CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM

Joppe Bos
NXP
joppe.bos@nxp.com

Léo Ducas
CWI
ducas@cwi.nl

Eike Kiltz
RUB
eike.kiltz@rub.de

Tancrède Lepoint
SRI International
tancrede.lepoint@sri.com

Vadim Lyubashevsky
IBM Research - Zurich
vad@zurich.ibm.com

John M. Schanck
University of Waterloo / IQC
jschanck@uwaterloo.ca

Peter Schwabe
Radboud University
peter@cryptojedi.org

Damien Stehlé
ENS de Lyon
damien.stehle@ens-lyon.fr

ABSTRACT

Recent advances in quantum computing and the announcement by the National Institute of Standards and Technology (NIST) to define new standards for digital-signature, encryption, and key-establishment protocols increased interest in post-quantum cryptographic schemes.

This paper introduces Kyber (part of the CRYSTALS – *Cryptographic Suite for Algebraic Lattices* – package that will be submitted to the NIST call for post-quantum standards), a portfolio of post-quantum cryptographic primitives built around a key-encapsulation mechanism (KEM), based on hardness assumptions over module lattices. We first introduce a CPA-secure public key encryption scheme, apply a variant of the Fujisaki–Okamoto transform to create a CCA-secure KEM, and eventually construct, in a black-box manner, CCA-secure encryption, key exchange, and authenticated-key-exchange schemes. The security of our primitives is based on the hardness of Module-LWE in the classical and quantum random oracle models, and our concrete parameters conservatively target more than 128 bits of post-quantum security.

We implemented and benchmarked the CCA-secure KEM and key exchange protocols against the ones that are based on LWE and Ring-LWE: we conclude that our schemes are not only as efficient but also feature more flexibility and security advantages over the latter schemes.

KEYWORDS

Lattice cryptography, Key encapsulation mechanism, Implementation, Module lattices, (Authenticated) key exchange, CCA security.

1 INTRODUCTION

There has been an increased interest in post-quantum cryptographic schemes triggered by recent advances in quantum computing [34] and the announcement by the National Institute of Standards and Technology (NIST) to define new standards for digital-signature, encryption, and key-establishment protocols [26]. Constructions based on the hardness of lattice problems are considered to be one of the leading candidates to replace the currently used schemes based on the believed hardness of the traditional number theoretic problems such as integer factorization and discrete logarithms.

Lattice cryptography initially gained a lot of interest in the theoretical community due to the fact that the designs for cryptographic constructions were accompanied by security proofs based on *worst-case* instances of lattice problems. The first lattice-based encryption scheme was proposed by Ajtai and Dwork [1]. This scheme was later simplified and improved upon by Regev in [65, 66]. One of the major achievements of Regev's work was the introduction of an intermediate problem – the Learning With Errors (LWE) Problem – which was relatively simple to use in cryptographic constructions and asymptotically at least as hard as some standard worst-case lattice problems [23, 59].

The LWE assumption states that it is hard to distinguish from uniform the distribution $(\mathbf{A}, \mathbf{As}+\mathbf{e})$, where $\mathbf{A}$ is a uniformly-random matrix in $\mathbb{Z}_q^{m\times n}$, $\mathbf{s}$ is a uniformly-random vector in $\mathbb{Z}_q^n$, and $\mathbf{e}$ is a vector with random "small" coefficients chosen from some distribution. Applebaum et al. [5] showed that the secret $\mathbf{s}$ in the LWE problem does not need to be chosen uniformly at random: the problem remains hard if $\mathbf{s}$ is chosen from the same narrow distribution as the errors $\mathbf{e}$. Based on the idea from the NTRU cryptosystem [43] of working with elements over polynomial rings rather than over the integers, and following a series of works on this topic [54, 56, 61, 68], Lyubashevsky et al. [55] showed that it is also hard to distinguish a variant of the LWE distribution from the uniform one over certain polynomial rings, thus defining the Ring-LWE assumption.

The combination of all of the above results finally led to the cryptosystem in Section 3.[1] Setting the parameter k to 1 and defining $R_q = \mathbb{Z}_q[X]/(X^n+1)$ makes the scheme a Ring-LWE cryptosystem as originally defined in [55], whereas setting the ring R_q to $\mathbb{Z}_q$, makes the scheme an LWE-based one.[2] If one sets the ring R_q to some polynomial ring of dimension greater than 1 and sets $k > 1$, then the scheme is based on the hardness of the Module-LWE problem [22, 51]. The number of bits that can be transmitted is related to the dimension of the ring, thus using a ring R_q of larger degree n allows one to transmit more bits, and this is the main reason that Ring-LWE encryption is more efficient than LWE encryption. On the other hand, having a smaller k implies more algebraic structure, making the scheme potentially susceptible to more avenues

[1] It should be noted that this cryptoscheme design, as well as the result from [5] applied to the Learning Parity with Noise (LPN) problem, was already present much earlier in the work of Alekhnovich [3] in which he constructed a cryptosystem based on the hardness of the LPN problem. The LWE problem is a generalization of LPN and results in more efficient cryptosystems.

[2] The original cryptosystems did not include the "bit-dropping" $\mathsf{Compress}_q$ functions in key generation and encryption, but this idea was considered folklore (see [63, Sec. 2.3] for some references).

of attack. Nevertheless, at this point in time, it is unknown how to exploit the algebraic structure of Ring-LWE and concrete parameters are chosen according to the corresponding LWE problem of dimension $k \cdot n$.

This cryptosystem design was also applied to build a CPA-secure KEM by Ding et al. [35] and Peikert [60]. The main difference between this KEM and the encryption scheme is in how the parameter v is defined in line 6 of the encryption algorithm (Algorithm 2). The advantage of the constructions in [35, 60] is that if one would like to construct a CPA-secure KEM transmitting a b-bit key, then the ciphertext is b bits shorter, which is about a 3% savings for typical parameters.[3] If one wishes to construct a CCA-secure KEM, however, this advantage disappears since typical 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. Since in this paper we are only concerned with CCA-secure constructions, we find it simpler to start directly from the CPA-secure encryption scheme design in Section 3.

The above designs based on Ring-LWE have resulted in many recent concrete proposals accompanied by practical implementations. The instantiation presented [19] is based on Ring-LWE and was subsequently improved in [4, 52], which resulted in an experiment by Google where they used this key-exchange protocol in their Chrome Canary browser from July to November 2016 [21, 49]. Although the Ring-LWE problem results in very practical key-sizes and protocol communication, the additional algebraic structure might inspire less confidence in the underlying security. This was the motivation to study a very similar practical instantiation of a key-exchange protocol but based on LWE in [18], or to propose an efficient implementation of a CCA-secure KEM over a different ring [12].

1.1 Our contribution

Our main contribution is a highly-optimized instantiation of a CCA-secure KEM called Kyber, which is based on the hardness of Module-LWE. More precisely, we instantiate a CPA-secure PKE scheme Kyber.CPA in Section 3, then apply a variant of the Fujisaki–Okamoto transform to create a CCA-Secure KEM Kyber in Section 4. The security reduction from the hardness of Module-LWE is tight in the random-oracle model, but non-tight is the quantum-random-oracle model [44]. From a CCA-secure KEM, one can construct, in a black-box manner, CCA-secure encryption (Kyber.Hybrid), key exchange (Kyber.KE), and authenticated-key-exchange (Kyber.AKE) schemes. Our resulting schemes are as efficient as ones that are based on Ring-LWE, but have additional flexibility and security advantages.

Flexibility. One of the most expensive operations in lattice-based schemes over rings is polynomial multiplication. If a scheme is based on the Ring-LWE assumption (i.e., with $k = 1$ in Algorithm 2), then if one wants to vary the security parameter related to the scheme, one would need to change the ring R_q and re-implement all the ring operations. With our design, where we only work over the ring $R_q = \mathbb{Z}_{7681}[X]/(X^{256} + 1)$, there is only one ring over which operations need to be optimized. Increasing and decreasing the security of the scheme can then be done simply by changing the dimension k of the matrix. Our proposed conservative parameters, which we believe have very generous margins for 128-bit post-quantum security, use $k = 3$. This is the scheme we recommend using for long-term security. But if one only needs short-term security, we believe that today (and probably for the near future) one can safely use $k = 2$ for which we conservatively estimate 102-bit post quantum security. This latter parameter set will reduce the communication size of the key exchange by around 33% and considerably speed up the scheme. The main building blocks of the two schemes are exactly the same, and any optimized software / hardware used for efficient multiplication in R_q can be re-used.

Security. There have been recent attacks exploiting the algebraic structure of cyclotomic ideal lattices [15, 24, 31, 32], and others that exploit the presence of dense sub-lattices in NTRU lattices [2, 47]. In these attacks, it appears that the dimension of the module makes a big difference. In particular, the quantum attacks on finding short vectors in ideals currently do not extend to Ring-LWE [15, 24, 31, 32]. The obstacle seems to be that solutions to the shortest vector problem in ideal lattices are ring elements, whereas solutions to the Ring-LWE problem are elements in a module of dimension 2. In that respect, solutions to Module-LWE are in a module of dimension $k + 1$. Similarly, the larger module dimension also decreases the relative dimension of the dense sub-lattice, making the attack of [47] inapplicable. Based on the recent cryptanalytic progress, it therefore seems that practical attacks are less likely to appear against Module-LWE than against Ring-LWE or NTRU.

High performance. As we previously mentioned, the main reason that Ring-LWE is preferred to LWE in practical applications is because it allows for a larger message to be transmitted in the same amount of communication. We show that the flexibility and security improvements by moving from Ring-LWE to Module-LWE come at almost no cost. In particular, since public-key protocols only need to transmit 256 bits of information, it is unnecessary to work with rings that are greater than dimension 256 in order to be able to transmit one bit per coefficient of a ring element. Thus the key and message sizes of our protocols versus those based on Ring-LWE are not affected.

The one part where using a $k > 1$ is less efficient than $k = 1$ is when dealing with the $k \times k$ random matrix $\mathbf{A}$. If one uses $k = 1$ and a ring of dimension n, then the representation of $\mathbf{A}$ is $k^2 n = n$ elements in $\mathbb{Z}_q$. On the other hand, if one uses $k = 3$ and a ring of dimension $n/3$ (thus keeping the lattice-reduction security the same), then $\mathbf{A}$ requires $k^2 n = 3n$ elements in $\mathbb{Z}_q$ to represent. Since the matrix $\mathbf{A}$ is never stored, but rather expanded from some seed ρ using an XOF, this disadvantage only manifests in the slight increase in the running time used in the expansion. This is to some extent mitigated because the k^2 entries of the matrix $\mathbf{A}$ can be expanded independently, which enables very efficient vectorization of the XOF computation.

Take away. In this paper, we propose and implement a portfolio of post-quantum cryptographic primitives (CPA-secure encryption, CCA-secure KEM, CCA-secure public-key encryption, key

[3] It was mentioned in [60, Sec. 4] that the ciphertext in the KEM goes down by a factor of two compared to encryption schemes. However, this applies only to the naive instantiations of encryption schemes where the "bit-dropping" $\mathsf{Compress}_q$ function is not applied to v in line 6 of Algorithm 2.

exchange and authenticated key exchange) based on the hardness of Module-LWE in the classical and quantum random-oracle models. Our schemes are as efficient as the ones based on Ring-LWE, but also feature flexibility and security advantages.

Availability of software. We place all software described in this paper into the public domain to maximize reusability of our results. It is available for download on GitHub: https://github.com/pq-crystals/kyber.

2 PRELIMINARIES

All our algorithms are probabilistic. If b is a string, then $a \leftarrow \mathsf{A}(b)$ denotes the output of algorithm A when run on input b; if A is deterministic, then a is a fixed value and we write $a := \mathsf{A}(b)$. We use the notation $b := \mathsf{A}(b; r)$ to make the randomness r of a probabilistic algorithm A explicit.

2.1 Cryptographic definitions

A public-key encryption scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is a triple of probabilistic algorithms together with a message space $\mathcal{M}$. The key-generation algorithm KeyGen returns a pair (pk, sk) consisting of a public key and a secret key. The encryption algorithm Enc takes a public key pk and a message $m \in \mathcal{M}$ to produce a ciphertext c. Finally, the deterministic decryption algorithm Dec takes a secret key sk and a ciphertext c, and outputs either a message $m \in \mathcal{M}$ or a special symbol $\perp$ to indicate rejection. We say that PKE is $(1-\delta)$-correct if for all messages $m \in \mathcal{M}$, we have $\Pr[\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m] \geq 1 - \delta$, where the probability is taken over $(pk, sk) \leftarrow \mathsf{KeyGen}()$ and the random coins of Enc.

We recall the standard security notions for public-key encryption of indistinguishability under chosen-ciphertext and chosen-plaintext attacks (IND-CCA and IND-CPA) [64]. The advantage of an adversary A is defined as $\mathbf{Adv}^{\mathrm{cca}}_{\mathsf{PKE}}(\mathsf{A}) =$

$$\left| \Pr\left[b = b' \ : \begin{array}{l} (pk, sk) \leftarrow \mathsf{KeyGen}(); \\ (m_0, m_1, s) \leftarrow \mathsf{A}^{\textsc{Dec}(\cdot)}(pk); \\ b \leftarrow \{0,1\}; c^* \leftarrow \mathsf{Enc}(pk, m_b); \\ b' \leftarrow \mathsf{A}^{\textsc{Dec}(\cdot)}(s, c^*) \end{array} \right] - \frac{1}{2} \right|,$$

where the decryption oracle is defined as $\textsc{Dec}(\cdot) := \mathsf{Dec}(sk, \cdot)$. We further require that $|m_0| = |m_1|$ and that in the second phase A is not allowed to query $\textsc{Dec}(\cdot)$ with the challenge ciphertext c^*. The advantage $\mathbf{Adv}^{\mathrm{cpa}}_{\mathsf{PKE}}(\mathsf{A})$ of an adversary A is defined as $\mathbf{Adv}^{\mathrm{cca}}_{\mathsf{PKE}}(\mathsf{A})$, with the modification that A cannot query the decryption oracle.

A key-encapsulation scheme $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ is a triple of probabilistic algorithms together with a key space $\mathcal{K}$. The key-generation algorithm KeyGen returns a pair (pk, sk) consisting of a public key and a secret key. The encapsulation algorithm Encaps takes a public key pk to produce a ciphertext c and a key $K \in \mathcal{K}$. Finally, the deterministic decapsulation algorithm Decaps takes a secret key sk and a ciphertext c, and outputs either a key $K \in \mathcal{K}$ or a special symbol $\perp$ to indicate rejection. We say that KEM is $(1-\delta)$-correct if $\Pr[\mathsf{Decaps}(sk, c) = K : (c, K) \leftarrow \mathsf{Encaps}(pk)] \geq 1 - \delta$, where the probability is taken over $(pk, sk) \leftarrow \mathsf{KeyGen}()$ and the random coins of Encaps.

We recall the standard security notion for key encapsulation of indistinguishability under chosen-ciphertext attack. The advantage of an adversary A is defined as $\mathbf{Adv}^{\mathrm{cca}}_{\mathsf{KEM}}(\mathsf{A}) =$

$$\left| \Pr\left[b = b' \ : \begin{array}{l} (pk, sk) \leftarrow \mathsf{KeyGen}(); \\ b \leftarrow \{0,1\}; \\ (c^*, K_0^*) \leftarrow \mathsf{Encaps}(pk); K_1^* \leftarrow \mathcal{K}; \\ b' \leftarrow \mathsf{A}^{\textsc{Decaps}(\cdot)}(pk, c^*, K_b^*) \end{array} \right] - \frac{1}{2} \right|,$$

where the Decaps oracle is defined as $\textsc{Decaps}(\cdot) := \mathsf{Decaps}(sk, \cdot)$. We further require that A is not allowed to query $\textsc{Decaps}(\cdot)$ with the challenge ciphertext c^*.

In the random oracle model [9], the adversary A is additionally given access to a random oracle that it can query up to q_H times. If the adversary has access to a quantum computer, it is realistic to model its access to all "offline primitives" (such as hash functions) in a quantum setting. Concretely, in the quantum random oracle model [17] the adversary has access to a quantum random oracle (also called quantum accessible random oracle) that can be queried up to q_H times on arbitrary quantum superpositions of input strings.

2.2 Rings and distributions

Let R and R_q denote the rings $\mathbb{Z}[X]/(X^n + 1)$ and $\mathbb{Z}_q[X]/(X^n + 1)$, respectively, where $n = 2^{n'-1}$ such that $X^n + 1$ is the $2^{n'}$-th cyclotomic polynomial. Throughout this paper, the values of n, n' and q are 256, 9 and 7681, respectively. 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.

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

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

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.

Extendable output function. Suppose that Sam is an extendable output function, that is a function on bit strings in which the output can be extended to any desired length. If we would like Sam to take as input x and then produce a value y that is distributed according to distribution S (or uniformly over a set S), we write

$y \sim S := \mathsf{Sam}(x)$. It is important to note that this procedure is completely deterministic: a given x will always produce the same y. For simplicity we assume that the output distribution of Sam is perfect, whereas in practice Sam will be implemented using random oracles and produces an output that is statistically close to the perfect distribution.

Binomial distribution. We define the centered binomial distribution B_η for some positive integer η as follows:

$$\text{Sample } (a_1,\ldots,a_\eta,b_1,\ldots,b_\eta) \leftarrow \{0,1\}^{2\eta} \text{ and output } \sum_{i=1}^{\eta}(a_i-b_i).$$

If v is an element of R, we write $v \leftarrow \beta_\eta$ to mean that $v \in R$ is generated from a distribution where each of its coefficients is generated according to B_η. Similarly, a k-dimensional vector of polynomials $\mathbf{v} \in R^k$ can be generated according to the distribution β_η^k.

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

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

The functions satisfying these requirements are defined as:

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

If x' is a function of x as in Eq. (1), then for a randomly chosen $x \leftarrow \mathbb{Z}_q$, the distribution of

$$x' - x \bmod^{\pm} q$$

is almost uniform over the integers of magnitude at most B_q. In particular, this distribution has equal weight over integers of magnitude at most $B_q - 1$ and has a smaller weight on the integer(s) of magnitude B_q.

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 public key and the ciphertext which do not have much effect on the correctness probability of decryption – thus making the parameters smaller. The $\mathsf{Compress}_q$ function is also used in one other place where its intuitive purpose is not to "compress". In line 3 of the decryption procedure (Algorithm 3), the 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.

2.3 Module-LWE

Let k be a positive integer parameter. The hard problem underlying the security of our schemes is Module-LWE. 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 $\mathbf{s} \leftarrow \beta_\eta^k$ common to all samples and $e_i \leftarrow \beta_\eta$ fresh for every sample.[4] More precisely, for an algorithm A, we define $\mathbf{Adv}^{\mathrm{mlwe}}_{m,k,\eta}(\mathsf{A}) =$

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

3 KYBER'S IND-CPA-SECURE ENCRYPTION

Let k, d_t, d_u, d_v be positive integer parameters, and recall that $n = 256$. Let $\mathcal{M} = \{0,1\}^{256}$ denote the message space, where every message $m \in \mathcal{M}$ can be viewed as a polynomial in R with coefficients in $\{0,1\}$. Consider the public-key encryption scheme $\mathsf{Kyber.CPA} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as described in Algorithms 1 to 3.

Algorithm 1 Kyber.CPA.KeyGen(): key generation

1: $\rho,\sigma \leftarrow \{0,1\}^{256}$
2: $\mathbf{A} \sim R_q^{k\times k} := \mathsf{Sam}(\rho)$
3: $(\mathbf{s},\mathbf{e}) \sim \beta_\eta^k \times \beta_\eta^k := \mathsf{Sam}(\sigma)$
4: $\mathbf{t} := \mathsf{Compress}_q(\mathbf{A}\mathbf{s}+\mathbf{e}, d_t)$
5: **return** $(pk := (\mathbf{t},\rho), sk := \mathbf{s})$

Algorithm 2 Kyber.CPA.Enc($pk = (\mathbf{t},\rho), m \in \mathcal{M}$): encryption

1: $r \leftarrow \{0,1\}^{256}$
2: $\mathbf{t} := \mathsf{Decompress}_q(\mathbf{t}, d_t)$
3: $\mathbf{A} \sim R_q^{k\times k} := \mathsf{Sam}(\rho)$
4: $(\mathbf{r},\mathbf{e}_1,e_2) \sim \beta_\eta^k \times \beta_\eta^k \times \beta_\eta := \mathsf{Sam}(r)$
5: $\mathbf{u} := \mathsf{Compress}_q(\mathbf{A}^T\mathbf{r}+\mathbf{e}_1, d_u)$
6: $v := \mathsf{Compress}_q\left(\mathbf{t}^T\mathbf{r} + e_2 + \left\lceil \frac{q}{2} \right\rfloor \cdot m, d_v\right)$
7: **return** $c := (\mathbf{u}, v)$

Algorithm 3 Kyber.CPA.Dec($sk = \mathbf{s}, c = (\mathbf{u}, v)$): decryption

1: $\mathbf{u} := \mathsf{Decompress}_q(\mathbf{u}, d_u)$
2: $v := \mathsf{Decompress}_q(v, d_v)$
3: **return** $\mathsf{Compress}_q(v - \mathbf{s}^T\mathbf{u}, 1)$

Correctness. We show below the correctness of the encryption scheme described in Algorithms 1 to 3. We will select parameters in Section 6 to make the decryption error negligible, i.e., so that Kyber.CPA is $(1-\delta)$-correct with $\delta < 2^{-128}$.

Theorem 3.1. *Let k be a positive integer parameter. Let $\mathbf{s},\mathbf{e},\mathbf{r},\mathbf{e}_1,e_2$ be random variables that have the same distribution as in Algorithms 1 and 2. Also, let $\mathbf{c}_t \leftarrow \psi_{d_t}^k, \mathbf{c}_u \leftarrow \psi_{d_u}^k, c_v \leftarrow \psi_{d_v}$ be distributed according to the distribution ψ defined as follows:*
Let ψ_d^k be the following distribution over R:

[4] While the exact distribution shape does not seem to play any role in the hardness of (Module)-LWE encryption schemes, we mention that it is possible to show with a simple Rényi divergence-based analysis a la [4, 7] that one can substitute β_η with the n-dimensional rounded Gaussian distribution of standard deviation $\sqrt{\eta/2}$, which was the one considered in [51].

1: *Choose uniformly-random* $\mathbf{y} \leftarrow R^k$
2: ***return*** $\left(\mathbf{y} - \mathsf{Decompress}_q\left(\mathsf{Compress}_q(\mathbf{y}, d), d\right)\right)$ $mod^{\pm}$ q.

Denote

$$\delta = \Pr\left[\left\|\mathbf{e}^T\mathbf{r} + e_2 + \mathbf{c}_v - \mathbf{s}^T\mathbf{e}_1 + \mathbf{c}_t^T\mathbf{r} + \mathbf{s}^T\mathbf{c}_u\right\|_\infty \geq \lceil q/4 \rfloor\right].$$

Then Kyber.CPA *is* $(1-\delta)$*-correct.*

Remark 3.2. We provide with our software a Python script that allows to compute a tight upper bound on δ; the parameter set we recommend for Kyber in Table 1 yields $\delta = 2^{-142}$.

Proof. The value of $\mathbf{t}$ in line 6 of Algorithm 2 is:

$$\mathbf{t} = \mathsf{Decompress}_q\left(\mathsf{Compress}_q(\mathbf{As} + \mathbf{e}, d_t), d_t\right) = \mathbf{As} + \mathbf{e} + \mathbf{c}_t,$$

for some $\mathbf{c}_t \in R^k$. The value of $\mathbf{u}$ in Algorithm 3 is

$$\mathbf{u} = \mathsf{Decompress}_q\left(\mathsf{Compress}_q(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1, d_u), d_u\right) = \mathbf{A}^T\mathbf{r} + \mathbf{e}_1 + \mathbf{c}_u,$$

for some $\mathbf{c}_u \in R^k$. And the value of v is

$$\begin{aligned} v &= \mathsf{Decompress}_q\left(\mathsf{Compress}_q(\mathbf{t}^T\mathbf{r} + e_2 + \lceil q/2 \rfloor \cdot m, d_v), d_v\right) \\ &= \mathbf{t}^T\mathbf{r} + e_2 + \lceil q/2 \rfloor \cdot m + \mathbf{c}_v \\ &= (\mathbf{As} + \mathbf{e} + \mathbf{c}_t)^T\mathbf{r} + e_2 + \lceil q/2 \rfloor \cdot m + c_v \\ &= (\mathbf{As} + \mathbf{e})^T\mathbf{r} + e_2 + \lceil q/2 \rfloor \cdot m + c_v + \mathbf{c}_t^T\mathbf{r}, \end{aligned}$$

for some $c_v \in R$. In all of the above, we can safely assume that the values $\mathbf{c}_t, \mathbf{c}_u$, and c_v are distributed according to the distribution ψ defined in the Theorem statement. The reason is that all of these are of the form $\left(\mathbf{y} - \mathsf{Decompress}_q\left(\mathsf{Compress}_q(\mathbf{y}, d), d\right)\right) \bmod^{\pm} q$ where $\mathbf{y}$ is pseudo-random based on the hardness of Module-LWE.

Using the above, we obtain

$$v - \mathbf{s}^T\mathbf{u} = \mathbf{e}^T\mathbf{r} + e_2 + \lceil q/2 \rfloor \cdot m + \mathbf{c}_v + \mathbf{c}_t^T\mathbf{e}_1 - \mathbf{s}^T\mathbf{r} + \mathbf{s}^T\mathbf{c}_u$$

If $\left\|\mathbf{e}^T\mathbf{r} + e_2 + \mathbf{c}_v - \mathbf{s}^T\mathbf{e}_1 + \mathbf{c}_t^T\mathbf{r} + \mathbf{s}^T\mathbf{c}_u\right\|_\infty < \lceil q/4 \rfloor$, then we can write $v - \mathbf{s}^T\mathbf{u} = w + \lceil q/2 \rfloor \cdot m$ where $\|w\|_\infty < \lceil q/4 \rfloor$. Define $m' = \mathsf{Compress}_q(v - \mathbf{s}^T\mathbf{u}, 1)$. We then know that

$$\lceil q/4 \rfloor \geq \|v - \mathbf{s}^T\mathbf{u} - \lceil q/2 \rfloor \cdot m'\|_\infty = \|w + \lceil q/2 \rfloor \cdot m - \lceil q/2 \rfloor \cdot m'\|_\infty.$$

By the triangle inequality and the fact that $\|w\|_\infty < \lceil q/4 \rfloor$, we obtain

$$\|\lceil q/2 \rfloor \cdot (m - m')\|_\infty < 2 \cdot \lceil q/4 \rfloor,$$

which (for all odd q) implies that $m = m'$, and proves the correctness of Kyber.CPA. □

Security. We prove that the encryption scheme defined above is IND-CPA secure under the Module-LWE hardness assumption.

Theorem 3.3. *For any adversary* A*, there exists an adversary* B *such that* $\mathbf{Adv}^{\mathrm{cpa}}_{\mathsf{Kyber.CPA}}(\mathsf{A}) \leq 2 \cdot \mathbf{Adv}^{\mathrm{mlwe}}_{k+1,k,\eta}(\mathsf{B})$.

Proof. Let A be an adversary that is executed in the IND-CPA security experiment which we call game G_0, i.e., $\mathbf{Adv}^{\mathrm{cpa}}_{\mathsf{PKE}}(\mathsf{A}) = |\Pr[b = b' \text{ in game } G_0] - 1/2|$. In game G_1, the value $\mathbf{t}' := \mathbf{As} + \mathbf{e}$ which is used in KeyGen is substituted by a uniform random value. It is possible to verify that there exists an adversary B with the same running time as that of A such that $|\Pr[b = b' \text{ in game } G_0] - |\Pr[b = b' \text{ in game } G_1]| \leq \mathbf{Adv}^{\mathrm{mlwe}}_{k,k,\eta}(\mathsf{B}) \leq \mathbf{Adv}^{\mathrm{mlwe}}_{k+1,k,\eta}(\mathsf{B})$. In game G_2, the values $\mathbf{u}' := \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$ and $v' := \mathbf{t}^T\mathbf{r} + e_2$ used in the generation of the challenge ciphertext are simultaneously substituted with uniform random values. Again, there exists an adversary B with the same running time as that of A with $|\Pr[b = b' \text{ in game } G_1] - |\Pr[b = b' \text{ in game } G_2]| \leq \mathbf{Adv}^{\mathrm{mlwe}}_{k+1,k,\eta}(\mathsf{B})$. Note that in game G_2, the value v from the challenge ciphertext is independent of bit b and therefore $\Pr[b = b' \text{ in game } G_2] = 1/2$. Collecting the probabilities yields the required bound. □

4 THE CCA-SECURE KEM

Let $\mathrm{G}\colon \{0,1\}^* \to \{0,1\}^{3\times 256}$ and $\mathrm{H}\colon \{0,1\}^* \to \{0,1\}^{256}$ be hash functions. Consider the public-key key encapsulation mechanism Kyber = (KeyGen, Encaps, Decaps) as described in Algorithms 1, 4 and 5, where KeyGen is the same as the one of Kyber.CPA from the previous section, with the difference that sk also contains $pk = (\rho, \mathbf{t})$ and a secret 256-bit random value z. It is obtained by applying a KEM variant [44] of the Fujisaki–Okamoto transform [37, 69] to the Kyber.CPA encryption scheme. Note that we make explicit the randomness r in the Enc algorithm.

Algorithm 4 Kyber.Encaps($pk = (\rho, \mathbf{t})$)

1: $m \leftarrow \{0,1\}^{256}$
2: $(\hat{K}, r, d) := \mathrm{G}(pk, m)$
3: $(\mathbf{u}, v) := \mathsf{Kyber.CPA.Enc}\left((\rho, \mathbf{t}), m; r\right)$
4: $c := (\mathbf{u}, v, d)$
5: $K := \mathrm{H}(\hat{K}, c)$
6: **return** (c, K)

Algorithm 5 Kyber.Decaps($sk = (\mathbf{s}, z, \rho, \mathbf{t}), c = (\mathbf{u}, v, d)$)

1: $m' := \mathsf{Kyber.CPA.Dec}(\mathbf{s}, (\mathbf{u}, v))$
2: $(\hat{K}', r', d') := \mathrm{G}(pk, m')$
3: $(\mathbf{u}', v') := \mathsf{Kyber.CPA.Enc}\left((\rho, \mathbf{t}), m'; r'\right)$
4: **if** $(\mathbf{u}', v', d') = (\mathbf{u}, v, d)$ **then**
5: **return** $K := \mathrm{H}(\hat{K}', c)$
6: **else**
7: **return** $K := \mathrm{H}(z, c)$
8: **end if**

We stress that Kyber.Decaps never returns $\perp$. Instead, in case re-encryption fails, it returns a pseudo-random key $K := \mathrm{H}(z, c)$, where z is a random, secret seed.

Correctness. If Kyber.CPA is $(1-\delta)$-correct and G is a random oracle, then Kyber is $(1-\delta)$-correct [44].

Security. The following concrete security statement proves Kyber's CCA-security when the hash functions G and H are modeled as random oracles. We provide the concrete security bounds from [44] which considers the KEM variant of the FO transformation and also takes a non-zero correctness error δ into account.

Theorem 4.1. *For any adversary* A *that makes at most* q_H *many queries to random oracle H, at most* q_G *many queries to random oracle G, and* q_D *queries to the decryption oracle, there exists an adversary* B *such that*

$$\mathbf{Adv}^{\mathrm{cca}}_{\mathsf{Kyber}}(\mathsf{A}) \leq 3\mathbf{Adv}^{\mathrm{cpa}}_{\mathsf{Kyber.CPA}}(\mathsf{B}) + q_H \cdot \delta + \frac{2q_G + q_H + 1}{2^{256}}.$$

Note that the security bound is tight. In particular, in combination with Theorems 3.1 and 3.3 we obtain a tight reduction from the Module-LWE hardness assumption. We remark that there exists an alternative security reduction from the weaker notion of ONE-WAY CPA-security [44] of Kyber.CPA which is, however, not tight as it loses a multiplicative factor q_G.

The value d in Kyber's ciphertexts is not necessary for the security proof in case H and G are modeled as standard random oracles. However, it is crucial for a proof in the quantum random oracle model. Concretely [44, 69] proved that Kyber is CCA secure in the quantum random oracle model, provided that Kyber.CPA is CPA-secure. Again, we provide the concrete bound from [44].

Theorem 4.2. *For any quantum adversary* A *that makes at most q_H many queries to quantum random oracle H, at most q_G many queries to quantum random oracle G, and at most q_D many (classical) queries to the decryption oracle, there exists a quantum adversary* B *such that*

$$\mathbf{Adv}^{\mathrm{cca}}_{\mathrm{Kyber}}(\mathsf{A}) \leq 4q_H\sqrt{q_D \cdot q_H \cdot \delta + q_G \cdot \sqrt{\mathbf{Adv}^{\mathrm{cpa}}_{\mathrm{Kyber.CPA}}(\mathsf{B})}}.$$

Unfortunately, as it is common for proofs in the quantum random oracle model, the above security bound is far from tight and therefore can only serve as an asymptotic indication of Kyber's CCA-security in the quantum random oracle model.

Hashing pk into $\hat{K}$. The Kyber CCA transform is essentially the transform from [44, 69], with one small tweak: we hash the public key pk into $\hat{K}$. This tweak has two effects. First, it makes the KEM contributory; the shared key K does not depend only on input of one of the two parties. The second effect is a multi-target protection. Consider an attacker who searches through many values m to find one that is "likely" to produce a failure during decryption. Such a decryption failure of a legitimate ciphertext would leak some information about the secret key. In the pre-quantum setting this attack approach is doomed because of the negligible failure probability δ. In a post-quantum setting, the attacker could use Grover's algorithm to search for such an m. However, the attacker is then facing the problem to encode "likely to produce a decryption failure" in the Grover oracle. This is equivalent to identifying noise vectors that are likely to have a large inner product with $(\mathbf{s}, \mathbf{e})$; probably the best strategy is to search for m that produce noise vectors of large norm. Even though we believe this attack approach is unlikely to result in any better performance than a brute-force Grover search of the 256-bit shared key K, hashing pk into $\hat{K}$ ensures that an attacker would not be able to use precomputed values m against multiple targets.

CCA-secure public-key encryption. We remark that a CCA-secure public-key encryption scheme can be obtained by combining the CCA-secure KEM Kyber with any CCA-secure symmetric encryption scheme [33] (aka. DEM). We describe the resulting hybrid encryption scheme Kyber.Hybrid in Appendix A.

5 KEY EXCHANGE PROTOCOLS

Let Kyber = (KeyGen, Encaps, Decaps) be the IND-CCA secure KEM from the previous section. Figure 1 describes the Kyber key exchange protocol Kyber.KE obtained as a direct application of the key encapsulation mechanism. In key exchange constructions using a KEM, it is common to hash the "view" of each participant (i.e., all received and sent messages) into the final key. In Kyber, the public key pk is hashed into the "pre-key" $\hat{K}$ and the ciphertext is hashed into the final key K; hence the shared key obtained in a key exchange already includes the complete "view" of each participant.

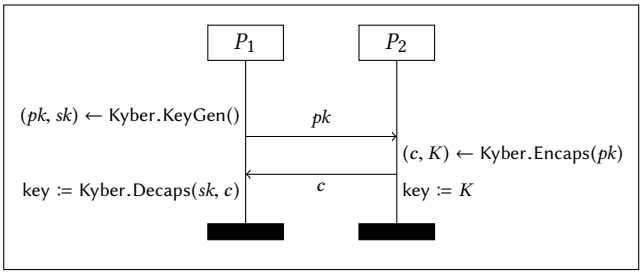

Figure 1: Kyber.KE – **Key Exchange protocol using the** Kyber = (KeyGen, Encaps, Decaps) **key encapsulation mechanism.**

Authenticated key exchanges protocols. Note that the protocol of Fig. 1 *by itself* only provides security against passive adversaries (and in particular fails to protect against man-in-the-middle attacks). Let $\mathrm{H}\colon \{0,1\}^* \to \{0,1\}^{256}$ be a hash function. Figure 2 describes our one-sided (unilateral) authenticated key exchange protocol Kyber.UAKE in which party P_1 knows the static (long-term) key of party P_2, and Fig. 3 describes our authenticated key-exchange protocol Kyber.AKE where each party knows the static (long-term) key of the other party.

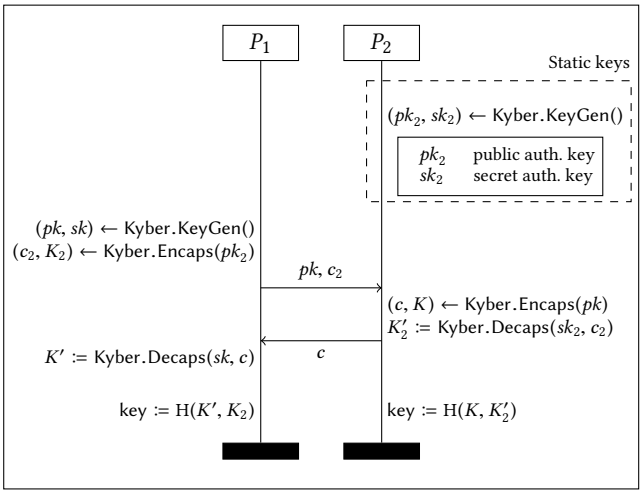

Figure 2: Kyber.UAKE – **One-sided authenticated key exchange protocol using** Kyber**, where P_1 knows the static public key of P_2.**

The shared key derived at the end of the above protocols not only depends on the ephemeral key and ciphertext (pk, c), but also on the static (long-term) keys pk_i and associated ephemeral ciphertexts c_i (where $i = 2$ and $i = 1, 2$ respectively).

Our authenticated key-exchange protocols follow a generic construction from any CCA-secure encryption scheme. Concretely,

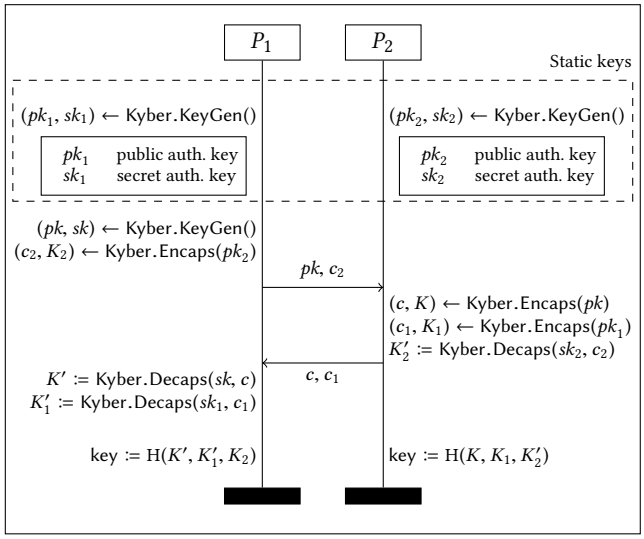

Figure 3: Kyber.AKE – **Authenticated key exchange protocol using** Kyber**, where both parties knows their respective static public keys.**

Table 1: Kyber parameter set, aiming at 128-bit classical and post-quantum security, with generous margins.

	n	k	q	η	(d_u, d_v, d_t)	δ	pq-sec
Kyber	256	3	7681	4	(11,3,11)	2^{-142}	161

security of Kyber.AKE in the Canetti–Krawczyk model with weak forward secrecy [25] follows directly from the generic security bounds of [20, 36]. (Note that full forward secrecy is not achievable for a two-round authenticated key-exchange protocol [25].)

6 PARAMETERS AND SECURITY ANALYSIS

In this section we give the Kyber parameter set that aims at 128 bits of post-quantum (and classical) security, with a generous security margin to account for future improvements in cryptanalysis. We only consider the parameters that are relevant to the underlying lattice problem; instantiations of symmetric primitives are given in Section 7.

The parameters of Kyber are summarized in Table 1. The first parameter we fixed was $n = 256$, which stems from the fact that we want to encapsulate 256 bits of entropy (targeting a 128-bit security level for symmetric keys [39]) and that we want to encode each of these bits into one polynomial coefficient. We then picked $q = 7681$ as the smallest prime that fulfills $q \equiv 1 \bmod 2n$, which allows us to use fast multiplication in R_q based on the negacyclic number-theoretic transform (NTT). The next parameter we fixed is $k = 3$, which controls the dimension of the lattice, and thereby largely the security. Finally we tuned the parameters η, d_u, d_v, and d_t to balance security, failure probability δ, public-key size, and ciphertext size. We decided to fix $d_u = d_t = 11$, which unifies compression of public keys and the "key component" $\mathbf{u}$ of the ciphertext.

Core-SVP hardness. To analyze the security of Kyber, we follow the methodology introduced in [4, Sec. 6.1]. This means that we assume that the best way to solve the Module-LWE problem underlying Kyber is to treat it as a general LWE problem. Moreover we consider the primal and dual attacks to be the only known attacks relevant to our parameter sets. After optimizing the parameters for the primal attack with respect to the success criteria of [4, Sec. 6.3], we find that the attack would invoke BKZ with blocksize 610 to 615 (depending on whether one uses the primal or dual attack). The cost of BKZ with blocksize 610 is dominated by a polynomial number of calls to a dimension 610 SVP solver. Suppressing this polynomial number of SVP calls and all subexponential factors in the cost of the best known quantum algorithm for SVP [48, Sec. 14.2.10], this implies a cost of $> 2^{161}$ operations in the quantum RAM model. According to this very conservative analysis, Kyber offers 161 bits of security against the best known quantum attacks targeting the underlying lattice problem.

LWE security vs. LWR security. The analysis in the previous paragraph considers only the noise introduced by addition of "noise polynomials" sampled from β_η; it does not take into account the additional uniform noise introduced by rounding. Various recent proposals for lattice-based cryptosystems rely *only* on this noise from rounding. See for example the schemes proposed in [12] and [28]. In some sense, the rounding induces a deterministic noise, and it is not clear how this determinism affects concrete security. On one hand, there are reductions between LWR and LWE, but they involve substantial losses, especially in the ring setting [8]. In particular, they do not apply when the number of dropped bits is so small. On the other hand, there is, to our knowledge, no attack that performs better on LWR than LWE with the corresponding uniform noise. (See [16, Sec. 4] for a reduction from LWR to LWE with corresponding uniform noise.) Yet it is not clear to us that LWR has received any dedicated attention from cryptanalysts, and we therefore prefer to remain conservative and estimate security of Kyber only based on the LWE noise.

Resistance to hybrid attacks. Several schemes [12, 42] are potentially vulnerable to a hybrid attack [38, 45], mixing lattice reduction techniques with Meet-in-the-Middle combinatorial search. This attack is particularly difficult to analyze, and recent work [70] suggests that it is often not as competitive as previously thought. We note that this attack is especially relevant when secrets and errors are ternary and sparse, which is not the case for our design.

Algebraic attacks. The main novelty of our design is in the use of Module-LWE rather than Ring-LWE. One of the motivations for this change is to move further away from the recently uncovered weaknesses of ideal lattices [15, 24, 31, 32] – yet without the cost of using completely unstructured LWE. 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.

Scaling security and performance. A particularly attractive feature of Module-LWE (as compared to LWE or Ring-LWE) is, that scaling security only needs marginal changes to existing, possibly

highly optimized implementations. Specifically, the only parameters that need to change to scale security (and performance) of Kyber, are k and η; note that optimized code for polynomial arithmetic is not affected by changing those parameters. Table 2 lists one "paranoid" parameter set aiming at security similar to NewHope (using dimension $n \cdot k = 1024$) and one "light" parameter set that might become interesting for the 96-bit security level, or, with a tighter security analysis for the 128-bit security level, if continued effort in cryptanalysis does not produce significantly better attacks.

The Core-SVP hardness analysis against the best known quantum attacks yields 218 bits of security for the paranoid parameter set and 102 bits of security for the light parameter set.

A note on passively secure KEMs. We note that in order to support the CCA transformation, we need a negligible (in the cryptographic sense) failure probability. Previous proposals like NewHope [4] or Frodo [18] are designed to only achieve passive security and can live with much higher failure probabilities ($\approx 2^{-60}$ for NewHope and $2^{-38.9}$ for the recommended parameter set of Frodo). If one were to optimize a passively secure KEM from Module-LWE, one could reduce the rounding parameters d_u and d_t to $d_u = d_t = 10$ to further reduce public-key size (to 992 bytes) and ciphertext size (to 1088 bytes) while increasing the failure probability (to $2^{-71.9}$).

7 IMPLEMENTATION

In this section we give all the remaining details of our implementations of Kyber and report on performance of subroutines. Both implementations are fully protected against timing attacks. All cycle counts in this section were obtained on one core of an Intel Core-i7 4770K (Haswell) with hyperthreading and TurboBoost turned off running at 3.5 GHz. They are median cycle counts over 1000 measurements.

7.1 Primitives and encodings

Sections 3 and 4 introduce Kyber in abstract terms without fixing concrete instantiations of the functions H, G, and Sam, and without fixing encodings of messages. This subsection details concrete instantiations of these building blocks.

Symmetric primitives. The main symmetric building blocks are the two hash functions H and G, a function that accepts as input the public seed ρ and generates the uniform matrix $\mathbf{A} \in R_q^{k \times k}$, and a function that accepts as input a secret seed r and generates as output noise polynomials sampled from β_η. Note that in passively secure KEMs like BCNS [19], NewHope [4], or Frodo [18], the choice of how noise polynomials are sampled is a local decision: implementations on different platforms can choose whatever PRNG is the best option on the respective platform. This is also true for noise generation in Kyber's key generation, but, because of the CCA transform, is no longer true for noise generation in encapsulation.

We decided to instantiate all hash functions with the extendable output function SHAKE-128, standardized in FIPS 202 [58]. For the expansion of (public and secret) seeds we use the domain-separated version cSHAKE-128, that has recently been standardized in FIPS 800-185 [46]. All cSHAKE-128 domain separators in Kyber are 2 bytes long; we will denote them in the following as (i, j), where i is the byte at the lower address. With this choice, all symmetric primitives in Kyber rely on the same underlying primitive, namely the Keccak-$f1600$ permutation. The only exception is that for key generation, different implementations are free to use whatever PRNG is offering the best performance and security on their respective platform.

We are aware that another choice of symmetric primitives would yield somewhat better performance on most platforms. For example, we could have decided to use SHA256 for all hashes (with output extension for G via MGF1; see [57, App. B.2.1]), and AES in counter mode for the expansion of seeds. This choice would certainly be faster on platforms with hardware AES and SHA256 support. However, on platforms without hardware support, AES implementations are notorious for timing-attack vulnerabilities. Furthermore, as pointed out in [4, Sec. 3], the use of a PRG (which AES in counter mode is), is not helpful to argue security, because in the generation of $\mathbf{A}$, the input is *public*, whereas security of a PRG is only given for secret inputs.

Other possible choices of primitives that would yield better performance are be the ChaCha20 stream cipher [10] that has recently been standardized for TLS [50] or the BLAKE2X extendable output function [6]. Unfortunately, neither of these functions has received a lot of cryptanalytic attention, yet, so we prefer to stick to the conservative choice of SHAKE-128, which was standardized after years of cryptanalytic scrutiny through the course of the SHA-3 competition.

The NTT domain. Computing the discrete Fourier transform on elements from R_q can be done with methods analogous to the fast Fourier transform [29], except that operations on coefficients are defined in a finite field [62]. This is often referred to as the number theoretic transform (NTT). Before being able to define the expansion of the seed ρ into the matrix $\mathbf{A}$, we need to define the NTT domain of polynomials. Let $\omega = 3844 \in \mathbb{Z}_q$ and $\psi = \sqrt{\omega} = 62$, where ψ is chosen as the smallest element of multiplicative order 2^9 in $\mathbb{F}_q^* = \mathbb{F}_{7681}^*$.

For a polynomial $\mathbf{g} = \sum_{i=0}^{255} g_i X^i \in R_q$ we define the polynomial $\hat{\mathbf{g}}$ *in NTT domain* as

$$\mathsf{NTT}(\mathbf{g}) = \hat{\mathbf{g}} = \sum_{i=0}^{255} \hat{g}_i X^i, \text{ with } \hat{g}_i = \sum_{j=0}^{255} \psi^j g_j \omega^{ij}.$$

The inverse NTT^{-1} of the function NTT is essentially the same as the computation of NTT, except that it uses $\omega^{-1} \bmod q = 6584$, multiplies by powers of $\psi^{-1} \bmod q = 1115$ after the summation, and also multiplies each coefficient by the scalar $n^{-1} \bmod q = 7651$, so that

$$\mathsf{NTT}^{-1}(\hat{\mathbf{g}}) = \mathbf{g} = \sum_{i=0}^{255} g_i X^i, \text{ with } g_i = n^{-1}\psi^{-i} \sum_{j=0}^{255} \hat{g}_j \omega^{-ij}.$$

For two polynomials $\mathbf{f}, \mathbf{g} \in R_q$, the product $\mathbf{fg}$ can be computed as $\mathsf{NTT}^{-1}(\mathsf{NTT}(\mathbf{f}) \circ \mathsf{NTT}(\mathbf{g}))$, where $\circ$ denotes the point-wise multiplication.

Generation of A. Generation of the matrix $\mathbf{A} = (a_{i,j}) \in R_q^{k \times k}$ receives as input the public seed ρ. To generate the entry $a_{i,j} \in R_q$ we first expand ρ through cSHAKE-128 with the 2-byte domain separator (i, j). The output of this expansion is considered a stream

Table 2: Alternative parameter sets for higher and lower security levels.

	n	k	q	η	(d_u, d_v, d_t)	δ	pq sec.	$\|pk\|$ in bytes	$\|c\|$ in bytes
Paranoid	256	4	7681	3	(11,3,11)	2^{-145}	218	1 440	1 536
Light	256	2	7681	5	(11,3,11)	2^{-169}	102	736	832

of 16-bit little-endian integers. On this sequence of 16-bit integers we run rejection sampling as follows: first set the upper 3 bits of the integers to zero, then use the resulting integer as coefficient for $a_{i,j}$ if it is smaller than q, otherwise discard it and move to the next 16-bit integer. Fill the polynomial $a_{i,j}$ starting from the constant coefficient moving to the coefficient belonging to X^{n-1}. The resulting polynomial $a_{i,j}$ is assumed to be in NTT domain. Note that this generation of $\mathbf{A}$ exhibits k^2-way parallelism for the expansion of ρ.

Generation of noise polynomials. Noise polynomials in Kyber are sampled from β_4. To obtain such a noise polynomial we first expand a seed to an array of $n = 256$ uniformly random bytes $(r_0, \ldots, r_{255})$. We then generate coefficient e_i of a noise polynomial $\mathbf{e} = \sum_{i=0}^{255} e_i X^i$ by subtracting the Hamming weight of the most significant nibble of r_i from the Hamming weight of the least significant nibble of r_i. As stated above, the choice of how the 256 uniformly random bytes are generated during key generation is a local, platform-dependent choice. During encapsulation we again use cSHAKE-128 with 2-byte domain separators to expand the 32-byte secret r to 256 bytes. To generate $\mathbf{r} = (r_0, r_1, r_2)$ we use domain separators $(0,0)$, $(1,0)$, and $(2,0)$; to generate $\mathbf{e}_1 = (e_{1,0}, e_{1,1}, e_{1,2})$ we use domain separators $(3,0)$, $(4,0)$, and $(5,0)$; and to generate e_2 we use domain separator $(6,0)$. Note that $\mathbf{r}$ is used as input to a multiplication, which is computed via the fast negacyclic NTT operation outlined above. In order to compute the NTT inplace, we would first have to bit-reverse the order of coefficients in $\mathbf{r}$. As in NewHope, we omit this bitreversal, and instead assume the coefficients of $\mathbf{r}$ to be in bit-reversed order.

Inputs to *G* and *H*. The description of the CCA transform includes the public key pk as input of G and includes the ciphertext $c = (\mathbf{u}, v, d)$ as input of H. In the implementation we instead include $\mathrm{H}(pk)$ and $\mathrm{H}(c)$. The two additional hashes may seem redundant, but simplify implementation with a non-incremental API for G and H. Furthermore using $\mathrm{H}(pk)$ instead of pk as input to G enables a small speedup for decapsulation at the cost of a slightly increased secret-key size as explained in the next paragraph.

Encoding of keys and ciphertexts. In NewHope, polynomials in public keys and the ciphertext are in NTT domain; in Kyber all polynomials sent over the channel are in normal domain. This is necessary for the compression through rounding (see Section 3) to work.

A Kyber public key is a tuple $(\mathbf{t}, \rho)$, where $\mathbf{t}$ is a vector of three polynomials with 256 11-bit coefficients each, and ρ is a 32-byte seed. We encode the polynomials in compressed little-endian format to fit it in $(256 \cdot 11)/8 = 352$ bytes, concatenate the compressed three polynomials and finally concatenate ρ to obtain public keys of $3 \cdot 352 + 32 = 1088$ bytes.

A Kyber secret key is a vector of three polynomials in NTT domain with 256 13-bit coefficients each. We store these polynomials in compressed little-endian format resulting in a total of $(3 \cdot 256 \cdot 13)/8 = 1248$ bytes. For re-encapsulation during decapsulation we additionally need the public key, which we simply concatenate and store as part of the secret key. Finally, we also concatenate $\mathrm{H}(pk)$ to avoid having to compute this hash during decapsulation and concatenates the 32 bytes of the value z that is used to compute the pseudo-random returned key when re-encapsulation fails. This results in a total size of $1248 + 1088 + 32 + 32 = 2400$ bytes for the secret key.

A Kyber ciphertext is a 3-tuple $(\mathbf{u}, v, d)$, where $\mathbf{u}$ is a vector of three polynomials with 256 11-bit coefficients each, v is a polynomial with 256 3-bit coefficients, and d is a 32-byte hash. Using the same compressed little-endian format for polynomials as for keys we obtain ciphertexts with a total size of $3 \cdot 352 + (3 \cdot 256)/8 + 32 = 1184$ bytes.

Size-speed tradeoffs. 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 $\mathrm{H}(pk)$ 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. 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.

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-encapsuation part of decapsulation.

All performance results reported in the following assume the secret-key format described in the previous paragraph; i.e., with polynomials in NTT domain, including the public key and $\mathrm{H}(pk)$, but not including $\mathbf{A}$.

7.2 Reference implementation

Kyber's reference implementation in C follows much in the spirit of the NewHope reference implementation described in [4, Sec. 7.2]. In particular, it only relies on 16-bit and 32-bit integer arithmetic (outside of Keccak) and uses the same combination of short Barrett reductions and Montgomery reductions to accelerate the NTT computation. One consequence of the modulus $q = 7681$ is that the short Barrett reduction becomes slightly more efficient; an unsigned 16-bit integer `a` can be reduced to an unsigned integer `r` between 0 and 11768 and congruent modulo q via the following three operations:

```
u = a >> 13;
u *= KYBER_Q;
r = a - u;
```

7.3 AVX2 implementation

Modern 64-bit Intel processors feature the AVX2 vector-instruction set that supports operations on 256-bit vectors that can be interpreted as vectors of 8 single-precision or 4-double-precision floating-point numbers, or as vectors of integers of various sizes. These AVX2 instructions were also used for the optimized implementation of the optimized NewHope software described in [4, Sec. 7].

Polynomial arithmetic. For polynomial arithmetic we represent polynomials as arrays of double-precision floating-point numbers. This representation results in a very fast NTT computation as first described in [41, Sec. 3.2] and also used for NewHope in [4]. Essentially, our implementation of the NTT follows the same approach as [41] and [4], except that we carefully optimize for $n = 256$ and $q = 7681$. Specifically, we merge levels 0–4 and then merge levels 5–7 to reduce load and store operations. The 13-bit modulus q allows us to reduce coefficients modulo q only every 3 levels. We reduce after level 1 (i.e., after 2 levels), then again after level 4, and finally after level 7. To make best use of the 53-bit radix of double-precision floats, we precompute powers of ω in the range $[-q/2, q/2]$ and also reduce to this range inside the NTT. Only the last modular reduction goes back to unsigned representation. One NTT takes 1 992 cycles; an NTT^{-1} operation includes a bit reversal and takes 2 632 cycles.

We also use vectorized double-precision floating point arithmetic for pointwise multiplication and polynomial addition and subtraction.

Vectorized Keccak. As mentioned earlier, Keccak has a reputation of not being particularly fast in software. One reason is that Keccak is very hard to vectorize; in fact, according to the eBACS benchmarks, the fastest implementation of Keccak on Intel Haswell processors is the non-vectorized "simple64" implementation.

The picture changes drastically if a protocol can compute multiple independent streams of SHA-3, SHAKE, or cSHAKE on inputs and outputs of the same length. More specifically, the Keccak code package [14] includes an implementation for AVX2 that computes 4 independent streams in parallel. We make use of this 4-way parallel implementation in the expansion of ρ involved in the generation of the matrix $\mathbf{A}$ and also in the generation of noise polynomials during encapsulation. Specifically, for the generation of $\mathbf{A}$, we generate 8 streams of uniformly random 16-bit numbers via two calls to this function, leaving only one sequential SHAKE-128 call. In encapsulation we generate 8 arrays of 256 uniformly random bytes via two calls to 4-way parallel cSHAKE-128 and discard one of those arrays. The speedup from vectorized Keccak is crucial: compared to NewHope, Kyber needs to generate more than twice as many uniformly random polynomial coefficients, yet, with 34 304 cycles, generation of the matrix $\mathbf{a}$ is about as fast as generation of the equivalent value $\mathbf{a}$ in NewHope.

Rejection sampling. Part of the generation of $\mathbf{A}$ is rejection sampling on the stream of 16-bit integers produced by the cSHAKE-128 expansion. We adopt the fast vectorized approach described in [40] for this task. One difference is that we do not need to first conditionally subtract q four times; we simply eliminate the upper 3 bits of each 16-bit integer in a 256-bit vector through one mask instructions and then compare to a constant vector filled with 16-bit copies of q.

7.4 Flexibility of Kyber

One possible use of Kyber is for ephemeral key exchange, for example in TLS 1.2 as illustrated by [19] and by Google's post-quantum TLS experiment [21] with NewHope.[5] Indeed, the experiment concluded that they "did not find any unexpected impediment to deploying something like NewHope" [49] and Kyber features performances close to the one of NewHope but with smaller sizes.

However, the CCA security of Kyber makes it a much more versatile tool. Not only is it possible to cache ephemeral keys for some time (which would be a security disaster for BCNS, Frodo, or NewHope), we can also use it for classical IND-CCA public-key encryption of messages of arbitrary length [33] (cf. the hybrid CCA-secure scheme of Appendix A) and for authenticated key exchange protocols, as described in Fig. 3. The Kyber software package includes implementations of the unilaterally authenticated key exchange Kyber.UAKE described in Fig. 2 and the mutually authenticated key exchange Kyber.AKE described in Fig. 3.

8 PERFORMANCE RESULTS AND COMPARISON

In this section we report on the performance of our standalone implementations of Kyber, Kyber-based authenticated key exchange, and an integration of Kyber within the Open Quantum Safe (OQS) framework[6] [67].

8.1 Standalone Kyber

In Table 3 we give performance results of the standalone implementations of Kyber and compare them to results from the literature on lattice-based KEMS, key-exchange protocols, and encryption schemes. We compiled the Kyber software with `gcc-4.9.2` with optimization flags `-O3 -fomit-frame-pointer -msse2avx -mavx2 -march=corei7-avx`, except for the non-vectorized implementation of Keccak, which we compile with `clang-3.5.10` with flags `-march=native -O3 -fomit-frame-pointer -fwrapv -Qunused-arguments`.

To give an indication of security levels obtained by the different schemes we include the core-SVP hardness estimation ("Sec. estim.") following the approach from [4]. Note that this estimate does not say anything about the applicability of hybrid or algebraic attacks.

8.2 Kyber-based authenticated key exchanges

To illustrate one use case of Kyber and to establish a data point for high-performance post-quantum authenticated key exchanges, the Kyber software package includes implementations of Kyber.AKE and Kyber.UAKE. The performance in terms of message sizes and CPU cycles (for our AVX2 optimized software) is summarized in

[5] Note that one can easily combine KEMs (e.g., Kyber with a pre-quantum KEM) by hashing the shared secret keys together.
[6] https://www.openquantumsafe.org

Table 3: Comparison of lattice-based KEMs and public-key encryption. Benchmarks were performed on an Intel Core i7-4770K (Haswell) if not indicated otherwise. Cycles are stated for key generation (K), encapsulation/encryption (E), and decapsulation/decryption (D) Bytes are given for secret keys (sk), public keys (pk), and ciphertexts (c). The column "ct?" indicates whether the software is running in constant time, i.e., with protection against timing attacks.

Scheme	Sec. estim.	Prob.	ct?		Cycles		Bytes
Passively secure KEMs							
BCNS [19]	78[a]	Ring-LWE	yes	**K:**	≈ 2 477 958	**sk:**	4096
				E:	≈ 3 995 977	**pk:**	4096
				D:	≈ 481 937	**c:**	4224
NewHope [4]	255[a]	Ring-LWE	yes	**K:**	88 920	**sk:**	1792
(AVX2 optimized)				**E:**	110 986	**pk:**	1824
				D:	19 422	**c:**	2048
Frodo [18]	130[a]	LWE	yes	**K:**	≈ 2 938 000[b]	**sk:**	11 280
(recommended parameters)				**E:**	≈ 3 484 000[b]	**pk:**	11 296
				D:	≈ 338 000[b]	**c:**	11 288
CCA-secure KEMs							
NTRU Prime [12]	129[a]	NTRU[k]	yes	**K:**	?[c]	**sk:**	1417
				E:	> 51488[c]	**pk:**	1232
				D:	?[c]	**c:**	1141
spLWE-KEM [27]	128[i]	spLWE	?	**K:**	≈ 336 700[d]	**sk:**	?
(128-bit PQ parameters)				**E:**	≈ 813 800[d]	**pk:**	?
				D:	≈ 785 200[d]	**c:**	804
Kyber (this paper)	161[i]	Module-LWE	yes	**K:**	276 720	**sk:**	2368
(C reference)				**E:**	332 800	**pk:**	1088
				D:	376 104	**c:**	1184
Kyber (this paper)	161[i]	Module-LWE	yes	**K:**	77 892	**sk:**	2400
(AVX2 optimized)				**E:**	119 652	**pk:**	1088
				D:	125 736	**c:**	1184
CCA-secure public-key encryption							
NTRUEncrypt ees743ep1[42]	159[a]	NTRU	no	**K:**	1 194 816	**sk:**	1 120
				E:	57 440	**pk:**	1 027
				D:	110 604	**c:**	980
Lizard [28]	128[i]	LWE+LWR	no	**K:**	97 573 000[f]	**sk:**	466 944[g,h]
(recommended parameters)				**E:**	≈ 35 050[f]	**pk:**	2 031 616[h]
				D:	≈ 80 840[f]	**c:**	1072

[a] According to the conservative "best known quantum attack" estimates from [4].
[b] Benchmarked on a 2.6GHz Intel Xeon E5 (Sandy Bridge).
[c] The NTRU Prime paper reports benchmarks only for polynomial multiplication.
[d] Benchmarked on "PC (Macbook Pro) with 2.6GHz Intel Core i5".
[e] Benchmarked by eBACS [13] on Intel Xeon E3-1275 (Haswell).
[f] As reported by the software from https://github.com/LizardOpenSource/Lizard_c, compiled with gcc-6.3 with flags `-O3 -fomit-frame-pointer -msse2avx -mavx2 -march=native` on Intel Core i7-4770K.
[g] Unlike our scheme, the paper reports secret-key size without the public key required for decryption in the Targhi-Unruh transform.
[h] Sizes used by the software; those could be compressed by a factor 1.6, incurring only small computational overhead.
[i] According to the conservative "best known quantum attack" estimates from [4], with appropriate adaptations (balanced lattice attacks [28, Sec. 4.2]).
[k] The problem underlying NTRU Prime is subtly different than in NTRU; it uses a different ring than commonly used in NTRU and uses deterministic noise.

Table 4: Message sizes and cycle counts for Kyber.UAKE **and** Kyber.AKE**.**

	Bytes		Cycles		
	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_1$	P_1(start)	P_2	P_1(end)
UAKE	2 272	1 184	197 980	248 388	253 448
AKE	2 272	2 368	195 464	364 188	379 036

Table 5: Timings for post-quantum key exchanges (C reference implementations) in Open Quantum Safe.

Scheme	Operations (μs)		
	P_1 (start)	P_2	P_1 (end)
SIDH [30]	15 015	33 530	14 241
McBits [11]	170 892	53	133
Frodo [18]	3 436	4 027	105
BCNS [19]	1 087	1 729	178
NewHope [53]	60	104	19
NewHope [4]	69	107	18
Kyber	77	100	110

Table 4. The only paper describing an implementation of lattice-based authenticated key exchange that we are aware of is [71]. Our software outperforms the results of [71] by more than two orders of magnitude.

8.3 Integration with OQS

The Open Quantum Safe (OQS) aims at supporting the development and prototyping of quantum-resistant cryptography. In particular, it proposes a common API for post-quantum key exchange algorithms, making it easy to integrate new algorithms and compare to alternatives. We integrated the reference C implementation of our CCA-secure KEM Kyber in OQS library, `liboqs`, and ran the benchmarking command `./test_kex --bench`; the results are provided in Table 5.

Note that the timings differ slightly from those of Table 3. Indeed, `liboqs` provides reference implementations for randomness sampling, AES, Keccak and ChaCha20; our reference implementation has been minimally modified to be compatible with those API. Also, only the reference C implementation of NewHope is (as of today) integrated to `liboqs` and not the AVX2-optimized version listed in Table 3. Finally, all implementations have been compiled with the same flags, which does not necessary reflect the full potential of every implementation. Note that, once integrated to `liboqs`, it is easy to integrate Kyber to OQS's fork of OpenSSL 1.0.2.[7]

ACKNOWLEDGMENTS

The authors would like to thank Isis Lovecruft for suggesting the name Kyber and Andreas Hülsing for very helpful discussions.

This work is supported by a Veni Innovational Research Grant from NWO under project number 639.021.645. This work is supported by Canada's NSERC CREATE program. IQC is supported in part by the Government of Canada and the Province of Ontario. Eike Kiltz was supported by the ERC Consolidator Grant ERC-2013-CoG-615073-ERCC. Vadim Lyubashevsky was supported by the the SNSF ERC Transfer Starting Grant CRETP2-166734-FELICITY and the H2020 Project Safecrypto. The last author was supported by the ERC Starting Grant ERC-2013-StG-335086-LATTAC.

[7] https://github.com/open-quantum-safe/openssl

REFERENCES

[1] Miklós Ajtai and Cynthia Dwork. 1997. A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence. In *29th ACM STOC*. ACM Press, 284–293.

[2] Martin R. Albrecht, Shi Bai, and Léo Ducas. 2016. A Subfield Lattice Attack on Overstretched NTRU Assumptions - Cryptanalysis of Some FHE and Graded Encoding Schemes. In *CRYPTO 2016, Part I (LNCS)*, Matthew Robshaw and Jonathan Katz (Eds.), Vol. 9814. Springer, Heidelberg, 153–178. https://doi.org/10.1007/978-3-662-53018-4_6

[3] Michael Alekhnovich. 2003. More on Average Case vs Approximation Complexity. In *44th FOCS*. IEEE Computer Society Press, 298–307.

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

[5] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. 2009. Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems. In *CRYPTO 2009 (LNCS)*, Shai Halevi (Ed.), Vol. 5677. Springer, Heidelberg, 595–618.

[6] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. 2016. BLAKE2X. (2016). https://blake2.net/blake2x.pdf.

[7] Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. 2015. Improved Security Proofs in Lattice-Based Cryptography: Using the Rényi Divergence Rather Than the Statistical Distance. In *ASIACRYPT 2015, Part I (LNCS)*, Tetsu Iwata and Jung Hee Cheon (Eds.), Vol. 9452. Springer, Heidelberg, 3–24. https://doi.org/10.1007/978-3-662-48797-6_1

[8] Abhishek Banerjee, Chris Peikert, and Alon Rosen. 2012. Pseudorandom Functions and Lattices. In *EUROCRYPT 2012 (LNCS)*, David Pointcheval and Thomas Johansson (Eds.), Vol. 7237. Springer, Heidelberg, 719–737.

[9] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM CCS 93*, V. Ashby (Ed.). ACM Press, 62–73.

[10] Daniel J. Bernstein. 2008. ChaCha, a variant of Salsa20. In *Workshop Record of SASC 2008: The State of the Art of Stream Ciphers*. http://cr.yp.to/papers.html#chacha.

[11] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. 2013. McBits: Fast Constant-Time Code-Based Cryptography. In *CHES 2013 (LNCS)*, Guido Bertoni and Jean-Sébastien Coron (Eds.), Vol. 8086. Springer, Heidelberg, 250–272. https://doi.org/10.1007/978-3-642-40349-1_15

[12] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. 2016. NTRU Prime. Cryptology ePrint Archive, Report 2016/461. (2016). http://eprint.iacr.org/2016/461.

[13] Daniel J. Bernstein and Tanja Lange. eBACS: ECRYPT Benchmarking of Cryptographic Systems. (????). http://bench.cr.yp.to (accessed 2017-05-19).

[14] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, , and Ronny Van Keer. 2017. Keccak Code Package. (2017). https://github.com/gvanas/KeccakCodePackage (accessed 2017-05-17).

[15] Jean-François Biasse and Fang Song. 2016. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In *27th SODA*, Robert Krauthgamer (Ed.). ACM-SIAM, 893–902. https://doi.org/10.1137/1.9781611974331.ch64

[16] Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. 2016. On the Hardness of Learning with Rounding over Small Modulus. In *TCC 2016-A (LNCS)*, Vol. 9562. Springer, Heidelberg, 209–224.

[17] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. 2011. Random Oracles in a Quantum World. In *ASIACRYPT 2011 (LNCS)*, Dong Hoon Lee and Xiaoyun Wang (Eds.), Vol. 7073. Springer, Heidelberg, 41–69.

[18] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. 2016. Frodo: Take off the Ring! Practical, Quantum-Secure Key Exchange from LWE. In *ACM CCS 16*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 1006–1018.

[19] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. 2015. Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 553–570. https://doi.org/10.1109/SP.2015.40

[20] Colin Boyd, Yvonne Cliff, Juan Gonzalez Nieto, and Kenneth G. Paterson. 2008. Efficient One-Round Key Exchange in the Standard Model. In *ACISP 08 (LNCS)*,

Yi Mu, Willy Susilo, and Jennifer Seberry (Eds.), Vol. 5107. Springer, Heidelberg, 69–83.

[21] Matt Braithwaite. 2016. Experimenting with Post-Quantum Cryptography. Posting on the Google Security Blog. (2016). https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html.

[22] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS'12*. ACM, 309–325.

[23] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. 2013. Classical hardness of learning with errors. In *45th ACM STOC*, Dan Boneh, Tim Roughgarden, and Joan Feigenbaum (Eds.). ACM Press, 575–584.

[24] Peter Campbell, Michael Groves, and Dan Shepherd. 2014. Soliloquy: A cautionary tale. In *ETSI 2nd Quantum-Safe Crypto Workshop*. 1–9.

[25] Ran Canetti and Hugo Krawczyk. 2001. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *EUROCRYPT 2001 (LNCS)*, Birgit Pfitzmann (Ed.), Vol. 2045. Springer, Heidelberg, 453–474.

[26] L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone. 2016. Report on Post-Quantum Cryptography. NISTIR 8105. (2016). http://dx.doi.org/10.6028/NIST.IR.8105.

[27] Jung Hee Cheon, Kyoohyung Han, Jinsu Kim, Changmin Lee, and Yongha Son. 2017. A Practical Post-Quantum Public-Key Cryptosystem Based on spLWE. In *Information Security and Cryptology – ICISC 2016 (LNCS)*, Seokhie Hong and Jong Hwan Park (Eds.), Vol. 10157. Springer, 51–74. https://eprint.iacr.org/2016/1055.

[28] Jung Hee Cheon, Duhyeong Kim, Joohee Lee, and Yong Soo Song. 2016. Lizard: Cut off the Tail! // Practical Post-Quantum Public-Key Encryption from LWE and LWR. IACR Cryptology ePrint Archive report 2016/1126. (2016). http://eprint.iacr.org/2016/1126.

[29] James W. Cooley and John W. Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation* 19, 90 (1965), 297–301.

[30] Craig Costello, Patrick Longa, and Michael Naehrig. 2016. Efficient Algorithms for Supersingular Isogeny Diffie-Hellman. In *CRYPTO 2016, Part I (LNCS)*, Matthew Robshaw and Jonathan Katz (Eds.), Vol. 9814. Springer, Heidelberg, 572–601. https://doi.org/10.1007/978-3-662-53018-4_21

[31] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. 2016. Recovering Short Generators of Principal Ideals in Cyclotomic Rings. In *EUROCRYPT 2016, Part II (LNCS)*, Marc Fischlin and Jean-Sébastien Coron (Eds.), Vol. 9666. Springer, Heidelberg, 559–585. https://doi.org/10.1007/978-3-662-49896-5_20

[32] Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. 2017. Short Stickelberger Class Relations and Application to Ideal-SVP. In *EUROCRYPT (1) (Lecture Notes in Computer Science)*, Vol. 10210. 324–348.

[33] Ronald Cramer and Victor Shoup. 2003. Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack. *SIAM J. Comput.* 33, 1 (2003), 167–226.

[34] M. H. Devoret and R. J. Schoelkopf. 2013. Superconducting circuits for quantum information: an outlook. *Science* 339, 6124 (2013), 1169–1174.

[35] Jintai Ding, Xiang Xie, and Xiaodong Lin. 2012. A Simple Provably Secure Key Exchange Scheme Based on the Learning with Errors Problem. Cryptology ePrint Archive, Report 2012/688. (2012). http://eprint.iacr.org/2012/688.

[36] Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama. 2012. Strongly Secure Authenticated Key Exchange from Factoring, Codes, and Lattices. In *PKC 2012 (LNCS)*, Marc Fischlin, Johannes Buchmann, and Mark Manulis (Eds.), Vol. 7293. Springer, Heidelberg, 467–484.

[37] Eiichiro Fujisaki and Tatsuaki Okamoto. 1999. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *CRYPTO'99 (LNCS)*, Michael J. Wiener (Ed.), Vol. 1666. Springer, Heidelberg, 537–554.

[38] Florian Göpfert, Christine van Vredendaal, and Thomas Wunderer. 2017. A Quantum Attack on LWE with Arbitrary Error Distribution. IACR Cryptology ePrint Archive report 2017/221. (2017). http://eprint.iacr.org/2017/221.

[39] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *28th ACM STOC*. ACM Press, 212–219.

[40] Shay Gueron and Fabian Schlieker. 2016. Speeding up R-LWE post-quantum key exchange. In *Secure IT Systems (LNCS)*, Billy Bob Brumley and Juha Röning (Eds.), Vol. 10014. Springer, 187–198. https://eprint.iacr.org/2016/467.

[41] Tim Güneysu, Tobias Oder, Thomas Pöppelmann, and Peter Schwabe. 2013. Software speed records for lattice-based signatures. In *Post-Quantum Cryptography (LNCS)*, Philippe Gaborit (Ed.), Vol. 7932. Springer, 67–82. http://cryptojedi.org/papers/#lattisigns.

[42] Jeffrey Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang. 2017. Choosing Parameters for NTRUEncrypt. In *CT-RSA (Lecture Notes in Computer Science)*, Vol. 10159. Springer, 3–18.

[43] Jeffrey Hoffstein, Jull Pipher, and Joseph H. Silverman. 1998. NTRU: A ring-based public key cryptosystem. In *Algorithmic number theory (LNCS)*, Joe P. Buhler (Ed.), Vol. 1423. Springer, 267–288. https://www.securityinnovation.com/uploads/Crypto/ANTS97.ps.gz.

[44] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. 2017. A Modular Analysis of the Fujisaki-Okamoto Transformation. IACR Cryptology ePrint Archive report 2017/604. (2017). http://eprint.iacr.org/2017/604.

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

[46] John Kelsey, Shu jen Chang, and Ray Perlner. 2016. SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash. FIPS Special Publication 800-185. (2016). http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf.

[47] Paul Kirchner and Pierre-Alain Fouque. 2017. Revisiting Lattice Attacks on Overstretched NTRU Parameters. In *EUROCRYPT (1) (Lecture Notes in Computer Science)*, Vol. 10210. 3–26.

[48] Thijs Laarhoven. 2015. *Search problems in cryptography*. Ph.D. Dissertation. Eindhoven University of Technology. http://www.thijs.com/docs/phd-final.pdf.

[49] Adam Langley. 2016. CECPQ1 results. Posting on Adam Langley's Personal Blog. (2016). https://www.imperialviolet.org/2016/11/28/cecpq1.html.

[50] A. Langley, W. Chang, N. Mavrogiannopoulos, J. Strombergson, and S. Josefsson. 2016. ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS). RFC 7905. (2016). https://tools.ietf.org/html/rfc7905.

[51] Adeline Langlois and Damien Stehlé. 2015. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography* 75, 3 (2015), 565–599. https://doi.org/10.1007/s10623-014-9938-4

[52] Patrick Longa and Michael Naehrig. 2016. Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography. IACR Cryptology ePrint Archive report 2016/504. (2016). https://eprint.iacr.org/2016/504.

[53] Patrick Longa and Michael Naehrig. 2016. Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography. In *CANS 16 (LNCS)*, Sara Foresti and Giuseppe Persiano (Eds.), Vol. 10052. Springer, Heidelberg, 124–139. https://doi.org/10.1007/978-3-319-48965-0_8

[54] Vadim Lyubashevsky and Daniele Micciancio. 2006. Generalized Compact Knapsacks Are Collision Resistant. In *ICALP 2006, Part II (LNCS)*, Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener (Eds.), Vol. 4052. Springer, Heidelberg, 144–155.

[55] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors over Rings. In *EUROCRYPT 2010 (LNCS)*, Henri Gilbert (Ed.), Vol. 6110. Springer, Heidelberg, 1–23.

[56] Daniele Micciancio. 2002. Improved cryptographic hash functions with worst-case/average-case connection. In *34th ACM STOC*. ACM Press, 609–618.

[57] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch. 2016. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017. (2016). https://tools.ietf.org/html/rfc8017.

[58] National Institute of Standards and Technology. 2015. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. FIPS PUB 202. (2015). http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.

[59] Chris Peikert. 2009. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *41st ACM STOC*, Michael Mitzenmacher (Ed.). ACM Press, 333–342.

[60] Chris Peikert. 2014. Lattice Cryptography for the Internet. In *Post-Quantum Cryptography (Lecture Notes in Computer Science)*, Michele Mosca (Ed.), Vol. 8772. Springer, 197–219. https://doi.org/10.1007/978-3-319-11659-4_12

[61] Chris Peikert and Alon Rosen. 2006. Efficient Collision-Resistant Hashing from Worst-Case Assumptions on Cyclic Lattices. In *TCC 2006 (LNCS)*, Shai Halevi and Tal Rabin (Eds.), Vol. 3876. Springer, Heidelberg, 145–166.

[62] John M. Pollard. 1971. The fast Fourier transform in a finite field. *Mathematics of computation* 25, 114 (1971), 365–374.

[63] Thomas Pöppelmann and Tim Güneysu. 2014. Towards Practical Lattice-Based Public-Key Encryption on Reconfigurable Hardware. In *SAC 2013 (LNCS)*, Tanja Lange, Kristin Lauter, and Petr Lisonek (Eds.), Vol. 8282. Springer, Heidelberg, 68–85. https://doi.org/10.1007/978-3-662-43414-7_4

[64] Charles Rackoff and Daniel R. Simon. 1992. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In *CRYPTO'91 (LNCS)*, Joan Feigenbaum (Ed.), Vol. 576. Springer, Heidelberg, 433–444.

[65] Oded Regev. 2003. New lattice based cryptographic constructions. In *35th ACM STOC*. ACM Press, 407–416.

[66] Oded Regev. 2005. On lattices, learning with errors, random linear codes, and cryptography. In *37th ACM STOC*, Harold N. Gabow and Ronald Fagin (Eds.). ACM Press, 84–93.

[67] Douglas Stebila and Michele Mosca. 2016. Post-Quantum Key Exchange for the Internet and the Open Quantum Safe Project. IACR Cryptology ePrint Archive report 2016/1017. (2016). http://eprint.iacr.org/2016/1017.

[68] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. 2009. Efficient Public Key Encryption Based on Ideal Lattices. In *ASIACRYPT'09 (LNCS)*, Vol. 5912. Springer, Heidelberg, 617–635.

[69] Ehsan Ebrahimi Targhi and Dominique Unruh. 2016. Post-Quantum Security of the Fujisaki-Okamoto and OAEP Transforms. In *TCC 2016-B, Part II (LNCS)*, Martin Hirt and Adam D. Smith (Eds.), Vol. 9986. Springer, Heidelberg, 192–216. https://doi.org/10.1007/978-3-662-53644-5_8

[70] Thomas Wunderer. 2016. Revisiting the Hybrid Attack: Improved Analysis and Refined Security Estimates. Cryptology ePrint Archive, Report 2016/733. (2016). http://eprint.iacr.org/2016/733.
[71] Jiang Zhang, Zhenfeng Zhang, Jintai Ding, Michael Snook, and Özgür Dagdelen. 2015. Authenticated Key Exchange from Ideal Lattices. In *EUROCRYPT 2015, Part II (LNCS)*, Elisabeth Oswald and Marc Fischlin (Eds.), Vol. 9057. Springer, Heidelberg, 719–751. https://doi.org/10.1007/978-3-662-46803-6_24

A THE CCA-SECURE ENCRYPTION SCHEME

We use the canonical way proposed by Cramer and Shoup to compose Kyber, our secure key encapsulation mechanism (KEM), with a secure one-time symmetric-key encryption (SKE, or DEM) scheme [33]. We call Kyber.Hybrid the resulting hybrid encryption scheme.

On the choice of a symmetric encryption scheme. Any SKE scheme that is (one-time) secure against chosen-ciphertext attacks and with key space $\mathcal{K} = \{0,1\}^{256}$ can be combined with our key encapsulation mechanism Kyber. Typical examples include AES-OCB, AES-GCM or ChaCha20-Poly1305. Depending on one's application *and architecture*, different needs and choices for the symmetric encryption scheme are possible; we decide in this paper to not restrict ourselves to a specific application nor to a specific cipher. Additionally to the previously mentioned ciphers, several submissions to the Caesar competition for authenticated encryption are serious candidates for SKE.

Description of Kyber.Hybrid. We describe the public-key hybrid encryption scheme Kyber.Hybrid = (KeyGen, Enc, Dec) in Algorithms 6 to 8, assuming a SKE (E, D) where the encryption algorithm E takes as input a key in $\mathcal{K} = \{0,1\}^{256}$ and a message in $\{0,1\}^*$ and outputs a ciphertext, and where the decryption algorithm D takes as input a key and a ciphertext and outputs a message (or the rejection symbol $\perp$).

Algorithm 6 Kyber.Hybrid.KeyGen()

1: $(pk := (\rho, \mathbf{t}), sk := (\mathbf{s}, \rho, \mathbf{t})) \leftarrow$ Kyber.KeyGen()
2: **return** (pk, sk)

Algorithm 7 Kyber.Hybrid.Enc($pk = (\rho, \mathbf{t}), m$)

1: $(c, K) \leftarrow$ Kyber.Encaps(pk)
2: $c' := E(K, m)$
3: **return** $c'' := (c, c')$

Algorithm 8 Kyber.Hybrid.Dec($sk = (\mathbf{s}, z, \rho, \mathbf{t}), c'' = (c, c')$)

1: $K :=$ Kyber.Decaps(sk, c)
2: **return** $m := D(K, c')$

Correctness and security. The correctness and security of our hybrid encryption scheme Kyber.Hybrid follow from those of the KEM and the chosen SKE [33, Th. 5].