CRYSTALS-Kyber

Algorithm Specifications And Supporting Documentation
(version 2.0)

Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint,
Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, Damien Stehlé

March 30, 2019

Changelog

In the following we list the changes from KYBER as submitted to the NIST PQC project in November 2017 and KYBER with round-2 tweaks together with brief explanations of the motivation for the changes.

Changes to the core KYBER design

- **Eliminate public-key compression.** As NIST remarked, the KYBER security proof only applies to a variant of round-1 KYBER that does not compress the public key. While we strongly believe that this didn't lower actual security, to alleviate any concerns, we decided to *not* include public-key compression in round-2 KYBER.
- **Change to $q = 3329$.** We balance the increased bandwidth requirement from dropping the public-key compression by choosing a smaller value of q. Specifically, we change q from 7681 to 3329. This change was enabled by recent results in [65] showing that we do not need to pick q such that $2n \mid (q-1)$ to obtain very fast inplace NTT-based multiplication. Using a q such that $n \mid (q-1)$ (and maybe even $\frac{n}{2} \mid q-1$) is enough to achieve equal (or even slightly better) performance.
- **Change to $\eta = 2$.** As a consequence of our smaller value of q, we also decrease the noise parameter η. This parameter is now the same for all security levels.
- **Update d_u and d_v.** We update the ciphertext-compression parameters d_u and d_v to achieve a similar balance of ciphertext size, failure probability, and security as in round-1 KYBER.
- **Update definition of the NTT.** Also as a consequence of changing to $q = 3329$, we need to update the definition of the NTT. Instead of decomposing an element $a \in \mathbb{Z}_q[X]/(X^{256}+1)$ as

  $$(a \bmod X - r_1, \ldots, a \bmod X - r_{256}),$$

  the NTT now decomposes it as

  $$(a \bmod X^2 - r'_1, \ldots, a \bmod X^2 - r'_{128}).$$

  In both cases, though, the NTT domain is still an element of $\mathbb{Z}_q^{256}$.
- **Represent public key in NTT domain.** As public keys are no longer compressed and as the NTT is already part of the specification, we also transmit public keys in NTT domain to save several NTT computations.
- **Update sampling of A.** Sampling of the public matrix $\mathbf{A}$ is updated to work for the new value of $q = 3329$.
- **Derive final key using SHAKE-256.** Instead of deriving the final shared key through SHA3-256 we now use SHAKE-256. This allows to easily derive keys of aritrary length (although this is not supported by the NIST API).

"90s" variant of KYBER

As an additional update for round 2, we also present a variant of KYBER that instead of relying on Keccak for all symmetric primitives, relies on AES and SHA-2. As the design of those primitives is rooted in the last years of the previous millenium, we refer to this variant as the "90s" variant.

The motivation for including this variant is the realization that optimized implementations of KYBER on different platforms consistently spend much more than half of the time in Keccak permutations. Benchmarking KYBER thus boils down to benchmarking the performance of Keccak, which is likely to change dramatically once hardware support becomes available. The 90s variant is thus meant to showcase the performance of KYBER if symmetric primitives are accelerated in hardware.

Furthermore, some experiments with post-quantum schemes—most notably the CECPQ1 and CECPQ2 TLS experiments by Google—ended up replacing Keccak-based symmetric primitives by AES and SHA-2 to achieve better performance. Should any early adopters want to use KYBER with AES and SHA-2, then we would rather have *one* variant of KYBER than a different one in each deployment.

Editorial changes to the specification

We edited the specification to match the round-2 parameters, fixed a few small small mistakes and updated performance numbers. We also added a section at the end that briefly discusses some recent papers that are relevant for the evaluation of Kyber.

Contents

1 Written specification

KYBER is an IND-CCA2-secure key-encapsulation mechanism (KEM), which has first been described in [22]. The security of KYBER is based on the hardness of solving the learning-with-errors problem in module lattices (MLWE problem [57]). The construction of KYBER follows a two-stage approach: we first introduce an IND-CPA-secure public-key encryption scheme encrypting messages of a fixed length of 32 bytes, which we call KYBER.CPAPKE. We then use a slightly tweaked Fujisaki–Okamoto (FO) transform [40] to construct the IND-CCA2-secure KEM. Whenever we want to emphasize that we are speaking about the IND-CCA2-secure KEM, we will refer to it as KYBER.CCAKEM.

In Subsection 1.1 we give preliminaries and fix notation. In Subsection 1.2 we give a full specification of KYBER.CPAPKE. Subsection 1.3 gives details of the transform that we use in KYBER to obtain KYBER.CCAKEM from KYBER.CPAPKE. Subsection 1.4 lists the parameters that we propose for different security levels. Finally, Subsection 1.5 explains the design rationale behind KYBER.

1.1 Preliminaries and notation.

Bytes and byte arrays. Inputs and outputs to all API functions of KYBER are byte arrays. To simplify notation, we denote by by $\mathcal{B}$ the set $\{0,\ldots,255\}$, i.e., the set of 8-bit unsigned integers (bytes). Consequently we denote by $\mathcal{B}^k$ the set of byte arrays of length k and by $\mathcal{B}^*$ the set of byte arrays of arbitrary length (or byte streams). For two byte arrays a and b we denote by $(a\|b)$ the concatenation of a and b. For a byte array a we denote by $a+k$ the byte array starting at byte k of a (with indexing starting at zero). For example, let a by a byte array of length ℓ, let b be another byte array and let $c=(a\|b)$ be the concatenation of a and b; then $b=a+\ell$. When it is more convenient to work with an array of bits than an array of bytes we make this conversion explicit via the BytesToBits function that takes as input an array of ℓ bytes and produces as output an array of 8ℓ bits. Bit β_i at position i of the output bit array is obtained from byte $b_{i/8}$ at position $i/8$ of the input array by computing $\beta_i = \left((b_{i/8}/2^{(i \mod 8)}) \mod 2\right)$.

Polynomial rings and vectors. We denote by R the ring $\mathbb{Z}[X]/(X^n+1)$ and by R_q the ring $\mathbb{Z}_q[X]/(X^n+1)$, where $n=2^{n'-1}$ such that X^n+1 is the $2^{n'}$-th cyclotomic polynomial. Throughout this document, the values of n, n' and q are fixed to $n=256$, $n'=9$, and $q=3329$. Regular font letters denote elements in R or R_q (which includes elements in $\mathbb{Z}$ and $\mathbb{Z}_q$) and bold lower-case letters represent vectors with coefficients in R or R_q. By default, all vectors will be column vectors. Bold upper-case letters are matrices. For a vector $\mathbf{v}$ (or matrix $\mathbf{A}$), we denote by $\mathbf{v}^T$ (or $\mathbf{A}^T$) its transpose. For a vector $\mathbf{v}$ we write $\mathbf{v}[i]$ to denote it's i-th entry (with indexing starting at zero); for a matrix $\mathbf{A}$ we write $\mathbf{A}[i][j]$ to denote the entry in row i, column j (again, with indexing starting at zero).

Modular reductions. For an even (resp. odd) positive integer α, we define $r' = r \bmod^{\pm} \alpha$ to be the unique element r' in the range $-\frac{\alpha}{2} < r' \leq \frac{\alpha}{2}$ (resp. $-\frac{\alpha-1}{2} \leq r' \leq \frac{\alpha-1}{2}$) such that $r' = r \bmod \alpha$. For any positive integer α, we define $r' = r \bmod^{+} \alpha$ to be the unique element r' in the range $0 \leq r' < \alpha$ such that $r' = r \bmod \alpha$. When the exact representation is not important, we simply write $r \bmod \alpha$.

Rounding. For an element $x \in \mathbb{Q}$ we denote by $\lceil x \rfloor$ rounding of x to the closest integer with ties being rounded up.

Sizes of elements. For an element $w \in \mathbb{Z}_q$, we write $\|w\|_\infty$ to mean $|w \bmod^{\pm} q|$. We now define the ℓ_∞ and ℓ_2 norms for $w = w_0 + w_1 X + \ldots + w_{n-1}X^{n-1} \in R$:

$$\|w\|_\infty = \max_i \|w_i\|_\infty, \quad \|w\| = \sqrt{\|w_0\|_\infty^2 + \ldots + \|w_{n-1}\|_\infty^2}.$$

Similarly, for $\mathbf{w} = (w_1,\ldots,w_k) \in R^k$, we define

$$\|\mathbf{w}\|_\infty = \max_i \|w_i\|_\infty, \quad \|\mathbf{w}\| = \sqrt{\|w_1\|^2 + \ldots + \|w_k\|^2}.$$

Sets and Distributions. For a set S, we write $s \leftarrow S$ to denote that s is chosen uniformly at random from S. If S is a probability distribution, then this denotes that s is chosen according to the distribution S.

Compression and Decompression. We now define a function $\mathsf{Compress}_q(x, d)$ that takes an element $x \in \mathbb{Z}_q$ and outputs an integer in $\{0, \ldots, 2^d - 1\}$, where $d < \lceil \log_2(q) \rceil$. We furthermore define a function $\mathsf{Decompress}_q$, such that

$$x' = \mathsf{Decompress}_q\big(\mathsf{Compress}_q(x, d), d\big) \tag{1}$$

is an element close to x – more specifically

$$|x' - x \bmod^{\pm} q| \leq B_q \coloneqq \left\lceil \frac{q}{2^{d+1}} \right\rfloor .$$

The functions satisfying these requirements are defined as:

$$\begin{aligned} \mathsf{Compress}_q(x, d) &= \lceil (2^d/q) \cdot x \rfloor \bmod^{+} 2^d , \\ \mathsf{Decompress}_q(x, d) &= \lceil (q/2^d) \cdot x \rfloor . \end{aligned}$$

When $\mathsf{Compress}_q$ or $\mathsf{Decompress}_q$ is used with $x \in R_q$ or $\mathbf{x} \in R_q^k$, the procedure is applied to each coefficient individually.

The main reason for defining the $\mathsf{Compress}_q$ and $\mathsf{Decompress}_q$ functions is to be able to discard some low-order bits in the ciphertext, which do not have much effect on the correctness probability of decryption – thus reducing the size of ciphertexts.

Yet, the $\mathsf{Compress}_q$ and $\mathsf{Decompress}_q$ are also used for another purpose than compression, namely to perform the usual LWE error correction during encryption and decryption. More precisely, in line 20 of the encryption procedure (Algorithm 5) the $\mathsf{Decompress}_q$ function is used to create error tolerance gaps by sending 0 to 0 and 1 to $\lceil q/2 \rfloor$. Later on, on line 4 of the decryption procedure (Algorithm 6), the $\mathsf{Compress}_q$ function is used to decrypt to a 1 if $v - \mathbf{s}^T\mathbf{u}$ is closer to $\lceil q/2 \rfloor$ than to 0, and decrypt to a 0 otherwise.

Symmetric primitives. The design of KYBER makes use of a pseudorandom function $\mathsf{PRF}\colon \mathcal{B}^{32} \times \mathcal{B} \to \mathcal{B}^*$ and of an extendable output function $\mathsf{XOF}\colon \mathcal{B}^* \times \mathcal{B} \times \mathcal{B} \to \mathcal{B}^*$. KYBER also makes use of two hash functions $\mathrm{H}\colon \mathcal{B}^* \to \mathcal{B}^{32}$ and $\mathrm{G}\colon \mathcal{B}^* \to \mathcal{B}^{32} \times \mathcal{B}^{32}$ and of a key-derivation function $\mathsf{KDF}\colon \mathcal{B}^* \to \mathcal{B}^*$.

NTTs, multiplication, and bitreversed order. A very efficient way to perform multiplications in R_q is via the so-called *number-theoretic transform* (NTT).

For our prime $q = 3329$ with $q - 1 = 2^8 \cdot 13$, the base field $\mathbb{Z}_q$ contains primitive 256-th roots of unity but not primitive 512-th roots. Therefore, the defining polynomial $X^{256} + 1$ of R factors into 128 polynomials of degree 2 modulo q and the NTT of a polynomial $f \in R_q$ is a vector of 128 polynomials of degree one. Simple in-place implementations of the NTT without reordering outputs these polynomials in bit-reversed order and we define the NTT in this way. Concretely, let $\zeta = 17$ be the first primitive 256-th root of unity modulo q, and $\{\zeta, \zeta^3, \zeta^5, \ldots, \zeta^{255}\}$ the set of all the 256-th roots of unity. The polynomial $X^{256} + 1$ can therefore be written as

$$X^{256} + 1 = \prod_{i=0}^{127}(X^2 - \zeta^{2i+1}) = \prod_{i=0}^{127}(X^2 - \zeta^{2\mathsf{br}_7(i)+1})$$

where $\mathsf{br}_7(i)$ for $i = 0, \ldots, 127$ is the bit reversal of the unsigned 7-bit integer i. This latter ordering of the factors is useful for compatibility with the idiosyncrasies of AVX instructions. Then the NTT of $f \in R_q$ is given by

$$(f \bmod X^2 - \zeta^{2\mathsf{br}_7(0)+1}, \ldots, f \bmod X^2 - \zeta^{2\mathsf{br}_7(127)+1}) \tag{2}$$

This vector of linear polynomials is then serialized to a vector in $\mathbb{Z}_q^{256}$ in the canonical way. Moreover, in order to not introduce additional data types and facilitate in-place implementations of the NTT we define $\mathsf{NTT}\colon R_q \to R_q$ to be the bijection that maps $f \in R_q$ to the polynomial with the aforementioned coefficient vector. Hence,

$$\mathsf{NTT}(f) = \hat{f} = \hat{f}_0 + \hat{f}_1 X + \cdots + \hat{f}_{255} X^{255}$$

with

$$\hat{f}_{2i} = \sum_{j=0}^{127} f_{2j}\zeta^{(2\mathsf{br}_7(i)+1)j}, \tag{3}$$

$$\hat{f}_{2i+1} = \sum_{j=0}^{127} f_{2j+1}\zeta^{(2\mathsf{br}_7(i)+1)j}. \tag{4}$$

We would like to stress that even though we write $\hat{f}$ as a polynomial in R_q, it has no algebraic meaning as such. The natural algebraic representation of $\mathsf{NTT}(f) = \hat{f}$ is as 128 polynomials of degree 1 as in (2) using the definitions for $\hat{f}_i$ from (3) and (4). That is,

$$\mathsf{NTT}(f) = \hat{f} = (\hat{f}_0 + \hat{f}_1 X, \hat{f}_2 + \hat{f}_3 X, \ldots, \hat{f}_{254} + \hat{f}_{255} X).$$

Using NTT and its inverse NTT^{-1} we can compute the product $f \cdot g$ of two elements $f, g \in R_q$ very efficiently as $\mathsf{NTT}^{-1}(\mathsf{NTT}(f) \circ \mathsf{NTT}(g))$ where $\mathsf{NTT}(f) \circ \mathsf{NTT}(g) = \hat{f} \circ \hat{g} = \hat{h}$ denotes the basecase multiplication consisting of the 128 products

$$\hat{h}_{2i} + \hat{h}_{2i+1} X = (\hat{f}_{2i} + \hat{f}_{2i+1} X)(\hat{g}_{2i} + \hat{g}_{2i+1} X) \bmod X^2 - \zeta^{2\mathsf{br}_7(i)+1}$$

of linear polynomials.

When we apply NTT or NTT^{-1} to a vector or matrix of elements of R_q, then this means that the respective operation is applied to each entry individually. When we apply $\circ$ to matrices or vectors it means that we perform a usual matrix multiplication, but that the individual products of entries are the above basecase multiplications.

Throughout the document we will write NTT and NTT^{-1} whenever we refer to the concrete functions as defined above and use normal-font NTT whenever we refer to the general technique.

Uniform sampling in R_q. Kyber uses a deterministic approach to sample elements in R_q that are statistically close to a uniformly random distribution. For this sampling we use a function $\mathsf{Parse}\colon \mathcal{B}^* \to R_q$, which receives as input a byte stream $B = b_0, b_1, b_2, \ldots$ and computes the NTT-representation $\hat{a} = \hat{a}_0 + \hat{a}_1 X + \cdots + \hat{a}_{n-1} X^{n-1} \in R_q$ of $a \in R_q$. Parse is described in Algorithm 1 (note that this description assumes that $q = 3329$).

Algorithm 1 $\mathsf{Parse}\colon \mathcal{B}^* \to R_q^n$

Input: Byte stream $B = b_0, b_1, b_2 \cdots \in \mathcal{B}^*$
Output: NTT-representation $\hat{a} \in R_q$ of $a \in R_q$

```
i := 0
j := 0
while j < n do
    d := b_i + 256 · b_{i+1}
    if d < 19q then
        â_j := d
        j := j + 1
    end if
    i := i + 2
end while
return â_0 + â_1 X + ··· + â_{n−1} X^{n−1}
```

The intuition behind the function Parse is that if the input byte array is statistically close to a uniformly random byte array, then the output polynomial is statistically close to a uniformly random element of R_q. It represents a uniformly random polynomial in R_q because NTT is bijective and thus maps polynomials with uniformly random coefficients to polynomials with again uniformly random coefficients.

Sampling from a binomial distribution. Noise in KYBER is sampled from a centered binomial distribution B_η for $\eta = 2$. We define B_η as follows:

$$\text{Sample } (a_1, \ldots, a_\eta, b_1, \ldots, b_\eta) \leftarrow \{0,1\}^{2\eta}$$

$$\text{and output } \sum_{i=1}^{\eta}(a_i - b_i).$$

When we write that a polynomial $f \in R_q$ or a vector of such polynomials is sampled from B_η, we mean that each coefficient is sampled from B_η.

For the specification of KYBER we need to define how a polynomial $f \in R_q$ is sampled according to B_η deterministically from 64η bytes of output of a pseudorandom function (we fix $n = 256$ in this description). This is done by the function CBD (for "centered binomial distribution") defined as described in Algorithm 2.

Algorithm 2 $\mathsf{CBD}_\eta \colon \mathcal{B}^{64\eta} \to R_q$

Input: Byte array $B = (b_0, b_1, \ldots, b_{64\eta-1}) \in \mathcal{B}^{64\eta}$
Output: Polynomial $f \in R_q$
  $(\beta_0, \ldots, \beta_{512\eta-1}) \coloneqq \mathsf{BytesToBits}(B)$
  for i from 0 to 255 **do**
    $a \coloneqq \sum_{j=0}^{\eta-1} \beta_{2i\eta+j}$
    $b \coloneqq \sum_{j=0}^{\eta-1} \beta_{2i\eta+\eta+j}$
    $f_i \coloneqq a - b$
  end for
  return $f_0 + f_1 X + f_2 X^2 + \cdots + f_{255} X^{255}$

Encoding and decoding. There are two data types that KYBER needs to serialize to byte arrays: byte arrays and (vectors of) polynomials. Byte arrays are trivially serialized via the identity, so we need to define how we serialize and deserialize polynomials. In Algorithm 3 we give a pseudocode description of the function Decode_ℓ, which deserializes an array of 32ℓ bytes into a polynomial $f = f_0 + f_1 X + \cdots + f_{255} X^{255}$ (we again fix $n = 256$ in this description) with each coefficient f_i in $\{0, \ldots, 2^\ell - 1\}$. We define the function Encode_ℓ as the inverse of Decode_ℓ. Whenever we apply Encode_ℓ to a vector of polynomials we encode each polynomial individually and concatenate the output byte arrays.

Algorithm 3 $\mathsf{Decode}_\ell \colon \mathcal{B}^{32\ell} \to R_q$

Input: Byte array $B \in \mathcal{B}^{32\ell}$
Output: Polynomial $f \in R_q$
  $(\beta_0, \ldots, \beta_{256\beta-1}) \coloneqq \mathsf{BytesToBits}(B)$
  for i from 0 to 255 **do**
    $f_i \coloneqq \sum_{j=0}^{\ell-1} \beta_{i\ell+j} 2^j$
  end for
  return $f_0 + f_1 X + f_2 X^2 + \cdots + f_{255} X^{255}$

1.2 Specification of KYBER.CPAPKE

KYBER.CPAPKE is similar to the LPR encryption scheme that was introduced (for Ring-LWE) by Lyubashevsky, Peikert, and Regev in the presentation of [61] at Eurocrypt 2010 [62]; the description is also in the full version of the paper [63, Sec. 1.1]. The roots of this scheme go back to the first LWE-based encryption scheme presented by Regev in [81, 82], with the main difference being that the underlying ring is not $\mathbb{Z}_q$ and both the secret and the error vectors have small coefficients. The idea of using a polynomial ring (instead of $\mathbb{Z}_q$) goes back to the NTRU cryptosystem presented by Hoffstein, Pipher, and Silverman in [47], while the symmetry between the secret and the error was already employed in very similar cryptographic schemes in [6, 60] with the security justification from [10].

The main difference from the LPR encryption scheme is to use Module-LWE instead of Ring-LWE. Also, we adopt the approach taken by Alkım, Ducas, Pöppelmann and Schwabe in [7] for the generation of the public matrix $\mathbf{A}$. Furthermore, we shorten ciphertexts by rounding off the low bits as in learning-with-rounding-based schemes [13, Eq. 2.1], which is a common technique for reducing ciphertext size also in LWE-based schemes (c.f. [71, 77]).

Parameters. KYBER.CPAPKE is parameterized by integers n, k, q, η, d_u, and d_v. As stated before, throughout this document n is always 256 and q is always 3329. Furthermore, throughout this document η is always 2. The values of k, d_u and d_v vary for different security levels.

Using the notation of Subsection 1.1 we give the definition of key generation, encryption, and decryption of the KYBER.CPAPKE public-key encryption scheme in Algorithms 4, 5, and 6. A more high-level view of these algorithms is given in the comments.

Algorithm 4 KYBER.CPAPKE.KeyGen(): key generation

Output: Secret key $sk \in \mathcal{B}^{12 \cdot k \cdot n/8}$
Output: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8+32}$

$d \leftarrow \mathcal{B}^{32}$
$(\rho, \sigma) := \mathrm{G}(d)$
$N := 0$
for i from 0 to $k-1$ **do** ▷ Generate matrix $\hat{\mathbf{A}} \in R_q^{k \times k}$ in NTT domain
  for j from 0 to $k-1$ **do**
    $\hat{\mathbf{A}}[i][j] := \mathsf{Parse}(\mathsf{XOF}(\rho, j, i))$
  end for
end for
for i from 0 to $k-1$ **do** ▷ Sample $\mathbf{s} \in R_q^k$ from B_η
  $\mathbf{s}[i] := \mathsf{CBD}_\eta(\mathsf{PRF}(\sigma, N))$
  $N := N + 1$
end for
for i from 0 to $k-1$ **do** ▷ Sample $\mathbf{e} \in R_q^k$ from B_η
  $\mathbf{e}[i] := \mathsf{CBD}_\eta(\mathsf{PRF}(\sigma, N))$
  $N := N + 1$
end for
$\hat{\mathbf{s}} := \mathsf{NTT}(\mathbf{s})$
$\hat{\mathbf{e}} := \mathsf{NTT}(\mathbf{e})$
$\hat{\mathbf{t}} := \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
$pk := (\mathsf{Encode}_{12}(\hat{\mathbf{t}} \mod^+ q) \| \rho)$ ▷ $pk := \mathbf{As} + \mathbf{e}$
$sk := \mathsf{Encode}_{12}(\hat{\mathbf{s}} \mod^+ q)$ ▷ $sk := \mathbf{s}$
return (pk, sk)

Algorithm 5 KYBER.CPAPKE.Enc(pk, m, r): encryption

Input: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8+32}$
Input: Message $m \in \mathcal{B}^{32}$
Input: Random coins $r \in \mathcal{B}^{32}$
Output: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

$N \coloneqq 0$
$\hat{\mathbf{t}} \coloneqq \mathsf{Decode}_{12}(pk)$
$\rho \coloneqq pk + 12 \cdot k \cdot n/8$
for i from 0 to $k-1$ **do** ▷ Generate matrix $\hat{\mathbf{A}} \in R_q^{k \times k}$ in NTT domain
  for j from 0 to $k-1$ **do**
    $\hat{\mathbf{A}}^T[i][j] \coloneqq \mathsf{Parse}(\mathsf{XOF}(\rho, i, j))$
  end for
end for
for i from 0 to $k-1$ **do** ▷ Sample $\mathbf{r} \in R_q^k$ from B_η
  $\mathbf{r}[i] \coloneqq \mathsf{CBD}_\eta(\mathsf{PRF}(r, N))$
  $N \coloneqq N + 1$
end for
for i from 0 to $k-1$ **do** ▷ Sample $\mathbf{e}_1 \in R_q^k$ from B_η
  $\mathbf{e}_1[i] \coloneqq \mathsf{CBD}_\eta(\mathsf{PRF}(r, N))$
  $N \coloneqq N + 1$
end for
$e_2 \coloneqq \mathsf{CBD}_\eta(\mathsf{PRF}(r, N))$ ▷ Sample $e_2 \in R_q$ from B_η
$\hat{\mathbf{r}} \coloneqq \mathsf{NTT}(\mathbf{r})$
$\mathbf{u} \coloneqq \mathsf{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$ ▷ $\mathbf{u} \coloneqq \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$
$v \coloneqq \mathsf{NTT}^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + \mathsf{Decompress}_q(\mathsf{Decode}_1(m), 1)$ ▷ $v \coloneqq \mathbf{t}^T\mathbf{r} + e_2 + \mathsf{Decompress}_q(m, 1)$
$c_1 \coloneqq \mathsf{Encode}_{d_u}(\mathsf{Compress}_q(\mathbf{u}, d_u))$
$c_2 \coloneqq \mathsf{Encode}_{d_v}(\mathsf{Compress}_q(v, d_v))$
return $c = (c_1 \| c_2)$ ▷ $c \coloneqq (\mathsf{Compress}_q(\mathbf{u}, d_u), \mathsf{Compress}_q(v, d_v))$

Algorithm 6 KYBER.CPAPKE.Dec(sk, c): decryption

Input: Secret key $sk \in \mathcal{B}^{12 \cdot k \cdot n/8}$
Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$
Output: Message $m \in \mathcal{B}^{32}$

$\mathbf{u} \coloneqq \mathsf{Decompress}_q(\mathsf{Decode}_{d_u}(c), d_u)$
$v \coloneqq \mathsf{Decompress}_q(\mathsf{Decode}_{d_v}(c + d_u \cdot k \cdot n/8), d_v)$
$\hat{\mathbf{s}} \coloneqq \mathsf{Decode}_{12}(sk)$
$m \coloneqq \mathsf{Encode}_1(\mathsf{Compress}_q(v - \mathsf{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \mathsf{NTT}(\mathbf{u})), 1))$ ▷ $m \coloneqq \mathsf{Compress}_q(v - \mathbf{s}^T\mathbf{u}, 1))$
return m

1.3 Specification of KYBER.CCAKEM

We construct the KYBER.CCAKEM IND-CCA2-secure KEM from the IND-CPA-secure public-key encryption scheme described in the previous subsection via a slightly tweaked Fujisaki–Okamoto transform [40]. In Algorithms 7, 8, and 9 we define key generation, encapsulation, and decapsulation of KYBER.CCAKEM.

Algorithm 7 KYBER.CCAKEM.KeyGen()

Output: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8+32}$
Output: Secret key $sk \in \mathcal{B}^{24 \cdot k \cdot n/8+96}$

$z \leftarrow \mathcal{B}^{32}$
$(pk, sk') := \text{KYBER.CPAPKE.KeyGen}()$
$sk := (sk' \| pk \| \text{H}(pk) \| z)$
return (pk, sk)

Algorithm 8 KYBER.CCAKEM.Enc(pk)

Input: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8+32}$
Output: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8+d_v \cdot n/8}$
Output: Shared key $K \in \mathcal{B}^*$

$m \leftarrow \mathcal{B}^{32}$
$m \leftarrow \text{H}(m)$ ▷ Do not send output of system RNG
$(\bar{K}, r) := \text{G}(m \| \text{H}(pk))$
$c := \text{KYBER.CPAPKE.Enc}(pk, m, r)$
$K := \text{KDF}(\bar{K} \| \text{H}(c))$
return (c, K)

Algorithm 9 KYBER.CCAKEM.Dec(c, sk)

Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8+d_v \cdot n/8}$
Input: Secret key $sk \in \mathcal{B}^{24 \cdot k \cdot n/8+96}$
Output: Shared key $K \in \mathcal{B}^*$

$pk := sk + 12 \cdot k \cdot n/8$
$h := sk + 24 \cdot k \cdot n/8 + 32 \in \mathcal{B}^{32}$
$z := sk + 24 \cdot k \cdot n/8 + 64$
$m' := \text{KYBER.CPAPKE.Dec}(\mathbf{s}, (\mathbf{u}, v))$
$(\bar{K}', r') := \text{G}(m' \| h)$
$c' := \text{KYBER.CPAPKE.Enc}(pk, m', r')$
if $c = c'$ **then**
    return $K := \text{KDF}(\bar{K}' \| \text{H}(c))$
else
    return $K := \text{KDF}(z \| \text{H}(c))$
end if
return K

1.4 KYBER parameter sets

We define three parameter sets for KYBER, which we call KYBER512, KYBER768, and KYBER1024. The parameters are listed in Table 1. Note that the table also lists the derived parameter δ, which is the probability that decapsulation of a valid KYBER.CCAKEM ciphertext fails. The parameters were obtained via the following approach:

Table 1: Parameter sets for KYBER

	n	k	q	η	(d_u, d_v)	δ
KYBER512	256	2	3329	2	$(10, 3)$	2^{-178}
KYBER768	256	3	3329	2	$(10, 4)$	2^{-164}
KYBER1024	256	4	3329	2	$(11, 5)$	2^{-174}

- n is set to 256 because the goal is to encapsulate keys with 256 bits of entropy (i.e., use a plaintext size of 256 bits in KYBER.CPAPKE.Enc). Smaller values of n would require to encode multiple key bits into one polynomial coefficient, which requires lower noise levels and therefore lowers security. Larger values of n would reduce the capability to easily scale security via parameter k.
- We choose q as a small prime satisfying $n \mid (q-1)$; this is required to enable the fast NTT-based multiplication. There are two smaller primes for this property holds, namely 257 and 769. However, for those primes we would not be able to achieve negligible failure probability required for CCA security, so we chose the next largest, i.e., $q = 3329$.
- k is selected to fix the lattice dimension as a multiple of n; changing k is the main mechanism in KYBER to scale security (and as a consequence, efficiency) to different levels.
- The remaining parameters η, d_u and d_v were chosen to balance between security (see Section 4), ciphertext size, and failure probability. Note that all three parameter sets achieve a failure probability of $< 2^{-128}$ with some margin. We discuss this in more detail in Subsections 1.5 and 5.3. Also note that $\eta = 2$ is fixed for all variants, which simplifies implementations.

The failure probability δ is computed with the help of the `Kyber.py` Python script which is available online at https://github.com/pq-crystals/security-estimates. For the theoretical background of that script see [22, Theorem 1].

Instantiating PRF, XOF, H, G, and KDF. What is still missing to complete the specification of KYBER is the instantiation of the symmetric primitives. We instantiate all of those primitives with functions from the FIPS-202 standard [68] as follows:

- We instantiate XOF with SHAKE-128;
- we instantiate H with SHA3-256;
- we instantiate G with SHA3-512;
- we instantiate $\mathsf{PRF}(s, b)$ with SHAKE-256$(s||b)$; and
- we instantiate KDF with SHAKE-256.

"90s" variant of KYBER In the 90s variant of KYBER

- we instantiate $\mathsf{XOF}(\rho, i, j)$ with AES-256 in CTR mode, where ρ is used as the key and $i\|j$ is zero-padded to a 12-byte nonce. The counter of CTR mode is initialized to zero.
- we instantiate H with SHA-256;
- we instantiate G with SHA-512;
- we instantiate $\mathsf{PRF}(s, b)$ with AES-256 in CTR mode, where s is used as the key and b is zero-padded to a 12-byte nonce. The counter of CTR mode is initialized to zero.
- we instantiate KDF with SHAKE-256.

1.5 Design rationale

The design of KYBER is based on the module version [57] of the Ring-LWE LPR encryption scheme [61] with bit-dropping [71, 77]. It is also enhanced by many of the improvements of preceding implementations of lattice-based encryption schemes such as NEWHOPE [7]. In NEWHOPE (and all other Ring-LWE schemes), operations were of the form $\mathbf{As}+\mathbf{e}$ where all the variables were polynomials in some ring. The main difference in KYBER is that $\mathbf{A}$ is now a matrix (with a small dimension like 3) over a constant-size polynomial ring and $\mathbf{s}, \mathbf{e}$ are vectors over the same ring. We refer to this as a scheme over "module lattices."

The use of Module-LWE. Previous proposals of LWE-based cryptosystems either used the very structured Ring-LWE problem (as, for example, NEWHOPE [7]) or standard LWE (as, for example, Frodo [21]). The main advantage of structured LWE variants based on polynomial rings is efficiency in terms of both speed and key and ciphertext sizes. The disadvantages are concerns that the additional structure might enable more efficient attacks and that tradeoffs between efficiency and security can be scaled only rather coarsely. The advantages of standard LWE is the lack of structure and easy scalability, but those come at the cost of significantly decreasing efficiency. Module-LWE offers a trade-off between these two extremes. In the specific case of the Module-LWE parameters used in KYBER, we obtain somewhat reduced structure compared to Ring-LWE, much better scalability, and—when encrypting messages of a fixed size of 256 bits—performance very similar to Ring-LWE-based schemes.

Active security. In [23], Bos, Costello, Naehrig, and Stebila used a *passively secure* KEM to migrate TLS to transitional post-quantum security (i.e., post-quantum confidentiality, but only pre-quantum authentication). Subsequent work, like NEWHOPE [7] or Frodo [21] followed up and proposed more efficient and more conservative instantiations of the underlying passively secure KEM. One advantage of passively secure KEMs is that they can accept a higher failure probability (which allows to either increase security by increasing noise or decreasing public-key and ciphertext size). The other advantage is that they do not require a CCA transform, and therefore come with faster decapsulation. Despite these advantages, KYBER is defined as an IND-CCA2 secure KEM only. For many applications like public-key encryption (via a KEM-DEM construction) or in authenticated key exchange active security is mandatory. However, also in use cases (like key exchange in TLS) that do not strictly speaking require active security, using an actively secure KEM has advantages. Most notably, it allows (intentional or accidental) caching of ephemeral keys. Furthermore, the CCA transform of KYBER protects against certain bugs in implementations. Specifically, passively secure schemes will not notice if the communication partner uses "wrong" noise, for example, all-zero noise. Such a bug in the encapsulation of KYBER will immediately be caught by the re-encryption step during decapsulation. As a conclusion, we believe that the overhead of providing CCA security is not large enough to justify saving it and making the scheme less robust.

The role of the NTT. Multiplication in R_q based on the number-theoretic transform (NTT) has multiple advantages: it is extremely fast, does not require additional memory (like, for example, Karatsuba or Toom multiplication) and can be done in very little code space. Consequently, it has become common practice to choose parameters of lattice-based crypto to support this very fast multiplication algorithm. Some schemes go further and make the NTT part of the definition of the scheme. A prominent example is again NEWHOPE, which samples the public value $\mathbf{a}$ in NTT domain and also sends public keys and ciphertexts in NTT domain to save 2 NTTs. NEWHOPE was not the first scheme to do this; for earlier examples see [59, 77, 83].

In KYBER we also decided to make the NTT part of the definition of the scheme, but only in the sampling of $\mathbf{A}$ and the public key, not for the format of the ciphertext. A consequence of this decision is that the NTT appears in the specification of KYBER.CPAPKE. Note that multiplications by $\mathbf{A}$ have to use the NTT, simply because $\hat{\mathbf{A}}$ is sampled in NTT domain[1]. Similarly, multiplications by the public-key $\hat{\mathbf{t}}$ have to use the NTT, because the public key is transmitted in NTT domain. As a consequence, implementations will also want to use the NTT for all other multiplications, so we make those invocations of NTT and NTT^{-1} also explicit in Alg. 4, Alg. 5, and Alg. 6. Note that also the secret key sk is stored in NTT domain.

We could have chosen to *not* make the NTT part of the definition of KYBER, which would have increased simplicity of the description. The cost for this increased simplicity would have been k^2 additional NTT operations in both key generation and encapsulation, which would result in a significant slowdown. We could also have chosen to not encode the public key in NTT domain; however, this would come at the cost

[1] An alternative would be to apply NTT^{-1} to $\hat{\mathbf{A}}$ but that would counteract the whole point of sampling $\mathbf{A}$ in NTT domain.

of additional NTTs. Finally, we could have chosen to also send the ciphertext in NTT domain; however, this would be incompatible with ciphertext compression via the $\mathsf{Compress}_q$ function.

Against all authority. For the generation of the public uniformly random matrix $\mathbf{A}$, we decided to adopt the "against-all-authority" approach of NewHope. This means that the matrix is not a system parameter but instead generated freshly as part of every public key. There are two advantages to this approach: First, this avoids discussions about how exactly a uniformly random system parameter was generated. Second, it protects against the all-for-the-price-of-one attack scenario of an attacker using a serious amount of computation to find a short basis of the lattice spanned by $\mathbf{A}$ *once* and then using this short basis to attack *all* users. The cost for this decision is the expansion of the matrix $\mathbf{A}$ from a random seed during key generation and encapsulation; we discuss this cost more in Subsection 2.1.

Binomial noise. Theoretic treatments of LWE-based encryption typically consider LWE with Gaussian noise, either rounded Gaussian [81] or discrete Gaussian [25]. As a result, many early implementations also sampled noise from a discrete Gaussian distribution, which turns out to be either fairly inefficient (see, for example, [23]) or vulnerable to timing attacks (see, for example, [26, 76, 38]). The performance of the best known attacks against LWE-based encryption does not depend on the exact distribution of noise, but rather on the standard deviation (and potentially the entropy). This motivates the use of noise distributions that we can easily, efficiently, and securely sample from. One example is the centered binomial distribution used in [7]. Another example is the use of "learning-with-rounding" (LWR), which adds deterministic uniform noise by dropping bits as in Kyber's $\mathsf{Compress}_q$ function. In the design of Kyber we decided to use centered binomial noise and thus rely on LWE instead of LWR as the underlying problem. The compression of ciphertexts via $\mathsf{Compress}_q$ introduces additional noise (making the scheme more secure), but we do not consider this noise in our security analysis (this choice is motivated by the absence of a Ring/Module variant of a hardness reduction for LWR [13]).

Allowing decapsulation failures. Another interesting design decision is whether to allow decapsulation failures (i.e., decryption failures in Kyber.CPAPKE) or choose parameters that not only have a negligible, but a zero chance of failure. The advantages of zero failure probability are obvious: CCA transforms and security proofs become easier and we could have avoided a whole discussion of attacks exploiting decapsulation failures in Subsection 5.3. The disadvantage of designing LWE-based encryption with zero failures is that it means either decreasing security against attacks targeting the underlying lattice problem (by significantly decreasing the noise) or decreasing performance (by compensating for the loss in security via an increase of the lattice dimension). The decision to allow failure probabilities of $< 2^{-160}$ in all parameter sets of Kyber reflects the intuition that

- decapsulation failures are a problem if they appear with non-negligible probability; but
- attacks attempting to exploit failures that occur with extremely low probability as in Kyber are a much smaller threat than, for example, improvements to hybrid attacks [49] targeting schemes with very low noise.

Additional Hashes. In the CCA transform we hash the (hash of the) public key pk into the pre-key $\bar{K}$ and into the random coins r (see line 3 of Alg. 8), and we hash the (hash of the) ciphertext into the final key K. These hashes would not be necessary for the security reduction (see Section 4), but they add *robustness*. Specifically, the final shared key output by Kyber.CCAKEM depends on the full view of exchanged messages (public key and ciphertext), which means that the KEM is contributory and safe to use in authenticated key exchanges without additional hashing of context. Hashing pk also into the random coins r adds protection against a certain class of multi-target attacks that attempt to make use of protocol failures. This is discussed in more detail in Subsection 5.3.

Choice of symmetric primitives. In the design of Kyber we need an extendable output function (XOF), two hash functions, a pseudorandom function, and a KDF. We decided to rely on only one underlying primitive for all those functions. This helps to reduce code size in embedded platforms and (for a conservative choice) reduces concerns that Kyber could be attacked by exploiting weaknesses in *one out of several* symmetric primitives. There are only relatively few extendable output functions described in the literature. The best known ones, which also coined the term XOF, are the SHAKE functions based on Keccak [19] and

standardized in FIPS-202 [68]. This standard conveniently also describes hash functions with the output lengths we need; furthermore, SHAKE is designed to also work as a PRF. These properties of the FIPS-202 function family made the choice easy, but there are still two decisions that may need explanation:

- We could have chosen to instantiate all symmetric primitives with only *one* function (e.g., SHAKE-256) from the FIPS-202 standard. The choice of SHAKE-128 as instantiation of the XOF is actually important for performance; also we do not need any of the traditional security properties of hash functions from SHAKE-128, but rather that the output "looks uniformly random". In an earlier version of Kyber we instantiated H, G, and PRF all with SHAKE-256. We decided to change this to *different* functions from the FIPS-202 family to avoid any domain-separation discussion. Note that this decision increases code-size at most marginally: all 4 functions can be obtained by a call to a "Keccak" function with appropriate arguments (see, for example, [18]).
- We could have decided to use KMAC from NIST Special Publication 800-185 to instantiate the PRF. We decided against this, because it would increase the numbers of Keccak permutations required in the generation of the noise polynomials and thus noticeably and unnecessarily decrease performance.

As a modification in round-2, we derive the final key using SHAKE-256 instead of SHA3-256. This is an advantage for protocols that need keys of more than 256 bits. Instead of first requesting a 256-bit key from Kyber and then expanding it, they can pass an additional key-length parameter to Kyber and obtain a key of the desired length. This feature is not supported by the NIST API, so in our implementations we set the keylength to a fixed length of 32 bytes in `api.h`.

Choice of symmetric primitives in the "90s" variant The 90s variant of Kyber uses symmetric primitives that are standardized by NIST and accelerated in hardware on a large variety of platforms. These two criteria narrow the choice to AES and SHA-256, which are, for example, implemented in hardware on recent Intel, AMD, and ARM processors. A natural choice for the hash function G with 512-bit output is SHA-512 from the same SHA-2 family of hash functions as SHA-256.

Supporting non-incremental hash APIs. In line 3 of Alg. 8 we feed $\mathrm{H}(pk)$ (instead of pk) into G and in line 5 we feed $\mathrm{H}(c)$ (instead of c) into H. Using $\mathrm{H}(pk)$ in the call to G enables a small speedup for decapsulation as described in Subsection 2.1. However, there is another reason why we first hash pk and c, namely that it simplifies implementing Kyber with a non-incremental hash API. If Kyber is implemented in an environment which already offers a library for hashing, but only offers calls of the form $h = \mathrm{H}(m)$, then producing a hash of the form $h = \mathrm{H}(m_1 \| m_2)$ would first require copying m_1 and m_2 into one consecutive area of memory. This would require unnecessary copies and, more importantly, additional stack space. Such non-incremental hash APIs are not uncommon: one example is the API of NaCl [17].

Return value for decapsulation failure. Traditionally the FO transform returns $\perp$ (i.e., a special failure symbol) when decapsulation fails. We use a variant that instead sets the resulting shared key to a pseudo-random value computed as the hash of a secret z and the ciphertext c. This variant of the FO transform was proven secure in [48]. In practice it has the advantage that implementations of Kyber's decapsulation are safe to use even if higher level protocols fail to check the return value. As a consequence of this *implicit rejection* approach, our implementations of decapsulation always return `0`.

2 Performance analysis

In this section we consider implementational aspects of Kyber and report performance results of two implementations: the ANSI C reference implementation requested by NIST and an implementation optimized using AVX2 vector instructions. included in the submission package under `Additional_Implementations/avx2/`. We remark that the optimized implementation in ANSI C in subdirectory `Optimized_Implementation/`, as requested by the Call for Proposals, is a copy of the reference implementation.

The big picture of Kyber performance. Thanks to the extremely efficient NTT-based multiplication and sampling of $\mathbf{A}$ in NTT domain, the performance of Kyber is largely determined by the performance of the symmetric primitives. This is illustrated, for example, by the fact that the AVX2-optimized implementation of the 90s variant of Kyber is almost a factor fo 2 faster than Kyber with symmetric primitives based on

Keccak. This difference is going to be even larger on systems with hardware-accelerated SHA-256. It is also illustrated by the fact that for optimized implementations decapsulation is *faster* than encapsulation, although it contains a re-encapsulation from the FO transform. The reason is that decapsulation can save, for example, computing H(pk) and sampling the 32 bytes of randomness.

2.1 Implementation considerations and tradeoffs

Implementing the NTT. Many different tradeoffs are possible when implementing the number-theoretic transform. The most important ones are between code size (which becomes mainly relevant on embedded processors) and speed. The two implementations of KYBER included in the submission package have a dedicated forward NTT (from normal to bitreversed order) and inverse NTT (from bitreversed to normal order). Also, both implementations use precomputed tables of powers of ζ. What is particularly interesting about using the NTT on embedded platforms is that the multiplication of two elements of R_q can be computed without any additional temporary storage. What is particularly interesting about using the NTT on large processors is that it is extremely efficiently vectorizable. Since 2013, the most efficient approach to compute the NTT on 64-bit Intel processors was to represent coefficients as double-precision floating-point values [44, 7]. In our AVX2-optimized implementation of KYBER, we show that carefully optimizing the NTT using AVX2 *integer* instructions results in much better performance. Specifically, on Intel Haswell CPUs one forward NTT in KYBER takes only about 320 cycles; an inverse NTT takes only about 290 cycles.

Keccak. The second speed-critical component inside KYBER are the symmetric primitives, i.e., SHA3-256, SHA3-512, SHAKE-128, and SHAKE-256, all based on the Keccak permutation. SHA3 has the reputation to not be the fastest hash function in software (see, for example, [56]). To some extent this is compensated by the fact that most calls to Keccak are parallel and thus very efficiently vectorizable. Our AVX2 implementation makes use of this fact. Also, ARM recently announced that future ARMv8 processors will have hardware support for SHA3 [43], so there is a good chance that at least on some architectures, software performance of SHA3 will not be an issue in the future.

AES and SHA-2. To illustrate what performance KYBER can achieve with hardware accelerated symmetric primitives we include the 90s variant using AES, SHA256 and SHA512 instead of symmetric primitives based on Keccak. This variant is interesting only if at least AES is accelerated in hardware as constant-time software implementations (required for the use of AES as a PRF) are not faster than parallel Keccak.

Hardware-RNGs for key generation. During key generation, the generation of $\mathbf{s}$ and $\mathbf{e}$ is performed using SHAKE-256 (and AES in the 90s variant). However, this is not required. The choice of RNG during key generation is a local decision that any user and platform can make independently. In particular on platforms with fast hardware AES one can adapt the AES-based PRF from the 90s variant also for the otherwise Keccak-based KYBER. We considered using this in our AVX2 implementation, but using this optimization means that testvectors would not match between our two implementations. This is not an issue in actual deployments, where `randombytes` is not deterministic.

Caching of ephemeral keys. Applications that are even more conscious of key-generation time can decide to cache ephemeral keys for some time. This is enabled by the fact that KYBER is IND-CCA2 secure.

Tradeoffs between secret-key size and speed. It is possible to use different tradeoffs between secret-key size and decapsulation speed. If secret-key size is critical, it is of course possible to not store H(pk) and also to not store the public key as part of the secret key but instead recompute it during decapsulation. Furthermore, not keeping the secret key in NTT domain makes it possible to compress each coefficient to only 5 bits, resulting in a total size of only 320 bytes for the three polynomials. This has the additional advantage that it makes key recovery via cold-boot attacks [45] somewhat harder [3]. Finally, as all randomness in key generation is generated from two 32-byte seeds, it is also possible to only store these seeds and re-run key generation during decapsulation. When opting for such 32-byte secret keys, the re-encapsulation step of decapsulation can save the expansion of the matrix $\mathbf{A}$, is it is already expanded (in transposed form) in key generation.

In the other direction, if secret-key size does not matter very much and decapsulation speed is critical, one might decide to store the expanded matrix $\mathbf{A}$ as part of the secret key and avoid recomputation from the seed ρ during the re-encapsulation part of decapsulation.

Both implementations included in the submission package use the secret-key format described in Algorithm 7, i.e., with polynomials in NTT domain, including the public key and H(pk), but not including the matrix $\mathbf{A}$.

Local storage format of static public keys. A user who is frequently encapsulating messages to the same public key can speed up encapsulation by locally storing an expanded public key containing the matrix $\mathbf{A}$ and H(pk). This saves the cost of expanding the matrix $\mathbf{A}$ from the seed ρ and the cost of hashing pk in every encapsulation.

2.2 Performance of reference and AVX2 implementations

Table 2 reports performance results of the reference implementation and of our implementation optimized using AVX2 vector instructions, also including performance results for the 90s variant of Kyber. All benchmarks were obtained on one core of an Intel Core i7-4770K (Haswell) processor clocked at 3500.000 MHz (as reported by `/proc/cpuinfo`) with TurboBoost and hyperthreading disabled. The benchmarking machine has 32 GB of RAM and is running Debian GNU/Linux with Linux kernel version 4.9.0. Both implementations were compiled with gcc version 6.3.0. We used compiler flags `-O3 -fomit-frame-pointer -march=native -fPIC`. to compile both implementations. All cycle counts reported are the median of the cycle counts of 10 000 executions of the respective function. The implementations are not optimized for memory usage, but generally Kyber has only very modest memory requirements. This means that in particular our implementations do not need to allocate any memory on the heap.

3 Known Answer Test values

All KAT values are included in subdirectories of the directory `KAT` of the submission package. Specifically, the KAT values of Kyber512 are in the subdirectory `KAT/kyber512`; the KAT values of the Kyber512-90s are in the subdirectory `KAT/kyber512-90s`; the KAT values of Kyber768 are in the subdirectory `KAT/kyber768`; the KAT values of the Kyber768-90s are in the subdirectory `KAT/kyber768-90s`; the KAT values of Kyber1024 are in the subdirectory `KAT/kyber1024`; and the KAT values of the Kyber1024-90s are in the subdirectory `KAT/kyber1024-90s`. Each of those directories contains the KAT values as generated by the `PQCgenKAT_kem` program provided by NIST. Specifically, those files are:

- `KAT/kyber512/PQCkemKAT_1632.req`,
- `KAT/kyber512/PQCkemKAT_1632.rsp`,
- `KAT/kyber512-90s/PQCkemKAT_1632.req`,
- `KAT/kyber512-90s/PQCkemKAT_1632.rsp`,
- `KAT/kyber768/PQCkemKAT_2400.req`,
- `KAT/kyber768/PQCkemKAT_2400.rsp`,
- `KAT/kyber768-90s/PQCkemKAT_2400.req`,
- `KAT/kyber768-90s/PQCkemKAT_2400.rsp`,
- `KAT/kyber1024/PQCkemKAT_3168.req`,
- `KAT/kyber1024/PQCkemKAT_3168.rsp`,
- `KAT/kyber1024-90s/PQCkemKAT_3168.req`, and
- `KAT/kyber1024-90s/PQCkemKAT_3168.rsp`,

Table 2: Key and ciphertext sizes and cycle counts for all paramter sets of Kyber. Cycle counts were obtained on one core of an Intel Core i7-4770K (Haswell); "ref" refers to the C reference implementation, "AVX2" to the implementation using AVX2 vector instructions; **sk** stands for secret key, **pk** for public key, and **ct** for ciphertext. In parenthesis are approximate values when including key generation in decapsulation to avoid having to store expanded secret keys. In this scenario, we only store the initial seed d in line 1 of Algorithm 4. The approximate cycle counts for this scenario are computed as the sum of cycle counts for standard decapsulation and key generation minus the number of cycles require to generate the matrix $\mathbf{A}$ from the public seed ρ. Note that this is a very conservative estimate; actual implementations of the approach can also save, for example, sampling the 32 bytes of randomness. See also the discussion on "tradeoffs between secret-key size and speed" in Subsection 2.1.

Kyber512					
Sizes (in Bytes)		**Haswell Cycles (ref)**		**Haswell Cycles (AVX2)**	
sk:	1632 (or 32)	**gen:**	118044	**gen:**	33428
pk:	800	**enc:**	161440	**enc:**	49184
ct:	736	**dec:**	190206 (or ≈ 279150)	**dec:**	40564 (or ≈ 62292)
Kyber512-90s					
Sizes (in Bytes)		**Haswell Cycles (ref)**		**Haswell Cycles (AVX2)**	
sk:	1632 (or 32)	**gen:**	232368	**gen:**	20004
pk:	800	**enc:**	285336	**enc:**	30384
ct:	736	**dec:**	313452 (or ≈ 436088)	**dec:**	24604 (or ≈ 40304)
Kyber768					
Sizes (in Bytes)		**Haswell Cycles (ref)**		**Haswell Cycles (AVX2)**	
sk:	2400 (or 32)	**gen:**	217728	**gen:**	62396
pk:	1184	**enc:**	272254	**enc:**	83748
ct:	1088	**dec:**	315976 (or ≈ 469008)	**dec:**	70304 (or ≈ 102652)
Kyber768-90s					
Sizes (in Bytes)		**Haswell Cycles (ref)**		**Haswell Cycles (AVX2)**	
sk:	2400 (or 32)	**gen:**	451018	**gen:**	30884
pk:	1184	**enc:**	514088	**enc:**	45892
ct:	1088	**dec:**	556972 (or ≈ 758934)	**dec:**	37844 (or ≈ 58668)
Kyber1024					
Sizes (in Bytes)		**Haswell Cycles (ref)**		**Haswell Cycles (AVX2)**	
sk:	3168 (or 32)	**gen:**	331418	**gen:**	88568
pk:	1568	**enc:**	396928	**enc:**	115952
ct:	1568	**dec:**	451096 (or ≈ 667596)	**dec:**	99764 (or ≈ 139552)
Kyber1024-90s					
Sizes (in Bytes)		**Haswell Cycles (ref)**		**Haswell Cycles (AVX2)**	
sk:	3168 (or 32)	**gen:**	735382	**gen:**	44040
pk:	1568	**enc:**	810398	**enc:**	64352
ct:	1568	**dec:**	860272 (or ≈ 1148394)	**dec:**	54448 (or ≈ 82444)

4 Expected security strength

4.1 Security definition

KYBER.CCAKEM (or short, KYBER) is an IND-CCA2-secure key encapsulation mechanism, i.e., it fulfills the security definition stated in Section 4.A.2 of the Call for Proposals.

4.2 Rationale of our security estimates

Our estimates of the security strength for the three different parameter sets of KYBER—and consequently the classification into security levels as defined in Section 4.A.5 of the Call for Proposals—are based on the cost estimates of attacks against the underlying module-learning-with-errors (MLWE) problem as detailed in Subsection 5.1.

To justify this rationale, we will in the following give two reductions from MLWE: a tight reduction in the random-oracle model (ROM) in Theorem 2 and a non-tight reduction in the quantum-random-oracle model (QROM) in Theorem 3. With those reductions at hand, there remain two avenues of attack that would break KYBER without solving the underlying MLWE problem, namely

1. breaking one of the assumptions of the reductions, in particular attacking the symmetric primitives used in KYBER; or
2. exploiting the non-tightness of the QROM reduction.

We briefly discuss 1.) in Subsection 5.2. The discussion of 2.) requires considering two separate issues, namely

- a (quadratic) non-tightness in the decryption-failure probability of KYBER.CPAPKE, and
- a (quadratic) non-tightness between the advantage of the MLWE attacker and the quantum attacker against KYBER.

In Subsection 5.3 we discuss quantum attacks exploiting decryption failures and in the presentation of the non-tight QROM reduction we explain why the non-tightness between quantum attacks against MLWE and quantum attacks against KYBER is unlikely to matter in practice. More specifically, we show how to eliminate this non-tightness if we allow the reasonable, but non-standard, assumption that KYBER.CPAPKE ciphertexts are pseudorandom, even if all randomness is generated pseudorandomly from a hash of the encrypted message.

4.3 Security Assumption

The hard problem underlying the security of our schemes is Module-LWE [24, 57]. It consists in distinguishing uniform samples $(\mathbf{a}_i, b_i) \leftarrow R_q^k \times R_q$ from samples $(\mathbf{a}_i, b_i) \in R_q^k \times R_q$ where $\mathbf{a}_i \leftarrow R_q^k$ is uniform and $b_i = \mathbf{a}_i^T \mathbf{s} + e_i$ with $\mathbf{s} \leftarrow B_\eta^k$ common to all samples and $e_i \leftarrow B_\eta$ fresh for every sample. More precisely, for an algorithm A, we define $\mathbf{Adv}^{\mathrm{mlwe}}_{m,k,\eta}(\mathsf{A}) =$

$$\left| \Pr\left[b' = 1 \ : \ \begin{array}{l} \mathbf{A} \leftarrow R_q^{m \times k}; (\mathbf{s}, \mathbf{e}) \leftarrow \beta_\eta^k \times \beta_\eta^m; \\ \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}; b' \leftarrow \mathsf{A}(\mathbf{A}, \mathbf{b}) \end{array} \right] - \Pr\left[b' = 1 \ : \ \mathbf{A} \leftarrow R_q^{m \times k}; \mathbf{b} \leftarrow R_q^m; b' \leftarrow \mathsf{A}(\mathbf{A}, \mathbf{b}) \right] \right|.$$

4.3.1 Tight reduction from MLWE in the ROM

We first note that KYBER.CPAPKE is tightly IND-CPA secure under the Module-LWE hardness assumption.

Theorem 1. *Suppose* XOF *and* G *are random oracles. For any adversary* A*, there exist adversaries* B *and* C *with roughly the same running time as that of* A *such that* $\mathbf{Adv}^{\mathrm{cpa}}_{\text{KYBER.CPAPKE}}(\mathsf{A}) \leq 2 \cdot \mathbf{Adv}^{\mathrm{mlwe}}_{k+1,k,\eta}(\mathsf{B}) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{PRF}}(\mathsf{C})$.

The proof of this theorem is easily obtained by noting that, under the MLWE assumption, public-key and ciphertext are pseudo-random.

KYBER.CCAKEM is obtained via a slightly tweaked Fujisaki-Okamoto transform [48, 40] applied to KYBER.CPAPKE. The following concrete security statement proves KYBER.CCAKEM's IND-CCA2-security when the hash functions G and H are modeled as random oracles. It is obtained by combining the generic bounds from [48] with Theorem 1 (and optimizing the constants appearing in the bound).

Theorem 2. *Suppose* XOF*, H, and G are random oracles. For any classical adversary* A *that makes at most* q_{RO} *many queries to random oracles* XOF*, H and G, there exist adversaries* B *and* C *of roughly the same running time as that of* A *such that*

$$\mathbf{Adv}^{\mathrm{cca}}_{\text{KYBER.CCAKEM}}(\mathsf{A}) \leq 2\mathbf{Adv}^{\mathrm{mlwe}}_{k+1,k,\eta}(\mathsf{B}) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{PRF}}(\mathsf{C}) + 4q_{RO}\delta.$$

Note that the security bound is tight. The negligible additive term $4q_{RO}\delta$ stems from KYBER.CPAPKE's decryption-failure probability δ.

4.3.2 Non-tight reduction from MLWE in the QROM

As for security in the quantum random oracle model (QROM), [48, 84] proved that KYBER.CCAKEM is IND-CCA2 secure in the QROM, provided that KYBER.CPAPKE is IND-CPA secure. A slightly tighter reduction can be obtained by requiring the base scheme KYBER.CPAPKE to be pseudo-random. Pseudo-randomness [84] requires that, for every message m, a (randomly generated) ciphertext $(c_1, c_2) \leftarrow \text{KYBER.CPAPKE.Enc}(pk, m)$ is computationally indistinguishable from a random ciphertext of the form $(\mathsf{Compress}_q(\mathbf{u}, d_u), \mathsf{Compress}_q(v, d_v))$, for uniform $(\mathbf{u}, v)$. (We also require the property of "statistical disjointness" [84] which is trivially fulfilled for KYBER.CPAPKE.) The proof of KYBER.CPAPKE's IND-CPA security indeed shows that KYBER.CPAPKE is tightly pseudo-random under the Module-LWE hardness assumption.

Theorem 3. *Suppose* XOF*, H, and G are random oracles. For any quantum adversary* A *that makes at most* q_{RO} *many queries to quantum random oracles* XOF*, H and G, there exist quantum adversaries* B *and* C *of roughly the same running time as that of* A *such that*

$$\mathbf{Adv}^{\mathrm{cca}}_{\text{KYBER.CCAKEM}}(\mathsf{A}) \leq 4q_{RO} \cdot \sqrt{\mathbf{Adv}^{\mathrm{mlwe}}_{k+1,k,\eta}(\mathsf{B})} + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{PRF}}(\mathsf{C}) + 8q^2_{RO}\delta.$$

Unfortunately, the above security bound is non-tight and therefore can only serve as an asymptotic indication of KYBER.CCAKEM's CCA-security in the quantum random oracle model.

Tight reduction under non-standard assumption. We can use [48, 84] to derive a tight security bound in the QROM from a non-standard security assumption, namely that a deterministic version of KYBER.CPAPKE, called DKYBER.CPAPKE, is pseudo-random in the QROM. Deterministic KYBER.CPAPKE is defined as KYBER.CPAPKE, but the random coins r used in encryption are derived deterministically from the message m as $r := \mathrm{G}(m)$. Pseudo-randomness for deterministic encryption states that an encryption (c_1, c_2) of a randomly chosen message is computationally indistinguishable from a random ciphertext $(\mathsf{Compress}_q(\mathbf{u}, d_u), \mathsf{Compress}_q(v, d_v))$, for uniform $(\mathbf{u}, v)$. In the classical ROM, pseudo-randomness of DKYBER.CPAPKE is tightly equivalent to MLWE but in the QROM the reduction is non-tight (and is the reason for the term $q_{RO} \cdot \sqrt{\mathbf{Adv}^{\mathrm{mlwe}}_{k+1,k,\eta}(\mathsf{B})}$ in Theorem 3). Concretely, we obtain the following bound:

$$\mathbf{Adv}^{\mathrm{cca}}_{\text{KYBER.CCAKEM}}(\mathsf{A}) \leq 2\mathbf{Adv}^{\mathrm{mlwe}}_{k+1,k,\eta}(\mathsf{B}) + \mathbf{Adv}^{\mathrm{pr}}_{\text{DKYBER.CPAPKE}}(\mathsf{C}) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{PRF}}(\mathsf{D}) + 8q^2_{RO}\delta.$$

We remark that we are not aware of any quantum attack on deterministic KYBER.CPAPKE that performs better than breaking the MLWE problem.

4.4 Estimated security strength

Table 3 lists the security levels according to the definition in Section 4.A.5 of the Call for Proposals for the different parameter sets of KYBER. Our claims are based on the cost estimates of the best known attacks

Table 3: Classical and quantum core-SVP hardness of the different proposed parameter sets of KYBER together with the claimed security level as defined in Section 4.A.5 of the Call for Proposals. Complexities are given in terms of the base-2 logarithm of the number of operations.

	core-SVP (classical)	core-SVP (quantum)	Claimed security level
KYBER512	111	100	**1** (AES-128)
KYBER768	181	164	**3** (AES-192)
KYBER1024	254	230	**5** (AES-256)

against the MLWE problem underlying KYBER as detailed in Subsection 5.1. Specifically we list the classical and the quantum *core-SVP hardness* and use those to derive security levels.

The impact of MAXDEPTH. The best known quantum speedups for the sieving algorithm, which we consider in our cost analysis (see Subsection 5.1.1), are only mildly affected by limiting the depth of a quantum circuit, because it uses Grover search on sets of small size (compared to searching through the whole keyspace of AES). For the core-SVP-hardness operation estimates to match the quantum gate cost of breaking AES at the respective security levels, a quantum computer would need to support a maximum depth of 70–80. When limiting the maximum depth to smaller values, or when considering classical attacks, the core-SVP-hardness estimates are smaller than the gate counts for attacks against AES. We discuss this difference in the following.

Gates to break AES vs. core-SVP hardness. The classical core-SVP hardness of the MLWE problem underlying KYBER differs by a factor of $\approx 2^{30}$ from the gate count to classically break the corresponding AES instances. The core-SVP hardness is a very conservative lower bound on the cost of an actual attack against the MLWE problem (for details, see Subsection 5.1). Specifically, the core-SVP-hardness ignores

- the (polynomial) number of calls to the SVP oracle that are required to solve the MLWE problem;
- the gate count required for one "operation";
- additional cost of sieving with asymptotically subexponential complexity;
- the cost of access into exponentially large memory; and
- the additional rounding noise (the LWR problem, see [13, 8]), *i.e.* the deterministic, uniformly distributed noise introduced in ciphertexts via the $\mathsf{Compress}_q$ function.

The state of research into SVP-solving algorithms is way too premature to assign meaningful cost estimates to each of those items. However, it seems clear that in any actual implementation of an attack algorithm the *product* of the cost of those items will exceed 2^{30}. See also the paragraph *"How conservative is this analysis?"* in Subsection 5.1.4.

4.5 Additional security properties

4.5.1 Forward secrecy.

KYBER has a very efficient key-generation procedure (see also Section 2) and is therefore particularly well suited for applications that use frequent key generations to achieve forward secrecy.

4.5.2 Side-channel attacks.

Timing attacks. Neither straight-forward reference implementations nor optimized implementations of KYBER use any secret-dependent branches or table lookups[2].This means that typical implementations of KYBER are free from the two most notorious sources of timing leakage. Another possible source of timing

[2]Note that the rejection sampling in generating the matrix $\mathbf{A}$ does not involve any secret data.

leakage are non-constant-time multipliers like the `UMULL` instruction on ARM Cortex-M3 processors, which multiplies two 32-bit integers to obtain a 64-bit result. However, multiplications in Kyber have only 16-bit inputs, and most non-constant-time multipliers show timing variation only for larger inputs. For example, on ARM Cortex-M3 processors the obvious way to implement multiplications in Kyber is through the constant-time `MUL` instruction, which multiplies two 32-bit integers, but returns only the bottom 32-bits of the result. What remains as a source of timing leakage are modular reductions, which are sometimes implemented via conditional statements. However, timing leakage in modular reductions is easily avoided by using (faster) Montgomery [67] and Barrett reductions [14] as illustrated in our reference and AVX2 implementations.

We note that the 90s variant is only really attractive to use if AES hardware support is available. If hardware support is not available, then table-based implementations of AES are notorious for leaking secrets trough cache timing. In our C reference implementation of the 90s variant we use a constant-time bitsliced implementation of AES, which is based on code from BearSSL [78].

Differential attacks. We expect that any implementation of Kyber without dedicated protection against differential power or electromagnetic radiation (EM) attacks will be vulnerable to such attacks. This is true for essentially any implementation of a cryptographic scheme that uses long-term (non-ephemeral) keys. Deployment scenarios of Kyber in which an attacker is assumed to have the power to mount such an attack require specially protected—typically masked—implementations. In [70], Oder, Schneider, Pöppelmann, and Güneysu present such a masked implementation of Ring-LWE decryption with a CCA transform very similar to the one used in Kyber. The implementation targets Cortex-M4F microcontrollers; the conclusion of the work is that protecting the decryption (decapsulation) step against first-order DPA incurs an overhead of about about a factor of 5.5. The techniques presented in that paper also apply to Kyber and we expect that the overhead for protecting Kyber against differential attacks is in the same ballpark.

Template attacks. Protections against differential attacks do not help if an attacker is able to recover even ephemeral secrets from a single power or EM trace. At CHES 2017, Primas, Pessl, and Mangard presented such a single-trace attack against an implementation of Ring-LWE on a Cortex-M4F microcontroller [79]. The attacker model in this attack is rather strong: it is the typical setting of template attacks, which assumes an attacker who is able to generate template traces on known inputs on a device with leakage very similar to the actual target device. In [79], the authors used *the same device* for generating target traces and in the attack. The attack was facilitated (maybe even enabled) by the fact that the implementation under attack used variable-time modular reductions. Consequently, the paper states that *"One of the first measures to strengthen an implementation against SPA attacks is to ensure a constant runtime and control flow"*. This is the case for all implementations of Kyber. The attack from [79] would thus certainly not straight-forwardly apply to implementations of Kyber, but more research is required to investigate whether also constant-time implementations of Kyber (and other lattice-based schemes) succumb to template attacks, and what the cost of suitable countermeasures is.

4.5.3 Multi-target attacks

Our security analysis makes no formal claims about security bounds in the multi-target setting. However, in the design of Kyber we made two decisions that aim at improving security against attackers targeting multiple users:

- We adopt the "against-all-authority" approach of re-generating the matrix $\mathbf{A}$ for each public key from NewHope [7]. This protects against an attacker attempting to break *many keys at the cost of breaking one key.*
- In the CCA transform (see Alg. 8) we hash the public key into the pre-key $\bar{K}$ and the coins r. Making the coins r dependent of the public key protects against precomputation attacks that attempt to break *one out of many keys.* For details, see Subsection 5.3.

4.5.4 Misuse resilience

The first, and most important, line of defense against misuse is the decision to make IND-CCA2 security non-optional. As discussed in Subsection 1.5, it would have been possible to achieve slightly shorter public

keys and ciphertexts, and faster decapsulation, in a CPA-secure variant of KYBER. Using IND-CCA2 security by default makes it safe to use KYBER with static keys and as a consequence also to re-use ephemeral keys for some time. What is *not* safe, is to reuse the same randomness in encapsulation, but that randomness is also not exposed to the outside by the API. The CCA transform has a second effect in terms of robustness: it protects against a broken implementation of the noise sampling. A rather peculiar aspect of LWE-based cryptography is that it will pass typical functional tests even if one communication partner does not add any noise (or by accident samples all-zero noise). The deterministic generation of noise via SHAKE-256 during encapsulation and the re-encryption step during decapsulation will reveal such an implementation mistake immediately.

An additional line of defense against misuse is to hash the public-key into the "pre-key" $\bar{K}$ and thus make sure that the KEM is contributory. Only few protocols require a KEM to be contributory and those protocols can always turn a non-contributory KEM into a contributory one by hashing the public key into the final key. Making this hash part of the KEM design in KYBER ensures that nothing will go wrong on the protocol level if implementers omit the hash there.

A similar statement holds for additionally hashing the ciphertext into the final key. Several protocols need to ensure that the key depends on the complete view of exchanged protocol messages. This is the case, for example, for the authenticated-key-exchange protocols described in the KYBER paper [22, Sec. 5]. Hashing the full protocol view (public key and ciphertext) into the final key already as part of the KEM makes it unnecessary (although of course still safe) to take care of these hashes on the higher protocol layer.

5 Analysis with respect to known attacks

5.1 Attacks against the underlying MLWE problem

MLWE as LWE. The best known attacks against the underlying MLWE problem in KYBER do not make use of the structure in the lattice. We therefore analyze the hardness of the MLWE problem as an LWE problem. We briefly discuss the current state of the art in *algebraic attacks*, i.e., attacks that exploit the structure of module lattices (or ideal lattices) at the end of this subsection.

5.1.1 Attacks against LWE.

Many algorithms exist for solving LWE (for a survey see [5]), but many of those are irrelevant for our parameter set. In particular, because there are only $m = (k+1)n$ LWE samples available to the attacker, we can rule out BKW types of attacks [51] and linearization attacks [11]. This essentially leaves us with two BKZ [85, 29] attacks, usually referred to as primal and dual attacks that we will recall in Subsections 5.1.2 and 5.1.3.

The algorithm BKZ proceeds by reducing a lattice basis using an SVP oracle in a smaller dimension b. It is known [46] that the number of calls to that oracle remains polynomial, yet concretely evaluating the number of calls is rather painful, and this is subject to new heuristic ideas [29, 28, 9]. We choose to ignore this polynomial factor, and rather evaluate only the *core SVP hardness*, that is the cost of *one call* to an SVP oracle in dimension b, which is clearly a pessimistic estimation (from the defender's point of view). This approach to deriving a conservative cost estimate for attacks against LWE-based cryptosystems was introduced in [7, Sec. 6].

Enumeration vs. sieving. There are two algorithmic approaches for the SVP oracle in BKZ: enumeration and sieving algorithms. These two classes of algorithms have very different performance characteristics and, in particular for sieving, it is hard to predict how *practical performance* scales from lattice dimensions that have been successfully tackled to larger dimensions that are relevant in attacks against cryptosystems like KYBER. The starting point of such an analysis is the fact that enumeration algorithms have super-exponential running time, while sieving algorithms have only exponential running time. Experimental evidence from typical implementations of BKZ [42, 29, 33] shows that enumeration algorithms are more efficient in "small" dimensions, so one question is at what dimension sieving becomes more efficient. So far it seems that sieving is slower in practice for accessible dimensions of up to $b \approx 130$. However, a recent work [35] showed (in the

classical setting) that sieving techniques can be sped up *in practice* for exact-SVP, being now less than an order of magnitude slower than enumeration already in dimension 60 to 80.

The analysis is complicated by the fact that sieving algorithms are much more *memory intensive* than enumeration algorithms. Specifically, sieving algorithms have exponential complexity not only in time, but also in memory, while enumeration algorithms require only small amounts of memory. In practice, the cost of access to memory increases with the size of memory, which typically only becomes noticeable once the memory requirement exceeds fast local memory (RAM). There is no study, yet, that investigates the algorithmic optimization and practical performance of sieving using slow background storage.

We follow the approach of [7, Sec. 6] to obtain a conservative lower bound on the performance of both sieving and enumeration for the dimensions that are relevant for the cryptanalysis of KYBER. This approach works in the RAM model, i.e., it assumes that access into even exponentially large memory is free. Under this assumption sieving becomes more efficient than even sophisticated enumeration, with serious optimization as described in [29] and with quantum speedups, for dimensions larger than 250, quite possibly already earlier. The smallest dimension that we are interested in for the cryptanalysis of KYBER is 390, so that the performance of sieving in the RAM model serves as a conservative lower bound for the performance of both enumeration and sieving.

A lot of recent work has pushed the efficiency of the original lattice sieve algorithms [69, 66], improving the heuristic complexity from $(4/3)^{b+o(b)} \approx 2^{0.415b}$ down to $\sqrt{3/2}^{b+o(b)} \approx 2^{0.292b}$ using *Locality Sensitive Hashing* (LSH) techniques [54, 16]. The hidden sub-exponential factor is known to be much greater than one in practice. Again, we ignore this factor to arrive at a security estimate with a conservative margin. Most of the sieving algorithms have been shown [55, 53] to benefit from Grover's quantum search algorithm, bringing the complexity down to $2^{0.265b}$. We will use $2^{0.292b}$ as the classical and $2^{0.265b}$ and the quantum cost estimate of both the primal and dual attacks with block size (dimension) b. We recall those two attacks in the following.

5.1.2 Primal attack.

The primal attack consists of constructing a unique-SVP instance from the LWE problem and solving it using BKZ. We examine how large the block dimension b is required to be for BKZ to find the unique solution. Given the matrix LWE instance $(\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s}+\mathbf{e})$ one builds the lattice $\Lambda = \{\mathbf{x} \in \mathbb{Z}^{m+kn+1} : (\mathbf{A}|\mathbf{I}_m|-\mathbf{b})\mathbf{x} = \mathbf{0} \bmod q\}$ of dimension $d = m+kn+1$, volume q^m, and with a unique-SVP solution $\mathbf{v} = (\mathbf{s}, \mathbf{e}, 1)$ of norm $\lambda \approx \varsigma\sqrt{kn+m}$. Note that the number of used samples m may be chosen between 0 and $(k+1)n$ in our case and we numerically optimize this choice.

Success condition. We model the behavior of BKZ using the geometric series assumption (which is known to be optimistic from the attacker's point of view), that finds a basis whose Gram-Schmidt norms are given by $\|\mathbf{b}_i^\star\| = \delta^{d-2i-1} \cdot \mathrm{Vol}(\Lambda)^{1/d}$, where $\delta = ((\pi b)^{1/b} \cdot b/2\pi e)^{1/2(b-1)}$ [28, 5]. The unique short vector $\mathbf{v}$ will be detected if the projection of $\mathbf{v}$ onto the vector space spanned by the last b Gram-Schmidt vectors is shorter than $\mathbf{b}_{d-b}^\star$. Its projected norm is expected to be $\varsigma\sqrt{b}$, that is the attack is successful if and only if

$$\varsigma\sqrt{b} \leq \delta^{2b-d-1} \cdot q^{m/d}. \tag{5}$$

We note that this analysis introduced in [7] differs and is more conservative than prior works, which were typically based on the hardness of unique-SVP estimates of [41]. The validity of the new analysis has been confirmed by further analysis and experiments in [4].

5.1.3 Dual attack

The dual attack consists of finding a short vector in the dual lattice $\mathbf{w} \in \Lambda' = \{(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^m \times \mathbb{Z}^{kn} : \mathbf{A}^t\mathbf{x} = \mathbf{y} \bmod q\}$. Assume we have found a vector $(\mathbf{x}, \mathbf{y})$ of length ℓ and compute $z = \mathbf{v}^t \cdot \mathbf{b} = \mathbf{v}^t\mathbf{A}\mathbf{s} + \mathbf{v}^t\mathbf{e} = \mathbf{w}^t\mathbf{s} + \mathbf{v}^t\mathbf{e} \bmod q$, which is distributed as a Gaussian of standard deviation $\ell\varsigma$ if $(\mathbf{A}, \mathbf{b})$ is indeed an LWE sample (otherwise it is uniform mod q). Those two distributions have maximal variation distance bounded by $\epsilon = 4\exp(-2\pi^2\tau^2)$, where $\tau = \ell\varsigma/q$, that is, given such a vector of length ℓ one has an advantage ϵ against decision-LWE.

Table 4: Classical and quantum core-SVP hardness of the MLWE problem (treated as LWE problem) underlying KYBER for different proposed parameter sets. The value b denotes the block dimension of BKZ (i.e., the dimension of the SVP considered in the core-SVP-hardness estimates), and m the number of used samples. Cost is given in $\log_2$ of operations and is the smallest cost for all possible choices of m and b.

	b	m	Core-SVP (classical)	Core-SVP (quantum)
KYBER512				
Primal attack:	385	410	112	102
Dual attack:	380	455	111	100
KYBER768				
Primal attack:	625	655	182	165
Dual attack:	620	650	181	164
KYBER1024				
Primal attack:	880	795	257	233
Dual attack:	870	795	254	230

The length ℓ of a vector given by the BKZ algorithm is given by $\ell = \|\mathbf{b}_0\|$. Knowing that Λ' has dimension $d = m + kn$ and volume q^{kn} we get $\ell = \delta^{d-1} q^{kn/d}$. Therefore, obtaining an ϵ-distinguisher requires running BKZ with block dimension b, where

$$-2\pi^2\tau^2 \geq \ln(\epsilon/4). \tag{6}$$

Note that small advantages ϵ are not relevant since the agreed key is hashed: an attacker needs an advantage of at least $1/2$ to significantly decrease the search space of the agreed key. He must therefore amplify his success probability by building about $1/\epsilon^2$ many such short vectors. Because the sieve algorithms provides $2^{0.2075b}$ vectors, the attack must be repeated at least R times where

$$R = \max(1, 1/(2^{0.2075b}\epsilon^2)).$$

This makes the conservative assumption that all the vectors provided by the Sieve algorithm are as short as the shortest one.

5.1.4 Core-SVP hardness of KYBER

In Table 4 we list the classical and quantum core-SVP-hardness of the three parameter sets of KYBER. The lower bounds of the cost of the primal and dual attack were computed following the approach outlined above using the analysis script `Kyber.py` that is available online at https://github.com/pq-crystals/kyber/tree/master/scripts/.

How conservative is this analysis? The core-SVP-hardness estimates give a lower bound on the cost of actual attacks rather than attempting to assign costs to various building blocks that require further study. To give an idea of how far this lower bound is from recent estimates that attempt to find a tighter bound, consider the example of the "BCNS" key exchange [23]. In [7, Table 6], the NEWHOPE paper computes the classical core-SVP hardness for the parameters used in [23] as 2^{86}. The claimed classical security level for those parameters in [23] is 2^{128}. Note that [7] does not contradict this claim, the factor of $2^{128}/2^{86} = 2^{42}$ rather indicates how conservative the core-SVP hardness estimate is. As a second example, consider another commonly used tool for estimating (classical) security of LWE-based cryptosystems, namely the `lwe-estimator` script by Albrecht [1]. Applied to the KYBER parameter sets it estimates a classical security of 2^{140} for KYBER512, 2^{211} for KYBER768, and 2^{285} for KYBER1024. Note that also these estimates count number of "operations" rather than gates, and are in the RAM model, i.e., ignore the cost of memory access for sieving.

5.1.5 Algebraic attacks.

While the best known attacks against the MLWE instance underlying Kyber do not make use of the structure in the lattice, we still discuss the current state of the art of such attacks. Most noticeably, several recent works propose new quantum algorithms against Ideal-SVP [37, 27, 20, 31, 32], i.e., solving the shortest vector problem in ideal lattices. The work of [32] mentions obstacles towards a quantum attack on Ring-LWE from their new techniques, but nevertheless suggests using Module-LWE, as it plausibly creates even more obstacles. In [2], Albrecht and Deo establish a reduction from MLWE to RLWE, whose implication is that a polynomial-time algorithm against RLWE with certain parameters would translate to a polynomial-time algorithm against MLWE. In practical terms, however, this attack has a significant slow-down (and this is not just due to the proof) as the dimension of the module increases. This does suggest that increasing the dimension of the module may make the scheme more secure in concrete terms. In particular, going through this reduction to attack Kyber768 would lead to an RLWE problem with quite large modulus and error ($q' = q^3$, $\varsigma' > q^2\varsigma$), and therefore require the attacker to consider more than 1 sample: the underlying lattice remains a module with a rank strictly larger than 2.

5.2 Attacks against symmetric primitives

All symmetric building blocks of Kyber are instantiated with functions derived from Keccak [19]. In the deterministic expansion of $\mathbf{A}$ from ρ we essentially need SHAKE-128 to produce output that "looks uniformly random" and does not create any backdoors in the underlying lattice problem. In the noise generation we require that concatenating a secret and a public input and feeding this concatenation to SHAKE-256 as input results in a secure pseudorandom function. Breaking any of these properties of SHAKE would be a major breakthrough in the cryptanalysis of SHAKE, which would require replacing SHAKE inside Kyber by another XOF.

The security proofs model SHAKE-128, SHA3-256, and SHA3-512 as random oracles, i.e., they are subject to the standard limitations of proofs in the (quantum-)random-oracle model. Turning these limitations into an attack exploiting the instantiation of XOF, H, or G with SHAKE and SHA3 would again constitute a major breakthrough in the understanding of either Keccak or random-oracle proofs in general.

5.3 Attacks exploiting decryption failures

All parameter sets of Kyber have a decapsulation-failure probability δ of somewhat below 2^{-160}; for the examples in the remainder of this subsection we assume the failure probability of 2^{-164} of Kyber768. In Theorems 2 and 3 we see that this failure probability plays a role in the attacker's advantage: in the classical context in the term $4q_{RO}\delta$ and in the quantum context in the term $8q_{RO}^2\delta$, where q_{RO} is the number of queries to the (classical or quantum) random oracle.

Attacks exploiting failures. This term in the attacker's advantage is not merely a proof artifact, it can be explained by the following attack: An attacker searches through many different values of m (see line 1 of Alg. 8) until he finds one that produces random coins r (line 3 of Alg. 8) that lead to a decapsulation failure, which will give the attacker information about the secret key. In the quantum setting the search through different values of m is accelerated by Grover's algorithm, which explains the square in the term q_{RO}^2. With this attack in mind note that with 2^{64} ciphertexts (cmp. Section 4.A.2 of the Call for Proposals), there is a chance of 2^{-100} of a decapsulation failure without any particular effort by the attacker.

The effect on Kyber. The attack sketched in the previous paragraph is based on two assumptions that do not hold for Kyber: First it requires the capability to determine *offline* (e.g., as part of the Grover oracle) if a certain value of r produces a decapsulation failure. Second it assumes that one decapsulation failure seriously threatens the secrecy of the private key. Concerning the first assumption, an attacker cannot determine offline whether a given value of r, or more specifically, the derived values $\mathbf{r}$ (line 9 of Alg. 5) and $\mathbf{e}_1$ (line 13 of Alg. 5), produce a decapsulation failure. The reason is that the probability of decapsulation failures largely depends on the products $\mathbf{s}^T\mathbf{e}_1$ and $\mathbf{e}^T\mathbf{r}$ and the attacker does not now the values of $\mathbf{s}$ and $\mathbf{e}$. A quantum attacker can try to use Grover search to precompute values of m that have a slightly higher chance to produce a failure; as the attacker does not know the signs of the coefficients of $\mathbf{e}_1$ and $\mathbf{s}$, the best strategy is probably to search for values of m that produce $\mathbf{e}_1$ and $\mathbf{r}$ with above-average norm. The gain

achieved through such an approach is limited due to the fact that the distribution of a high-dimensional Gaussian is tightly concentrated around its expected value, while that of a 1-dimensional Gaussian is not as tightly concentrated around its mean.

The polynomial pair $(\mathbf{e}_1, \mathbf{r})$ can be seen as a vector in $\mathbb{Z}^{1536}$ distributed as a discrete Gaussian with standard deviation $\sigma = \sqrt{\eta/2} = 1$. By standard tail bounds on discrete Gaussians [12], we know that an m-dimensional vector $\mathbf{v}$ drawn from a discrete Gaussian of standard deviation σ will satisfy

$$\Pr[\|\mathbf{v}\| > \kappa\sigma\sqrt{m}] < \kappa^m \cdot e^{\frac{m}{2}(1-\kappa^2)}, \tag{7}$$

for any $\kappa > 1$.

So for example, the probability of finding a vector which is of length $1.33 \cdot \sigma\sqrt{1536}$ is already as small as 2^{-220}. Even if Grover's algorithm reduces the search space and increases the probability to 2^{-110}, finding such a vector merely increases the chances of getting a decryption error; and the probability increase is governed by the tail-bounds for 1-dimensional Gaussians.[3] For any vector $\mathbf{v}$, if $\mathbf{z}$ is chosen according to a Gaussian with standard deviation σ, then for any κ,

$$\Pr[|\langle \mathbf{z}, \mathbf{v}\rangle| > \kappa\sigma\|\mathbf{v}\|] \leq 2e^{-\kappa^2/2}. \tag{8}$$

If originally, the above probability is set so that decryption errors occur with probability $\approx 2^{-160}$, then $\kappa \approx 15$.[4] If the adversary is then able to increase $\|\mathbf{v}\|$ by a factor of 1.33 (by being able to find larger $(\mathbf{e}_1, \mathbf{r})$), then we can decrease κ by a factor of 1.33 to ≈ 11.25 in (8), which would still give us a probability of a decryption error of less than 2^{-90}. However, finding such a large $\mathbf{v}$ would take at least 2^{110} time, which would make the whole attack cost at least 2^{200}.

Of course one can try to find a slightly smaller $\mathbf{v}$ in the first step so that the entire attack takes less time. If Grover's algorithm really saves a square-root factor, then the optimal value is ≈ 1.05 for κ in (7), which would allow us to lower κ by a factor of 1.05 in (8) to $15/1.05 \approx 14.28$, and would still give a total time to find one decryption error $\approx 2^{-150}$. This makes the attack completely impractical.

Furthermore, a single decapsulation failure in Kyber does not allow an attacker to recover much information about the secret key $\mathbf{s}$. To get an intuition for the amount of information obtained from failures, consider the attack described in [39]. This is the standard attack that exploits failures in RLWE key encapsulation schemes that reuse keys without the CCA transform. In this scenario the attacker can *adaptively choose arbitrary noise*, i.e., set failure probabilities to arbitrary values and maximize the information obtained from each failure or non-failure. The paper concludes that attacking RLWE key exchange in lattice dimension 1024 in this setting *"can be done with perhaps* $4,000$ *queries"*. It seems extremely unlikely that even 10 decapsulation failures in Kyber would allow an attacker to recover any meaningful information about the secret key $\mathbf{s}$. Note that the probability of f decapsulation failures in 2^{64} ciphertexts is about $2^{-(e-64)f}$ where 2^{-e} is the largest probability of failure an attacker can achieve for one ciphertext. We have established that even with a Grover search making 2^{110} calls to the hash function, an attacker can only get $e > 90$. This very loose analysis shows that an attacker can't reasonably hope to produce more than two or three failures in less than 2^{128} time. We therefore conclude that the decryption failures do not introduce any weaknesses into Kyber.

Multitarget attacks using failures. Despite the limited gain, an attacker could consider using Grover's algorithm to precompute values of m that produce $\mathbf{r}$ and $\mathbf{e}_1$ with large norm and then use this precomputed set of values of m against many users. This multi-target attack is prevented by hashing the public key pk into the random coins r and thereby into $\mathbf{r}$ and $\mathbf{e}_1$ (line 3 of Alg. 8).

[3] The decryption noise is generated as an inner product of two vectors, and the distribution of this inner product closely resembles the Gaussian distribution.

[4] The above formula only roughly approximates how the decryption error is calculated where $\mathbf{z}$ corresponds to the secret key $(\mathbf{s}, \mathbf{e})$. We should also point out that a part of the decryption error in Kyber is caused by the rounding function Compress, which the adversary has no control over. Therefore this attack will be even less practical than what we describe.

6 Advantages and limitations

6.1 Advantages

In addition to the very competitive speeds, small parameters, and being based on a well-studied problem, the unique advantages of Kyber are:

Ease of implementation: Optimized implementations only have to focus on a fast dimension-256 NTT and a fast Keccak permutation. This will give very competitive performance *for all parameter sets of* Kyber.

Scalability: Switching from one Kyber parameter set to another only requires changing the matrix dimension (i.e., a `#define` in most C implementations) and the noise sampling.

We will now give a brief comparison of Kyber to other types of post-quantum schemes (that we are aware of) and, more importantly, to other manners in which lattice-based schemes could be instantiated.

6.2 Comparison to SIDH

An interesting alternative to lattice-based KEMs is supersingular-isogeny Diffie-Hellman (SIDH) [50]. The obvious advantage of SIDH is the sizes of public keys and ciphertexts that—with suitable compression [30]—are about a factor of 3 smaller than Kyber's public keys and ciphertexts. The downside of SIDH is that it is more than 2 orders of magnitude slower than Kyber. The scheme is also rather new, which makes it hard to make definitive comparisons. In the coming years, both implementation speeds and (quantum) attacks against SIDH can improve which may result in faster schemes and/or larger parameters.

6.3 Comparison to code-based KEMs

When considering code-based KEMs, one needs to distinguish the "classical" McEliece and Niederreiter schemes based on binary Goppa codes, and schemes with a less conservative (but more efficient) choice of code. A KEM based on binary Goppa codes can reasonably claim to be a very conservative choice of post-quantum primitive; however, its deployment will, in many scenarios, be hampered by massive public-key size and key-generation time. Less conservative choices, like quasi-cyclic medium-density parity-check (QC-MDPC) codes, are a closer competition in terms of performance but suffer from the fact that for efficient parameters at high security levels they do not achieve (provably) negligible failure probability, which precludes their use in CCA-secure KEMs.

6.4 Comparison to other lattice-based schemes

There are certain design choices that one can make when designing lattice-based schemes, some of which can have significant effects on the efficiency of the resulting scheme and on the underlying security assumption. Below we list the most important ones and explain the advantages / disadvantages of them versus what we chose for Kyber.

6.4.1 Schemes that build a KEM directly

The Kyber KEM is constructed by encrypting a random message using the LPR encryption [61] (with "bit-dropping"). Another approach one could take is directly building a KEM using the slightly different ideas described in [34, 73]. The advantage of the constructions in [34, 73] over our approach is that if one were to construct a CPA-secure KEM transmitting a b-bit key, then the ciphertext would be b bits shorter, which is about a 3% saving for typical parameters [58]. If, however, one wishes to construct a CCA-secure KEM like Kyber, then this advantage disappears since transformations from CPA-secure KEMs to CCA-secure ones implicitly go through a CPA-secure encryption scheme, which will result in adding b bits to the KEM. This is why, in Kyber, we simply use the LPR encryption scheme (instead of the CPA-secure key encapsulation) to define Kyber.CPAPKE, and then use this as a building block to construct the IND-CCA2-secure KEM Kyber.CCAKEM. Since there is virtually no difference between the two approaches, we will not draw a distinction between schemes constructed in either manner throughout the rest of this section.

6.4.2 LWE based schemes

If one does not want to use any algebraic structure in the LWE problem (i.e. if one takes the MLWE problem over the ring $\mathbb{Z}$), then there are two possibilities for constructing encryption or key-exchange schemes. The first approach makes the public key and the secret key very large (on the order of Megabytes), while keeping the ciphertext at essentially the same size as in Kyber. This type of scheme is the [75] version of the original Regev scheme from [81]. Because of the very large public-key size, this scheme would be extremely inefficient as a key exchange. A scheme more amenable to key exchange is [21], whose public key and ciphertext sizes are both approximately 11 KB each, which is approximately 10 times larger than in Kyber. The running time of each party is also larger by a factor of at least 10. In short, LWE-based schemes do not have any ring structure but are an order of magnitude slower and larger than Kyber. They are good back-up schemes in case algebraic structure in lattice schemes could somehow be devastatingly exploited by attackers.

6.4.3 Ring-LWE based schemes

The other extreme in the LWE design space are Ring-LWE (RLWE) schemes based on [61] (e.g., [7]). RLWE is a special case of the MLWE problem where the width of the matrix $\mathbf{A}$ over the ring R is always 1 (and typically, its height would be 2 for a PKE or KEM scheme). Varying the hardness of an RLWE scheme therefore requires to change the dimension of the ring, whereas in Kyber, the ring is always the same and the dimension of the module is being varied. As we mentioned above, one advantage of the approach we chose for Kyber is that we only need to have one good implementation for operations over the ring; varying the dimension of the module simply involves doing more (or fewer) of the same ring operations. Changing the ring, on the other hand, would require completely re-implementing all the operations.

Another advantage of working with a constant-degree "small" ring is that it enables more fine-grained tradeoffs between performance and security. The simplest and most efficient way of implementing RLWE is to work over rings $\mathbb{Z}[X]/(X^n + 1)$ where n is a power of 2. Since n is the only parameter that determines the efficiency and security of RLWE schemes, limiting it to powers of 2 may require overshooting the needed security bound. For example, the dimension of Kyber768 is not reachable. One could of course work directly modulo a polynomial of any desired degree (with the main restriction being that it has to be irreducible over $\mathbb{Z}$), but then the security would decrease slightly due the geometry of non-power-of-2 number fields (see [64, 65]).

The one advantage of RLWE over Kyber is that if $\mathbf{A}$ is a $k \times k$ matrix, then extracting it from a seed requires k times more XOF output than for a 1×1 matrix.

6.4.4 NTRU

When compared to Kyber, NTRU [47] has all the advantages and disadvantages of RLWE, but in addition has two further negative points against it. First NTRU key generation is considerably more expensive than in RLWE when the ring does not support NTT. The reason is that NTRU key generation requires polynomial division, whereas RLWE key generation requires only multiplication (if the ring supports NTT, then division is not much slower than multiplication). The second possible downside of NTRU is that the geometry of its underlying lattice leads to attacks that do not exist against RLWE or MLWE schemes [52]. While this property does not seem to aid in attacks against the small parameters that are used for defining NTRU cryptosystems, it may point to a possible weakness that could be further exploited. The one possible advantage of using NTRU is a small performance advantage during encryption (encapsulation), but given the disadvantages we do not consider this a good tradeoff. Furthermore, it is not possible to define an efficient version of "Module-NTRU" that would allow for the advantages of Kyber described above in Section 6.1.

6.4.5 Different Polynomial Rings

One could consider using Kyber with a ring that is not $\mathbb{Z}[X]/(X^n + 1)$. An argument that could be made for using different rings is that the rings currently used in Kyber have algebraic properties (e.g., subrings, large Galois groups, etc.) which may be exploited in attacks. We choose to work with $\mathbb{Z}[X]/(X^n + 1)$ for the following reasons:

- From a performance perspective there is no serious competition; the NTT-based multiplication supported by the parameters we chose for KYBER is at the same time very memory efficient and faster than any other algorithm for multiplication in polynomial rings.
- Lattice-based schemes using the ring $\mathbb{Z}[X]/(X^n+1)$ have been studied since at least [59]. When the noise vectors are chosen as specified in [61], there have been no improved attacks against RLWE (or MLWE) that use the underlying algebraic structure [74]. Furthermore, being based on MLWE, the algebraic structure of KYBER is very different from that which was exploited in the attacks against *ideal* lattices in [20, 31, 32][5] – we emphasize that the lattice problems underlying the hardness of KYBER are *not* ideal lattices.
- Some of the additional algebraic structure of $\mathbb{Z}[X]/(X^n+1)$ is actually *helpful against* certain possible attack vectors. As a simple example, it can be proved that when X^n+1 (almost) fully splits modulo q, there do not exist polynomials in the ring that have small norm and many zeros in the NTT representation—the existence of such polynomials for *any* q would weaken the security of MLWE.
- Finally, $\mathbb{Z}[X]/(X^n+1)$ is one of the most widely studied, and best understood, rings (along with other cyclotomic rings) in algebraic number theory. The fact that no attacks have been found against its use for cryptosystems like KYBER makes it a much more conservative choice than some ring that is harder to analyze and may show weaknesses only after many more years of study.

6.4.6 Deterministic Noise.

Instead of adding noise $\mathbf{e}, \mathbf{e}_1$, and e_2, one can add "deterministic" noise by simply dropping bits. This is the basis behind the "Learning with Rounding" (LWR) problem [13], which for certain parameters is as hard as the LWE problem. We believe that asymptotically this is a sound approach but the number of bits that can be dropped before significant decryption error is introduced is not very large (≈ 2) in certain places in the scheme. This may allow for a possibility of slightly improved attacks against the scheme. Since generating noise is not a particularly costly operation, we did not choose to potentially weaken the scheme to save a little time.

7 Brief discussion of relevant results since Nov. 2017

In this section we briefly reference and comment on results relevant to KYBER that were published after the original NIST PQC submission deadline in November 2017.

"small KYBER". The idea of decreasing the modulus q from $q = 7681$ to $q = 3329$ for KYBER was already present in [86]; the paper refers to the resulting parameter set as "small-KYBER". Other parameters chosen in small-KYBER are different than what we describe in this round-2 update; in particular, small-KYBER does not eliminate the public-key compression. More importantly, the polynomial multiplication using the NTT algorithm is different in small-KYBER. Our multiplication algorithm uses the NTT algorithm to compute the decomposition of polynomials $a_i \in \mathbb{Z}_q[X]/(X^{256}+1)$ as

$$(a_i \bmod X^2 - r_1, \ldots, a_i \bmod X^2 - r_{128}),$$

and multiplication is performed using pointwise products modulo $X^2 - r_j$. The algorithm in [86], on the other hand, divides the a_i into two polynomials of degree 128 and performs a "full-splitting" NTT (i.e. modulo $X - r_i'$) on each of them. This method, which computes two (smaller) NTTs and some additional steps to reassemble the polynomials, has slightly worse performance than the original KYBER and considerably worse than KYBER with our round-2 tweaks; though we believe that applying our new tweaks would speed up the algorithm in [86] as well.

Fault attacks. In [80], Ravi, Roy, Bhasin, Chattopadhyay, and Mukhopadhyay describe a fault attack against implementations of NewHope, KYBER, Frodo, and Dilithium on an ARM Cortex-M4. Specifically, these attacks target the instructions loading the nonce in the PRF; skipping this load through a clock glitch

[5] Also, like the attacks against NTRU, these do not apply for the small parameters used public key encryption schemes.

reuses the same nonce multiple times, which leads to efficiently breakable instances of the schemes. We could have decided to modify the *specification* of KYBER to generate randomness in a different way. Indeed, generating all randomness at once through one call to the PRF would thwart this specific attack and *maybe* aid future fault-attack protected implementations. However, such a change to the specification would reduce parallelism and thus cost performance on many platforms. More importantly however, the benefits of such a change are still completely unclear. It is not at all surprising that an implementation without any fault-attack countermeasures, such as the one targeted in [80], succumbs to *some* fault attack. It is not clear if the attack described in [80] is the most powerful attack and it is also unclear what serious fault-attack countermeasures for any of the targeted schemes have to look like to thwart *any* realistic attack.

Attacks exploiting decapsulation failures. In [36], D'Anvers, Vercauteren, and Verbauwhede present a detailed analysis of attacks exploiting decapsulation failures. The paper essentially confirms the analysis we give in Section 5.3, namely that failures that occur with negligibly small probability are no concern for security. For example, the best attack the paper presents against KYBER768 collects 42 decapsulation failures from 2^{131} decapsulation queries and takes time 2^{142}. Note that these numbers are for KYBER without the round-2 tweaks; the parameters we propose for round-2 have even lower failure probability.

In [15], Bauer, Gilbert, Renault, and Rossi revisit CCA attacks against schemes that only offer CPA security (more specifically, against NewHope-CPAKEM). The attacks presented show once more how important CCA security is when not using purely ephemeral secrets.

References

[1] Martin Albrecht. Security estimates for the learning with errors problem, 2017. Version 2017-09-27, https://bitbucket.org/malb/lwe-estimator. 25

[2] Martin Albrecht and Amit Deo. Large modulus Ring-LWE ≥ Module-LWE, 2017. To appear. https://eprint.iacr.org/2017/612. 26

[3] Martin R. Albrecht, Amit Deo, and Kenneth G. Paterson. Cold boot attacks on ring and module LWE keys under the NTT. *Transactions on Cryptographic Hardware and Embedded Systems*, (3):173–213, 2018. https://doi.org/10.13154/tches.v2018.i3.173-213. 16

[4] Martin R Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, volume 10211 of *LNCS*, pages 65–102. Springer, 2017. https://eprint.iacr.org/2017/815. 24

[5] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015. https://eprint.iacr.org/2015/046. 23, 24

[6] Michael Alekhnovich. More on average case vs approximation complexity. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 298–307, 2003. 8

[7] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange – a new hope. In *Proceedings of the 25th USENIX Security Symposium*, pages 327–343. USENIX Association, 2016. http://cryptojedi.org/papers/#newhope. 9, 13, 14, 16, 22, 23, 24, 25, 29

[8] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, volume 8042 of *LNCS*, pages 57–74. Springer, 2013. https://eprint.iacr.org/2013/098. 21

[9] Yoshinori Aono, Yuntao Wang, Takuya Hayashi, and Tsuyoshi Takagi. Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, volume 9665 of *LNCS*, pages 789–819. Springer, 2016. https://eprint.iacr.org/2016/146. 23

[10] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, pages 595–618, 2009. 8

[11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzingeri, and Jiří Sgall, editors, *Automata, Languages and Programming*, volume 6755 of *LNCS*, pages 403–415. Springer, 2011. https://www.cs.duke.edu/~rongge/LPSN.pdf. 23

[12] Wojciech Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 296(1):625–635, 1993. 27

[13] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 719–737. Springer, 2012. http://www.iacr.org/archive/eurocrypt2012/72370713/72370713.pdf. 9, 14, 21, 30

[14] Paul Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer-Verlag Berlin Heidelberg, 1987. https://link.springer.com/chapter/10.1007/3-540-47721-7_24. 22

[15] Aurélie Bauer, Henri Gilbert, Guénaël Renault, and Mélissa Rossi. Assessment of the key-reuse resilience of newhope. IACR Cryptology ePrint Archive report 2019/075, 2019. https://eprint.iacr.org/2019/075.pdf. 31

[16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *SODA '16 Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete Algorithms*, pages 10–24. SIAM, 2016. https://eprint.iacr.org/2015/1128. 24

[17] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. The security impact of a new cryptographic library. In Alejandro Hevia and Gregory Neven, editors, *Progress in Cryptology – LATINCRYPT 2012*, volume 7533 of *LNCS*, pages 159–176. Springer, 2012. http://cryptojedi.org/papers/#coolnacl. 15

[18] Daniel J. Bernstein, Peter Schwabe, and Gilles Van Assche. Tweetable FIPS 202, 2015. https://keccak.team/2015/tweetfips202.html (accessed 2017-11-29). 15

[19] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference. Submission to the NIST SHA-3 competition, 2011. https://keccak.team/files/Keccak-reference-3.0.pdf. 14, 26

[20] Jean-François Biasse and Fang Song. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In *SODA '16 Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete Algorithms*, pages 893–902. SIAM, 2016. http://fangsong.info/files/pubs/BS_SODA16.pdf. 26, 30

[21] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In *CCS '16 Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1006–1018. ACM, 2016. https://eprint.iacr.org/2016/659. 13, 29

[22] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018*. IEEE, 2018. To appear. https://eprint.iacr.org/2017/634. 5, 12, 23

[23] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy*, pages 553–570, 2015. https://eprint.iacr.org/2014/599. 13, 14, 25

[24] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS '12 Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325. ACM, 2012. https://eprint.iacr.org/2011/277. 19

[25] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC '13 Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 575–584. ACM, 2013. http://arxiv.org/pdf/1306.0281. 14

[26] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, Gauss, and Reload – a cache attack on the BLISS lattice-based signature scheme. In Benedikt Gierlichs and Axel Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, volume 9813 of *LNCS*, pages 323–345. Springer, 2016. https://eprint.iacr.org/2016/300. 14

[27] Peter Campbell, Michael Groves, and Dan Shepherd. Soliloquy: A cautionary tale. In *ETSI 2nd Quantum-Safe Crypto Workshop*, pages 1–9, 2014. https://docbox.etsi.org/workshop/2014/201410_CRYPTO/S07_Systems_and_Attacks/S07_Groves_Annex.pdf. 26

[28] Yuanmi Chen. *Lattice reduction and concrete security of fully homomorphic encryption.* PhD thesis, l'Université Paris Diderot, 2013. http://www.di.ens.fr/~ychen/research/these.pdf. 23, 24

[29] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, 2011. http://www.iacr.org/archive/asiacrypt2011/70730001/70730001.pdf. 23, 24

[30] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. Efficient compression of SIDH public keys. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, volume 10210 of *LNCS*, pages 679–706. Springer, 2017. https://eprint.iacr.org/2016/963. 28

[31] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, volume 9666 of *LNCS*, pages 559–585. Springer, 2016. https://eprint.iacr.org/2015/313. 26, 30

[32] Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. Short Stickelberger class relations and application to Ideal-SVP. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, volume 10210 of *LNCS*, pages 324–348. Springer, 2017. https://eprint.iacr.org/2016/885. 26, 30

[33] The FPLLL development team. fplll, a lattice reduction library. Available at https://github.com/fplll/fplll, 2017. 23

[34] Jintai Ding, Xiang Xie, and Xiaodong Lin. A simple provably secure key exchange scheme based on the learning with errors problem. IACR Cryptology ePrint Archive report 2012/688, 2012. https://eprint.iacr.org/2012/688. 28

[35] Léo Ducas. Shortest vector from lattice sieving: a few dimensions for free. IACR Cryptology ePrint Archive report 2017/999, 2017. https://eprint.iacr.org/2017/999. 23

[36] Jan-Pieter D'Anvers, Frederik Vercauteren, and Ingrid Verbauwhede. On the impact of decryption failures on the security of LWE/LWR based schemes. IACR Cryptology ePrint Archive report 2018/1089, 2018. https://eprint.iacr.org/2018/1089. 31

[37] Kirsten Eisenträger, Sean Hallgren, Alexei Kitaev, and Fang Song. A quantum algorithm for computing the unit group of an arbitrary degree number field. In *STOC '14 Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 293–302. ACM, 2014. http://www.personal.psu.edu/kxe8/unitgroup.pdf. 26

[38] Thomas Espitau, Pierre-Alain Fouque, Benoït Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In *CCS '17 Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1857–1874. ACM, 2017. https://eprint.iacr.org/2017/505. 14

[39] Scott Fluhrer. Cryptanalysis of ring-LWE based key exchange with key share reuse. IACR Cryptology ePrint Archive report 2016/085, 2016. https://eprint.iacr.org/2016/085. 27

[40] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology - CRYPTO '99*, pages 537–554, 1999. https://link.springer.com/chapter/10.1007/3-540-48405-1_34. 5, 11, 20

[41] Nicolas Gama and Phong Nguyen. Predicting lattice reduction. In Nigel Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 31–51. Springer, 2008. https://www.iacr.org/archive/eurocrypt2008/49650031/49650031.pdf. 24

[42] Nicolas Gama, Phong Q Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 257–278. Springer, 2010. http://www.iacr.org/archive/eurocrypt2010/66320257/66320257.pdf. 23

[43] Matthew Gretton-Dann. Introducing 2017's extensions to the Arm architecture, 2017. https://community.arm.com/processors/b/blog/posts/introducing-2017s-extensions-to-the-arm-architecture. 16

[44] Tim Güneysu, Tobias Oder, Thomas Pöppelmann, and Peter Schwabe. Software speed records for lattice-based signatures. In Philippe Gaborit, editor, *Post-Quantum Cryptography*, volume 7932 of *LNCS*, pages 67–82. Springer, 2013. http://cryptojedi.org/papers/#lattisigns. 16

[45] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys, 2008. https://www.usenix.org/legacy/event/sec08/tech/full_papers/halderman/halderman.pdf. 16

[46] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Terminating BKZ. IACR Cryptology ePrint Archive report 2011/198, 2011. https://eprint.iacr.org/2011/198. 23

[47] Jeffrey Hoffstein, Jull Pipher, and Joseph H. Silverman. NTRU: a ring-based public key cryptosystem. In Joe P. Buhler, editor, *Algorithmic number theory*, volume 1423 of *LNCS*, pages 267–288. Springer, 1998. https://www.securityinnovation.com/uploads/Crypto/ANTS97.ps.gz. 8, 29

[48] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, LNCS, pages 341–371. Springer, 2017. https://eprint.iacr.org/2017/604. 15, 20

[49] Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *LNCS*, pages 150–169. Springer, 2007. http://www.iacr.org/archive/crypto2007/46220150/46220150.pdf. 14

[50] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography – PQCrypto 2011*, volume 7071 of *LNCS*, pages 19–34. Springer, 2011. https://eprint.iacr.org/2011/506. 28

[51] Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, volume 9215 of *LNCS*, pages 43–62. Springer, 2015. http://www.iacr.org/archive/crypto2015/92160264/92160264.pdf. 23

[52] Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, volume 10210 of *LNCS*, pages 3–26. Springer, 2017. https://www.di.ens.fr/~fouque/euro17a.pdf. 29

[53] Thijs Laarhoven. *Search problems in cryptography*. PhD thesis, Eindhoven University of Technology, 2015. http://www.thijs.com/docs/phd-final.pdf. 24

[54] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In Rosiario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, volume 9216 of *LNCS*, pages 3–22. Springer, 2015. http://www.iacr.org/archive/crypto2015/92160123/92160123.pdf. 24

[55] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77(2):375–400, 2015. https://eprint.iacr.org/2014/907. 24

[56] Adam Langley. Maybe skip SHA-3. Blog post on ImperialViolet, 2017. https://www.imperialviolet.org/2017/05/31/skipsha3.html. 16

[57] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015. https://eprint.iacr.org/2012/090. 5, 13, 19

[58] Vadim Lyubashevsky. Standardizing lattice crypto and beyond. Slides of the talk given by Vadim Lyubashevsky at PQCrypto 2017, 2017. https://2017.pqcrypto.org/conference/slides/pqc_2017_lattice.pdf. 28

[59] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFT: A modest proposal for FFT hashing. In Kaisa Nyberg, editor, *Fast Software Encryption – FSE 2008,*, volume 5086 of *LNCS*, pages 54–72. Springer, 2008. https://www.eecs.harvard.edu/~alon/PAPERS/lattices/swifft.pdf. 13, 30

[60] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, pages 382–400, 2010. 8

[61] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, 2010. http://www.iacr.org/archive/eurocrypt2010/66320288/66320288.pdf. 8, 13, 28, 29, 30

[62] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. Slides of the talk given by Chris Peikert at Eurocrypt 2010, 2010. http://crypto.rd.francetelecom.com/events/eurocrypt2010/talks/slides-ideal-lwe.pdf. 8

[63] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM*, 60(6):43:1–43:35, 2013. http://www.cims.nyu.edu/~regev/papers/ideal-lwe.pdf. 8

[64] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for Ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, 2013. http://www.iacr.org/archive/eurocrypt2013/78810035/78810035.pdf. 29

[65] Vadim Lyubashevsky and Gregor Seiler. NTTRU: Truly fast NTRU using NTT. *TCHES*, 2019. https://eprint.iacr.org/2019/040. 2, 29

[66] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *SODA '10 Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1468–1480. SIAM, 2010. https://cseweb.ucsd.edu/~daniele/papers/Sieve.pdf. 24

[67] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985. http://www.ams.org/journals/mcom/1985-44-170/S0025-5718-1985-0777282-X/S0025-5718-1985-0777282-X.pdf. 22

[68] National Institute of Standards and Technology. FIPS PUB 202 – SHA-3 standard: Permutation-based hash and extendable-output functions, 2015. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf. 12, 15

[69] Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008. ftp://ftp.di.ens.fr/pub/users/pnguyen/JoMC08.pdf. 24

[70] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-secure and masked Ring-LWE implementation. IACR Cryptology ePrint Archive report 2016/1109, 2016. https://eprint.iacr.org/2016/1109. 22

[71] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem, 2009. https://web.eecs.umich.edu/~cpeikert/pubs/svpcrypto.pdf (full version of [72]). 9, 13, 36

[72] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *STOC '09 Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 333–342. ACM, 2009. See also full version [71]. 36

[73] Chris Peikert. Lattice cryptography for the Internet. In Michele Mosca, editor, *Post-Quantum Cryptography*, volume 8772 of *LNCS*, pages 197–219. Springer, 2014. http://web.eecs.umich.edu/~cpeikert/pubs/suite.pdf. 28

[74] Chris Peikert. How (not) to instantiate Ring-LWE. In Vassilis Zikas and Roberto De Prisco, editors, *Security and Cryptography for Networks*, volume 9841 of *LNCS*, pages 411–430. Springer, 2016. https://web.eecs.umich.edu/~cpeikert/pubs/instantiate-rlwe.pdf. 30

[75] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David A. Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, 2008. https://www.iacr.org/archive/crypto2008/51570556/51570556.pdf. 29

[76] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. To BLISS-B or not to be – attacking strongSwan's implementation of post-quantum signatures. In *CCS '17 Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1843–1855. ACM, 2017. https://eprint.iacr.org/2017/490. 14

[77] Thomas Pöppelmann and Tim Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. In Tanja Lange, Kristin Lauter, and Petr Lisoněk, editors, *Selected Areas in Cryptography – SAC 2013*, volume 8282 of *LNCS*, pages 68–85. Springer, 2013. https://www.ei.rub.de/media/sh/veroeffentlichungen/2013/08/14/lwe_encrypt.pdf. 9, 13

[78] Thomas Pornin. BearSSL – a smaller SSL/TLS library, 2018. https://bearssl.org/ (accessed 2019-03-15). 22

[79] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *LNCS*, pages 513–533. Springer, 2017. https://eprint.iacr.org/2017/594. 22

[80] Prasanna Ravi, Debapriya Basu Roy, Shivam Bhasin, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Number "not used" once - practical fault attack on pqm4 implementations of nist candidates. IACR Cryptology ePrint Archive report 2018/211, 2018. https://eprint.iacr.org/2018/211. 30, 31

[81] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC '05 Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 84–93. ACM, 2005. Preliminary version of [82]. 8, 14, 29

[82] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):34, 2009. http://www.cims.nyu.edu/~regev/papers/qcrypto.pdf. 8, 37

[83] Sujoy Sinha Roy, Frederik Vercauteren, Nele Mentens, Donald Donglong Chen, and Ingrid Verbauwhede. Compact Ring-LWE cryptoprocessor. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *LNCS*, pages 371–391. Springer, 2014. http://www.iacr.org/archive/ches2014/87310183/87310183.pdf. 13

[84] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. IACR Cryptology ePrint Archive report 2017/1005, 2017. https://eprint.iacr.org/2017/1005. 20

[85] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, 1994. http://www.csie.nuk.edu.tw/~cychen/Lattices/Lattice%20Basis%20Reduction_%20Improved%20Practical%20Algorithms%20and%20Solving%20Subset%20Sum%20Problems.pdf. 23

[86] Shuai Zhou, Haiyang Xue, Daode Zhang, Kunpeng Wang, Xianhui Lu, Bao Li, and Jingnan He. Preprocess-then-ntt technique and its applications to KYBER and NEWHOPE. IACR Cryptology ePrint Archive report 2018/995, 2018. https://eprint.iacr.org/2018/995. 30