operation such as signing or decapsulation is "load private key." In SCA-protected implementation, the private key clearly can't really 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. However, this is not practical with ROM or Flash keys. Furthermore, masked representations significantly increase the secret key storage requirement. Secure, non-volatile key storage is an expensive resource. Standard-format Kyber1024 private keys are 25,344 bits, while Dilithium5 secret keys are 38,912 bits (Table 3.) 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 [15, 37] is 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 management to that of protecting (or deriving) the shorter, symmetric AE wrapping key(s). However, the standard AES-based techniques can't easily protect the plaintext payload from side-channel leakage, just the AES/KEK key itself.

1.3 Outline of this work and Our Contributions

There exists a body of work discussing the side-channel protection of lattice cryptography schemes, including GLP [7], Dilithium [4, 26] and Kyber [11, 18]. The key management issue has not been addressed previously; keys have been assumed to be immediately available in a dynamically refreshable masked form.

We define the *side-channel secure key wrapping* problem and outline the WrapQ approach. Here the key import function performs a simultaneous unwrapping (symmetric decryption) and refreshing of PQC private key masks. No write-back of refreshed keys is necessary. WrapQ enables compact storage of 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. Side-channel security requires a sensitivity analysis and classification of these algorithms' Critical Security Parameters (CSPs) so that each variable in the secret key is appropriately handled. We then describe an FPGA implementation and perform a leakage assessment of masked Kyber and Dilithium key import and key export functions. No leakage was found in 100K traces.

2 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) must also remain 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 always the same: Not a unique, random representation as required for masking security.

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.

2.1 High level interface

WrapQ implements masked Key Wrapping (protection of the confidentiality and integrity of cryptographic keys [15]) for lattice cryptography with a special type of Authenticated Encryption with Associated Data (AEAD) [36] 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:

- **[[K]]** **Key Encrypting Key (KEK):** Symmetric secret for integrity and confidentiality (short, Boolean-masked key.)
- **[[P]]** **Payload:** Asymmetric key material to be encrypted (a set of masked arithmetic and Boolean variables.)
- **AD** **Authenticated Associated Data:** Additional elements that only require integrity protection (e.g. the public key.)
- **C** **Wrapped key:** 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 K or P do not leak while operating WrapQ and WrapQ^{-1} thousands of times.

3 WrapQ 1.0 Design 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 is not intended as an "universal" format.

3.1 Design Choices

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.

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.

Non-Determinism is Preferable Rogaway and Shrimpton [37] 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.

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.

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 [27] can be used (after additional authorization).

3.2 Masked XOF and Domain Separation

WrapQ uses a masked XOF (extensible output function [28]) 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.)*

The XOF (Definition 2) is instantiated with a masked Keccak[1600] [28] 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 primitive can be expected to be available in masked Kyber and Dilithium implementations.

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

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

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 $frame_{\text{hash}}$, $frame_{\text{mac}}$ and $frame_{\text{enc}}$.

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 $frame_{DS}$ 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** $T = \mathsf{Decode}([[T]])$

3.3 Integrity Protection: Masked MAC Computation

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

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 [17] 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 [9], 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) [36] scheme, input tuple 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 and all other variables.

3.4 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 [28], 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 $frame_{enc}$.
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. Algorithms 2 and 3 are analogous to counter-mode (CTR) encryption/decryption, except that the payload $[[P]]$ is masked. Confidentiality of ciphertext 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 $[[K]]$.

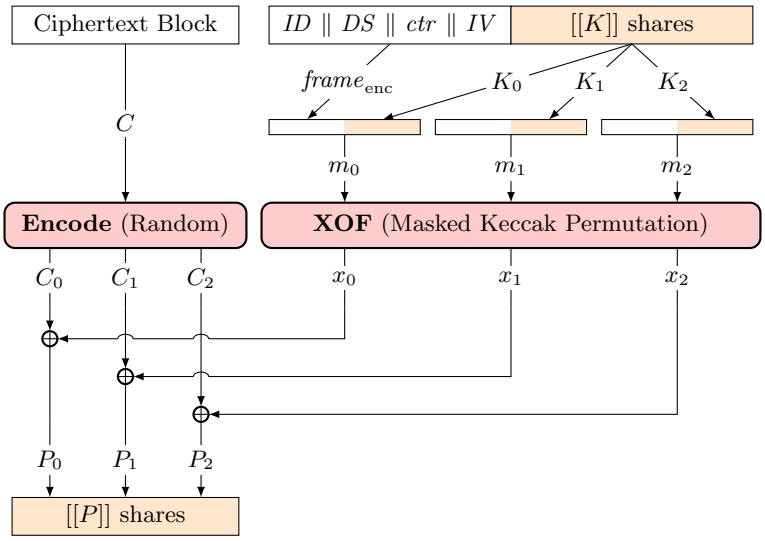


Fig. 1. The WrapQ^{-1} key import function uses a masked XOF in counter mode to decrypt ciphertext blocks C into randomized Boolean shares $[[P]]$. The Keccak Permutation (pictured here with three shares) exists in secure implementations of Dilithium and Kyber; WrapQ just reuses the component.

4 Kyber and Dilithium Private Keys

Cryptographic module security standards (FIPS 140-3 [29] / ISO 19790 [20]) 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 [20, Sect 7.8], not PSPs.

Table 1. Kyber public and secret key components: Variable sensitivity classification and WrapQ encoding for Kyber secret keys.

CRYSTALS-Kyber		**Public Key**	**Secret Key**
Standard encoding [3]:		$pk = (\hat{\mathbf{t}}, \rho)$	$sk = (\hat{\mathbf{s}}, pk, pkh), z)$
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 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 (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.	

4.1 CRYSTALS-Kyber

Table 1 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 $(\mathbf{A}, \mathbf{t})$ and the secret key is $\mathbf{s}$ (ephemeral error $\mathbf{e}$ is not stored.) In Kyber, the $\mathbf{A}$ matrix is represented by a SHAKE128 seed ρ that deterministically generates it.

Kyber standard secret key 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 four bits per coefficient for Boolean masking conversion 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.

In standard encoding, the Kyber secret key contains a full copy of the public key. 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.

Table 2. Dilithium public and secret key components: Variable sensitivity classification and WrapQ encoding for Dilithium secret keys.

CRYSTALS-Dilithium		**Public Key**	**Secret Key**
Standard encoding [5]		$pk = (\rho, \mathbf{t}_1)$	$sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$
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 Secret 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 (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.	
$\mathbf{s}_2$f	$k \times 4 \times 256$	Encrypted: Secret vector 2.	

4.2 CRYSTALS-Dilithium

Table 2 contains a classification of Dilithium key variables. WrapQ encrypts and authenticates masked CSPs $(K, \mathbf{s}_1, \mathbf{s}_2)$ and only authenticates the rest of the parameters. In the underlying equation $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$, variables $(\mathbf{A}, \mathbf{t})$ are public and $(\mathbf{s}_1, \mathbf{s}_2)$ are secret. The $\mathbf{A}$ matrix is expanded from 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.) However, from a cryptanalytic viewpoint, the entire $\mathbf{t}$ is a public variable. Hence $\mathbf{t}_0$ is placed within the secret key blob but as a PSP. There is no need to encrypt $\mathbf{t}_0$; we just authenticate it.

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 (as a collision-resilient message processing). Since tr is an authenticated part of the key blob, we also use this quantity to verify that a separately supplied public key is valid.

* The size of tr is 256 bits in Dilithium 3.1 [5]. It may change to 512 bits in a future revision of Dilithium [31].

The distribution of both $\mathbf{s}_1$ and $\mathbf{s}_2$ is uniform in $[-\eta, +\eta]$. Depending on the security parameters, we have $\eta \in \{2, 4\}$. While the standard encoding uses $d_\eta = \lceil \log_2(2\eta + 1) \rceil$ bits (either 3 or 4), WrapQ uses 4 Boolean masked bits per coefficient with all security parametrizations.

The K variable is a secret "seed" value used in deterministic signing (making the signature a deterministic, non-randomized function of the private key and the message to be signed). We treat K as a 256-bit 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.

5 Parameter Selection and Algorithm Analysis

Cryptography in WrapQ is entirely built from SHA3/SHAKE (FIPS 202 [28]) components, which in turn are based on the Keccak permutation. The $\mathsf{XOF}()$ function (Definition 2) uses a masked version while $\mathsf{Hash}()$ (Section 3.3) is non-masked. A straightforward first-order threshold implementation of masked Keccak is roughly three times larger [10] than the unmasked one, and the complexity grows quadratically with the masking order [6]. Other operations in the process are related to mask refreshing or trivial ones such as linear XORs, packing of bits, etc.

Algorithm 1 requires $\lceil (|\mathit{frame}| + |A| + |\text{padding}|)/r \rceil$ 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. It is not economical to encrypt blocks substantially smaller than r, as that will result in an increased number of permutations and slower speed. However, for some parameters, we sacrifice optimality for the logical separation of data items, simplifying implementation.

5.1 Wrapping Process

In the implementation of the key wrapping operation WrapQ (Eq. (1)), all CSPs are converted to Boolean shares (Tables 1 and 2). For internal secret $[[\hat{\mathbf{s}}]]$ shares, this involves Inverse-NTT operations to $[[\mathbf{s}]]$ 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 can 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; their serialization is the same as given in Tables 1 and 2, although ID, T, IV are omitted.

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

5.2 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 $T \neq T'$, 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.

5.3 Size Metrics

Table 3 summarizes the sizes of both standard encodings for Kyber and Dilithium keypairs. We observe that each randomized arithmetic CSP share would be larger than the WrapQ format (even if packed to $\lceil \log_2 q \rceil$ bits per coefficient). 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 the implementation.

Note that the NIST standardization process will likely bring some changes to Kyber 3.02 [3] and Dilithium 3.1 [5].

6 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 secret key conversion program was written in Python for interoperability testing.

Table 3. The size of a WrapQ secret key (Tables 1 and 2) does not depend on the masking order. Each individual internal (unpacked) masking share is larger, as are Kyber's "standard serialization" secret keys due to a lack of bit packing.

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**

6.1 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 low-area 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 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 [10, 13] of the masked Keccak permutation. See discussion in Section 3.
- 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).

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 $r = \mathsf{Random}()$. The function $\mathsf{Decode}([[x]]) = x_0 \oplus x_1 \oplus \cdots x_d = x$ simply unmasks x.

6.2 Implementation Overview

The implementation supported all main versions of Kyber and Dilithium (Table 3). In the internal representation, the algorithms hold two copies of the secret CSP variables in Tables 1 and 2 either in compressed or uncompressed format. Kyber polynomials are manipulated at 16 bits per coefficient for arithmetic operations, while Dilithium polynomials use 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 ($\bmod q$) arithmetic masking. The 256-bit quantities z (Kyber) and K (Dilithium) were Boolean masked in the internal representation.

The confidentiality algorithm used in the test target matches the details of Algorithms 2 and 3 in Section Section 2.1. 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.

6.3 Leakage Assessment: Fixed-vs-Random Experiments

Our methodology broadly follows the ISO/IEC WD 17825:2021(E) "General Testing Procedure," [21, Figure 7] with statistical corrections. This, in turn, was based on Test Vector Leakage Assessment (TVLA) proposed by CRI / Rambus in 2011 [16] and refined in [14, 38].

Traditionally a critical value C of ± 4.5 has been used for $L = 1$, which matches an $\alpha < 10^{-5}$ in that case [8, 34]. 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. [14]. 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: $C = \mathsf{CDF}^{-1}(1 - \frac{\alpha_L}{2})$.

KEK Leakage Testing. 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 [21, 34]. 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.

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 1) while Dilithium's CSPs are the ring vectors $\mathbf{s}_1, \mathbf{s}_2$ (signing key) and the deterministic seed K (Table 2.) All other variables are PSPs (public.)

6.4 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. The C language firmware was with 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 4 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 due to space constraints, we only include graphs for the (highest) Category 5 versions, Kyber1024 and Dilithium5.

Table 4. Summary of Random-vs-Fixed tests on WrapQ key import and export functions. The tests were designed to test leakage from both the KEK (Key-Encrypting Key) and the payload CSPs (PQC Secret Keys.) See traces in Fig. 2.

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

The functions passed the tests with 100,000 traces. Even though the critical value C has been adjusted for long traces (as discussed above), from Fig. 2, 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 (key generation, signatures, encapsulation, decapsulation), but those tests are out of scope for the present work.

7 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 detail 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. The size of the WrapQ secret key is independent of the masking order and is often even smaller than the standard encoding.

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.

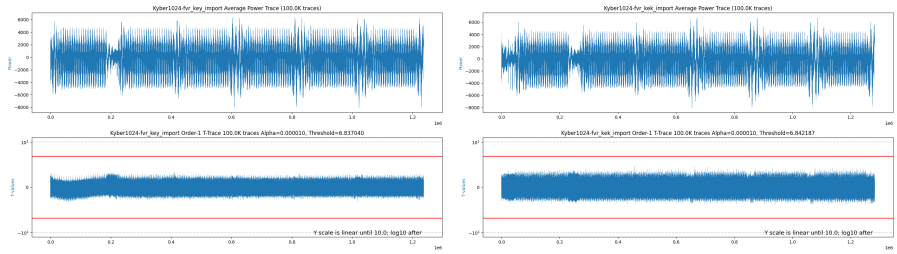
Our experimental work has focused on first-order protections. However, the file format works also with higher-order masking. As the masking order grows, so does the complexity of all nonlinear operations and refresh gadgets. We acknowledge that the construction of higher-order gadgets for WrapQ (Section 2.1) requires further investigation. Furthermore, the formal SNI security of the gadgets remains to be shown.

Acknowledgments

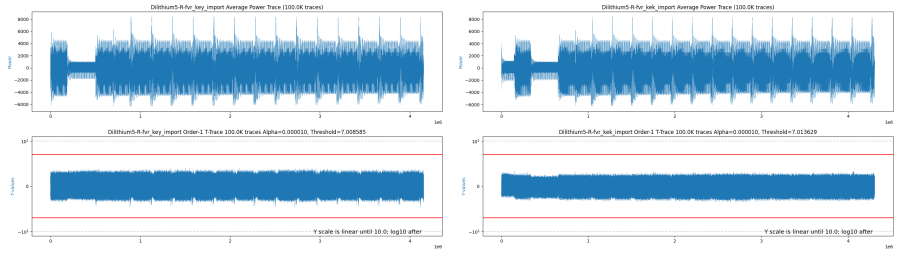
The author wishes to thank Ben Marshall for running the leakage assessment tests and Oussama Danba and Kevin Law for helping to make the FPGA test target operational. Further thanks to Thomas Prest, Rafael del Pino, and Melissa Rossi for the technical and theoretical discussions. The author is to blame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

References

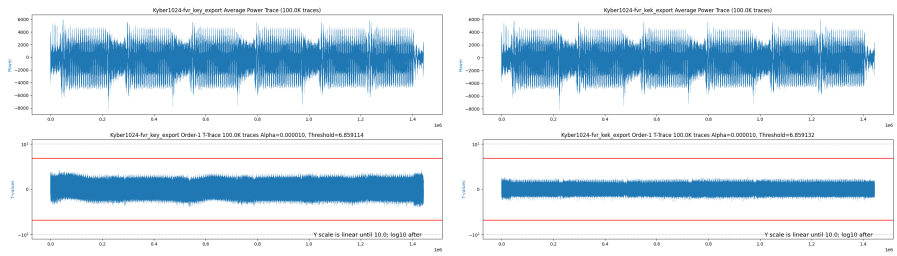
1. Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Smith-Tone, D.: Status report on the third round of the NIST post-quantum cryptography standardization process. Interagency or internal report, National Institute of Standards and Technology (September 2022). https://doi.org/10.6028/NIST.IR.8413-upd1, https://csrc.nist.gov/publications/detail/nistir/8413/final
2. Alioto, M., Bongiovanni, S., Djukanovic, M., Scotti, G., Trifiletti, A.: Effectiveness of leakage power analysis attacks on DPA-resistant logic styles under process variations. IEEE Transactions on Circuits and Systems I: Regular Papers **61**(2), 429–442 (2014). https://doi.org/10.1109/TCSI.2013.2278350
3. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber: Algorithm specifications and supporting documentation (version 3.02). NIST PQC Project, 3rd Round Submission Update (August 2021), https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf
4. Azouaoui, M., Bronchain, O., Cassiers, G., Hoffmann, C., Kuzovkova, Y., Renes, J., Schönauer, M., Schneider, T., Standaert, F.X., van Vredendaal, C.: Leveling Dilithium against leakage: Revisited sensitivity analysis and improved implementations. Cryptology ePrint Archive, Paper 2022/1406 (2022), https://eprint.iacr.org/2022/1406, fourth PQC Standardization Conference, NIST (Virtual) 29 Nov – 1 Dec 2022
5. Bai, S., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: Algorithm specifications and supporting documentation (version 3.1). NIST PQC Project, 3rd Round Submission Update (February 2021), https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf
6. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security,



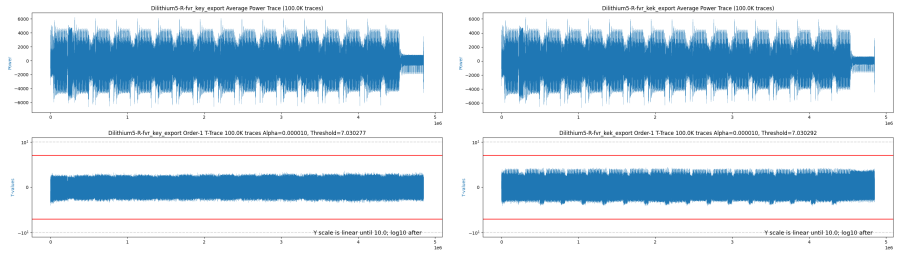
Kyber1024 WrapQ Key Import Random-vs-Fixed CSP (#1 left), KEK (#2 right).



Dilithium5 WrapQ Key Import Random-vs-Fixed CSP (#3 left), KEK (#4 right).



Kyber1024 WrapQ Key Export Random-vs-Fixed CSP (#5 left), KEK (#6 right).



Dilithium5 WrapQ Key Export Random-vs-Fixed CSP (#7 left), KEK (#8 right).

Fig. 2. Kyber and Dilithium average power traces and TVLA t-traces for WrapQ key import and export functions (See Table 4.) 100,000 traces were measured for each test. The TVLA results were well within leakage assessment boundaries (red lines.)

Vienna, Austria, October 24-28, 2016. pp. 116–129. ACM (2016). https://doi.org/10.1145/2976749.2978427, http://dl.acm.org/citation.cfm?id=2976749
7. Barthe, G., Belaïd, S., Espitau, T., Fouque, P., Grégoire, B., Rossi, M., Tibouchi, M.: Masking the GLP lattice-based signature scheme at any order. In: Nielsen, J.B., Rijmen, V. (eds.) 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. Lecture Notes in Computer Science, vol. 10821, pp. 354–384. Springer (2018). https://doi.org/10.1007/978-3-319-78375-8_12, https://eprint.iacr.org/2018/381
8. Becker, G., Cooper, J., DeMulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., Kouzminov, T., Leiserson, A., Marson, M., Rohatgi, P., Saab, S.: Test vector leakage assessment (TVLA) methodology in practice (2013), presented at International Cryptography Module Conference – ICMC 2013
9. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. J. Cryptol. **21**(4), 469–491 (2008). https://doi.org/10.1007/s00145-008-9026-x
10. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Building power analysis resistant implementations of Keccak (August 2010), https://csrc.nist.gov/Events/2010/The-Second-SHA-3-Candidate-Conference
11. Bos, J.W., Gourjon, M., Renes, J., Schneider, T., van Vredendaal, C.: Masking kyber: First- and higher-order implementations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 173–214 (2021). https://doi.org/10.46586/tches.v2021.i4.173-214
12. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener [39], pp. 398–412. https://doi.org/10.1007/3-540-48405-1_26
13. Daemen, J.: Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10529, pp. 137–153. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_7
14. Ding, A.A., Zhang, L., Durvaux, F., Standaert, F., Fei, Y.: Towards sound and optimal leakage detection procedure. In: Eisenbarth, T., Teglia, Y. (eds.) Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10728, pp. 105–122. Springer (2017). https://doi.org/10.1007/978-3-319-75208-2_7
15. Dworkin, M.: Recommendation for block cipher modes of operation: Methods for key wrapping. NIST Special Publication SP 800-38F (December 2012). https://doi.org/10.6028/NIST.SP.800-38F
16. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for sidechannel resistance validation. CMVP & AIST Non-Invasive Attack Testing Workshop (NIAT 2011) (September 2011), https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/08_goodwill.pdf
17. Halevi, S., Krawczyk, H.: Strengthening digital signatures via randomized hashing. In: Dwork, C. (ed.) Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4117, pp. 41–59. Springer (2006). https://doi.org/10.1007/11818175_3

18. Heinz, D., Kannwischer, M.J., Land, G., Pöppelmann, T., Schwabe, P., Sprenkels, D.: First-order masked Kyber on ARM Cortex-M4. IACR ePrint 2022/058 (2022), https://eprint.iacr.org/2022/058
19. Ishai, Y., Sahai, A., Wagner, D.A.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2729, pp. 463–481. Springer (2003). https://doi.org/10.1007/978-3-540-45146-4_27
20. ISO: Information technology – security techniques – security requirements for cryptographic modules. Standard ISO/IEC WD 19790:2022(E), International Organization for Standardization (2022)
21. ISO: Information technology – security techniques – testing methods for the mitigation of non-invasive attack classes against cryptographic modules. Draft International Standard ISO/IEC DIS 17825:2022(E), International Organization for Standardization (2023)
22. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1109, pp. 104–113. Springer (1996). https://doi.org/10.1007/3-540-68697-5_9
23. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener [39], pp. 388–397. https://doi.org/10.1007/3-540-48405-1_25
24. Menhorn, N.: External secure storage using the PUF. Application Note: Zynq UltraScale+ Devices, XAPP1333 (v1.2) (April 2022), 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), https://learn.microsoft.com/en-us/azure/key-vault/keys/byok-specification
26. Migliore, V., Gérard, B., Tibouchi, M., Fouque, P.: Masking Dilithium - efficient implementation and side-channel evaluation. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) Applied Cryptography and Network Security - 17th International Conference, ACNS 2019, Bogota, Colombia, June 5-7, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11464, pp. 344–362. Springer (2019). https://doi.org/10.1007/978-3-030-21568-2_17
27. Moriarty, K.M., Nystrom, M., Parkinson, S., Rusch, A., Scott, M.: PKCS #12: Personal information exchange syntax v1.1. IETF RFC 7292 (July 2014). https://doi.org/10.17487/RFC7292
28. NIST: SHA-3 standard: Permutation-based hash and extendable-output functions. Federal Information Processing Standards Publication FIPS 202 (August 2015). https://doi.org/10.6028/NIST.FIPS.202
29. NIST: Security requirements for cryptographic modules. Federal Information Processing Standards Publication FIPS 140-3 (March 2019). https://doi.org/10.6028/NIST.FIPS.140-3
30. NSA: Announcing the commercial national security algorithm suite 2.0. National Security Agency, Cybersecurity Advisory (September 2022), https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF
31. Perlner, R.: Planned changes to the Dilithium spec. Posting on PQC Forum (April 2023), https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/3pBJsYjfRw4/m/GjJ2icQkAQAJ

32. Prouff, E., Rivain, M.: Masking against side-channel attacks: A formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7881, pp. 142–159. Springer (2013). https://doi.org/10.1007/978-3-642-38348-9_9
33. Quisquater, J., Samyde, D.: Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In: Attali, I., Jensen, T.P. (eds.) Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2140, pp. 200–210. Springer (2001). https://doi.org/10.1007/3-540-45418-7_17
34. Rambus: Test vector leakage assessment (TVLA) derived test requirements (DTR) with AES. Rambus CRI Technical Note (February 2015), https://www.rambus.com/wp-content/uploads/2015/08/TVLA-DTR-with-AES.pdf
35. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6225, pp. 413–427. Springer (2010). https://doi.org/10.1007/978-3-642-15031-9_28
36. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002. pp. 98–107. ACM (2002). https://doi.org/10.1145/586110.586125, http://dl.acm.org/citation.cfm?id=586110
37. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) 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. Lecture Notes in Computer Science, vol. 4004, pp. 373–390. Springer (2006). https://doi.org/10.1007/11761679_23
38. Schneider, T., Moradi, A.: Leakage assessment methodology - A clear roadmap for side-channel evaluations. In: Güneysu, T., Handschuh, H. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9293, pp. 495–513. Springer (2015). https://doi.org/10.1007/978-3-662-48324-4_25
39. Wiener, M.J. (ed.): Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings, Lecture Notes in Computer Science, vol. 1666. Springer (1999). https://doi.org/10.1007/3-540-48405-1