NewHope

Algorithm Specifications and Supporting Documentation

Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas,
Antonio de la Piedra, Thomas Pöppelmann, Peter Schwabe, Douglas Stebila

Version 1.0 - (Updated June 14, 2018)

Contents

Errata and updates

In Table 1 we list errata and updates to the specification document.

Table 1: Errata and updates to the document.

Type	Date	Description
Typo fix	June 2018	The secret key is supposed to be stored in the NTT domain during key generation. Due to a typo this was not specified correctly and has been fixed in Algorithm 1 by changing $sk = \mathsf{EncodePolynomial}(\mathbf{s})$ to $sk = \mathsf{EncodePolynomial}(\hat{\mathbf{s}})$. We are thankful to James Howe for reporting this issue.

1 Written specification

1.1 Mathematical background

1.1.1 Basic definitions

Let $\mathbb{Z}$ be the ring of rational integers. We define for an $x \in \mathbb{R}$ the rounding function $\lfloor x \rceil = \lfloor x + \frac{1}{2} \rfloor \in \mathbb{Z}$. Let $\mathbb{Z}_q$, for an integer $q \geq 1$, denote the quotient ring $\mathbb{Z}/q\mathbb{Z}$. We define $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ as the ring of integer polynomials modulo $X^n + 1$. By $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ we mean the ring of integer polynomials modulo $X^n + 1$ where each coefficient is reduced modulo q. In case χ is a probability distribution over $\mathcal{R}$, then $x \xleftarrow{\$} \chi$ means the sampling of $x \in \mathcal{R}$ according to χ.

For a probabilistic algorithm $\mathcal{A}$ we denote by $y \xleftarrow{\$} \mathcal{A}$ that the output of $\mathcal{A}$ is assigned to y and that $\mathcal{A}$ is running with randomly chosen coins. We recall the discrete Gaussian distribution $D_{\mathbb{Z},\sigma}$, which is parametrized by the Gaussian parameter $\sigma \in \mathbb{R}$ and defined by assigning a weight proportional to $\exp(\frac{-x^2}{2\sigma^2})$ to all integers x.

The Euclidean length $\|\mathbf{v}\|$ of a vector $\mathbf{v} = (v_1, \dots v_n) \in \mathbb{R}^n$ is defined as $\|\mathbf{v}\| = \sqrt{\sum_{i=1}^{n} v_i^2}$. A lattice is a discrete subgroup of a finite dimensional Euclidean vector space, *i.e.* a discrete subgroup $\mathcal{L} \subset \mathbb{R}^n$. The minimal distance of a lattice is defined as the Euclidean length of its shortest non-zero vector, namely $\lambda_1(\mathcal{L}) = \min_{\mathbf{v} \in \mathcal{L} \setminus \{\mathbf{0}\}} \|\mathbf{x}\|$.

1.1.2 Computational problems on lattices

The shortest vector problem (SVP) and the closest vector problem (CVP) are two fundamental problems in lattices and their conjectured intractability is the foundation for a large number of cryptographic applications of lattices.

The (Approximate) Shortest Vector Problem, (SVP), statement from [106]. The shortest vector problem (SVP) asks, given a lattice basis $\mathbf{B}$, to find a shortest nonzero lattice vector, i.e., a vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ with $\|\mathbf{v}\| = \lambda_1(\mathcal{L}(\mathbf{B}))$. In the γ-approximate SVP_γ, for $\gamma \geq 1$, the goal is to find a shortest nonzero lattice vector $\mathbf{v} \in \mathcal{L}(\mathbf{B}) \setminus \{\mathbf{0}\}$ of norm at most $\|\mathbf{v}\| \leq \gamma \cdot \lambda_1(\mathcal{L}(\mathbf{B}))$.

The SVP asks for *a* shortest nonzero vector in a lattice, but not *the* shortest nonzero vector as several short vectors can exist. The approximate SVP_γ is more difficult for a small factor γ and becomes easier for an increasing γ. An algorithm that solves SVP in polynomial time and with exponential approximation factor $2^{\mathcal{O}(n)}$ is the Lenstra, Lenstra, Lovász (LLL) algorithm [96], which was extended in works like [135, 134, 64] (see [113] for a survey). Algorithms that achieve an exact solution or approximate solutions of SVP within $\text{poly}(n)$ factors either run in $2^{\mathcal{O}(n)}$ and require exponential space [4] or in $2^{\mathcal{O}(n \log n)}$ and require only polynomial space [89]. Based on these observations Micciancio and Regev conclude that "there is no polynomial time algorithm that approximates lattice problems to within polynomial factors" [108].

The (Approximate) Closest Vector Problem, (CVP), statement from [106]. The closest vector problem (CVP) asks, given a lattice basis $\mathbf{B}$ and target vector $\mathbf{t}$, to find the lattice vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that the distance to the target $\|\mathbf{v} - \mathbf{t}\|$ is minimized. In the γ-approximate CVP_γ, for $\gamma \geq 1$, the goal is to find a lattice vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $\|\mathbf{v} - \mathbf{t}\| \leq \gamma \cdot \text{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B}))$ where $\text{dist}(\mathbf{t}, \Lambda) = \inf\{\|\mathbf{v} - \mathbf{t}\| : \mathbf{v} \in \Lambda\}$ is the distance of $\mathbf{t}$ to Λ.

The CVP is the inhomogeneous version of the SVP and can also be formulated as syndrome decoding problem for full rank lattices [106]. The NP-hardness of SVP was shown by van Emde Boas in [140] for the ℓ_∞ norm. Ajtai then proved that SVP is NP-hard for the ℓ_2 norm using randomized reductions [3] and that the corresponding decision problem is NP-complete. It was also shown in [140] that CVP is NP-hard. However, when building cryptosystems in practice only subclasses of CVP or SVP are used that are not supposed to be NP-hard (see [84, Remark 6.24.]). A comprehensive discussion of the hardness of SVP, CVP, and its variants can be found in [141, Section 2.3] and [107]. In [77] Hanrot et al. provide a survey on the history and state-of-the-art of solvers for SVP and CVP.

1.1.3 Ring-LWE problem

The Learning with Errors (LWE) problem was popularized by Regev [126] who showed that, under a quantum reduction, solving a random LWE instance is as hard as solving certain worst-case instances of certain lattice problems. The LWE problem can be seen as a generalization of the learning parity with noise (LPN) problem [29] and is related to hard decoding problems [108]. In general, to solve the LWE problem, one has to recover a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ when given a sequence of approximate random linear equations on $\mathbf{s}$. Non-quantum reductions from variants of the shortest vector problem to variants of the LWE problem have also been shown [118]. The LWE problem is usually used to build primitives such as CPA or CCA-secure public-key encryption, identity-based encryption (IBE), or fully-homomorphic encryption schemes [128]. It can be defined as a search problem (sLWE) where the task is to recover the secret vector $\mathbf{s}$ or as a decision problem (dLWE) that asks to distinguish LWE samples from uniformly random samples.

The Learning With Errors Problem [126], (sLWE), search version. The learning with errors problem, search version, $\text{sLWE}_{n,m,q,\chi}$, with n unknowns, $m \geq n$ samples, modulo q and with error distribution χ is as follows: for a random secret s uniformly chosen in $\mathbb{Z}_q^n$, and given m samples of the form $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e \mod q)$ where $e \xleftarrow{\$} \chi$ and $\mathbf{a}$ is uniform in $\mathbb{Z}_q^n$, recover the secret vector $\mathbf{s}$.

The Learning With Errors Problem [126], (dLWE), decisional version. The learning with errors problem, decisional version, $\text{dLWE}_{n,m,q,\chi}$, with n unknowns, $m \geq n$ samples, modulo q and with error distribution χ is as follows: for a random secret $\mathbf{s}$ uniformly chosen in $\mathbb{Z}_q^n$, and given m samples either all of the form $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e \mod q)$ where $e \xleftarrow{\$} \chi$, or from the uniform distribution $(\mathbf{a}, b) \xleftarrow{\$} \mathcal{U}(\mathbb{Z}^n \times \mathbb{Z}_q)$, decide if the samples come from the former or the latter case.

An interesting property of the LWE problem is the equivalence of the (search) sLWE problem and the (decisional) dLWE problem. While it is clear that a solver for the sLWE problem can be used to solve the dLWE problem, it is also possible to solve the sLWE problem if the dLWE problem can be solved.

Theorem 1.1 (Decision to Search Reduction for LWE) *For any integers n and m, any prime $q \leq poly(n)$, and any distribution χ over $\mathbb{Z}_q$, if there exists a PPT algorithm that solves $\text{dLWE}_{n,m,q,\chi}$ with non-negligible probability, then there exists a PPT algorithm that solves $\text{sLWE}_{n,m',q,\chi}$ for some $m' = m \cdot poly(n)$ with non-negligible probability.*

Variants of the LWE problem relying on the ring of integer of a number field (or polynomial rings) were later defined and studied [138, 103]. More specifically, the lattices underlying this problem are module lattices, as in NTRU [83, 137], and its hardness can be related to the worst case hardness of finding short vectors in ideal lattices [138, 103]. The Ring-LWE problem may be defined over the ring of integers of an arbitrary number-field [104, 121]. The general definition is rather intricate involving the so-called co-different ideal $R^\vee$. For simplicity we restrict our definition to the case of cyclotomic number field with a power-of-two conductor.

The Ring Learning With Errors Problem [126], dRLWE, decisional version Let R denote the ring $Z[X]/(X^n + 1)$ for n a power of 2, and R_q the residue ring R/qR. The ring learning with errors problem, decisional version, $\text{dRLWE}_{m,q,\chi}$, with m unknowns, $m \geq 1$ samples, modulo q and with error distribution χ is as follows: for a uniform random secret $\mathbf{s} \xleftarrow{\$} \mathcal{U}(R_q)$, and given m samples either all of the form $(\mathbf{a}, \mathbf{b} = \mathbf{a} \cdot \mathbf{s} + \mathbf{e} \mod q)$ where the coefficients of $\mathbf{e}$ are independently sampled following the distribution χ, or from the uniform distribution $(\mathbf{a}, \mathbf{b}) \xleftarrow{\$} \mathcal{U}(R_q \times R_q)$, decide if the samples come from the former or the latter case.

We will in fact rely on a variant of the above problem, where the secret $\mathbf{s}$ follows the same distribution χ^n as the error $\mathbf{e}$. These variants can be proven to be equivalent to the original problem by putting the system in systematic form, as done in [13].

1.2 Algorithm description

The NewHope cryptosystem is a suite of key encapsulation mechanisms (KEM) denoted as NewHope-CPA-KEM and NewHope-CCA-KEM that are based on the conjectured quantum hardness of the RLWE

problem. Both schemes are based on a variant of the previously proposed NewHope-Simple [8] scheme modeled as semantically secure public-key encryption (PKE) scheme with respect to adaptive chosen plaintext attacks (CPA) that we refer to as NewHope-CPA-PKE. However, in this submission NewHope-CPA-PKE is only used inside of NewHope-CPA-KEM and NewHope-CCA-KEM and not intended to be an independent CPA-secure PKE scheme, in part because it does not accept arbitrary length messages. For our proposed NewHope-CPA-KEM we provide a transformation of NewHope-CPA-PKE into a passively secure KEM. For NewHope-CCA-KEM we show how to realize a semantically secure key encapsulation with respect to adaptive chosen ciphertext attacks (CCA) based on NewHope-CPA-PKE. In this section we only provide a functional description of the algorithms and refer to Section 1.3 for more background on our design decisions. Note that some algorithms, especially the ones dealing with encoding and decoding, are optimized for dimensions $n = 512$ or $n = 1024$ that we are supporting in NewHope. Moreover, the algorithms are designed for $q = 12289$ and $k = 8$ (parameter of the noise distribution). Further information on the supported parameters can be found in Section 1.4.

1.2.1 IND-CPA-secure public key encryption scheme

For our intermediate building block, the passively secure PKE scheme NewHope-CPA-PKE with a fixed message space of 256 bits, we define key generation in Algorithm 1 (NewHope-CPA-PKE.Gen), encryption in Algorithm 2 (NewHope-CPA-PKE.Encrypt), and decryption in Algorithm 3 (NewHope-CPA-PKE.Decrypt). All sub-functions used in NewHope-CPA-PKE are described in this section. Note that we assume in every function implicit access to the global parameters n, q, γ that are determined by the chosen parameter set.

Sampling, randomness, byte arrays and SHAKE. Besides polynomials in $\mathcal{R}_q$ and vectors, the main other data structure we use are byte arrays. As an example, all randomness is sampled as byte arrays. In key generation, $seed \xleftarrow{\$} \{0, \ldots, 255\}^{32}$ denotes the sampling of a byte array with 32 uniform integer elements in the range 0 to 255 from a random number generator. This random number generator shall be unpredictable and should thus be using a physical source of entropy or other means.

As strong hash function we use SHAKE256 as specified in [112]. The $\mathsf{SHAKE256}(l, d)$ function takes as input an integer l that specifies the number of output bytes and an input data byte array d. The amount of data to be absorbed is the length of d. As an example, in key generation we use SHAKE256 to compute $v \leftarrow \mathsf{SHAKE256}(64, seed)$ where we hash a 32 byte random seed denoted as $seed$ and output a byte array v with 64 elements in the range $\{0, \ldots, 255\}$.

To access byte arrays we use the bracket notation where $v[i]$ for a positive integer i denotes the i-th byte in the array v. To access ranges of bytes we use the notation $x \leftarrow v[i : j]$ for positive integers $i \leq j$ where x is assigned byte i to j of v. By $r \leftarrow \{0, \ldots, 255\}^x$ we declare that r is a byte array of length x. Using a similar notation, by $\mathbf{r} \leftarrow \mathcal{R}_q$ we declare that a variable $\mathbf{r}$ is a polynomial in $\mathcal{R}_q$ where all coefficients are zero. For bit-operations we use the operators $\gg$, $\ll$, $|$, and $\&$ as in the context of the C programming language. Thus $x \gg i$ for positive integers i, x denotes a right-shift by i. The same operator can be applied to a byte of a byte array so that $y[j] \gg i$ for positive integer i, j represents a right shift by i of the j-th byte of the byte array y. A left shift is denoted as $x \ll i$ for positive integers i, y and implicit modular reduction modulo 2^{32} is assumed (equal to writing $(x \ll i) \bmod 2^{32}$). When the left shift operator is applied to the j-th byte of the byte array y as $y[j] \ll i$ for positive integer i, j an implicit reduction modulo 2^8 is assumed (equal to writing $(y[j] \ll i) \bmod 2^8$). The $a \,|\, b$ operator denotes a bit-wise 'or' while the $a \,\&\, b$ operator denotes a bitwise 'and' of two positive integers a, b or of two bytes in a byte array. To convert a byte $a[i]$ in a byte array a to a positive integer z we use $z = \mathsf{b2i}(a[i])$. To denote positive integers in hexadecimal representation we use the prefix $0x$ such that $\texttt{0x01010101} = 16843009$. To compute the Hamming weight, the sum of all bits that are set to one in binary notation, of a byte or integer b we write $\mathsf{HW}(b)$.

Note that NewHope-CPA-PKE.Encrypt does not directly access a random number generator as all pseudo-random data is derived by expansion of a 32-byte user supplied seed $coin \in \{0, \ldots, 255\}^{32}$ that has to be obtained from a true random value generator. This is required to allow the straightforward use of NewHope-CPA-PKE.Encrypt in standard CCA transformations. Decryption is deterministic and does not need random values. For the distribution of the RLWE secret and error we use the centered binomial distribution ψ_k of parameter $k = 8$. In general, one may sample from ψ_k for integer $k > 0$ by computing $\sum_{i=0}^{k-1} b_i - b'_i$, where the $b_i, b'_i \in \{0, 1\}$ are uniform independent bits. The distribution ψ_k is centered (its mean

Algorithm 1 NewHope-CPA-PKE Key Generation

function NewHope-CPA-PKE.Gen()
  $seed \xleftarrow{\$} \{0,\ldots,255\}^{32}$
  $z \leftarrow \mathsf{SHAKE256}(64, seed)$
  $publicseed \leftarrow z[0{:}31]$
  $noiseseed \leftarrow z[32{:}63]$
  $\hat{\mathbf{a}} \leftarrow \mathsf{GenA}(publicseed)$
  $\mathbf{s} \leftarrow \mathsf{PolyBitRev}(\mathsf{Sample}(noiseseed, 0))$
  $\hat{\mathbf{s}} \leftarrow \mathsf{NTT}(\mathbf{s})$
  $\mathbf{e} \leftarrow \mathsf{PolyBitRev}(\mathsf{Sample}(noiseseed, 1))$
  $\hat{\mathbf{e}} \leftarrow \mathsf{NTT}(\mathbf{e})$
  $\hat{\mathbf{b}} \leftarrow \hat{\mathbf{a}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
  return $(pk = \mathsf{EncodePK}(\hat{\mathbf{b}}, publicseed), sk = \mathsf{EncodePolynomial}(\hat{\mathbf{s}}))$

Algorithm 2 NewHope-CPA-PKE Encryption

function NewHope-CPA-PKE.Encrypt($pk \in \{0,\ldots,255\}^{7\cdot n/4+32}$, $\mu \in \{0,\ldots,255\}^{32}$, $coin \in \{0,\ldots,255\}^{32}$)
  $(\hat{\mathbf{b}}, publicseed) \leftarrow \mathsf{DecodePk}(pk)$
  $\hat{\mathbf{a}} \leftarrow \mathsf{GenA}(publicseed)$
  $\mathbf{s}' \leftarrow \mathsf{PolyBitRev}(\mathsf{Sample}(coin, 0))$
  $\mathbf{e}' \leftarrow \mathsf{PolyBitRev}(\mathsf{Sample}(coin, 1))$
  $\mathbf{e}'' \leftarrow \mathsf{Sample}(coin, 2)$
  $\hat{\mathbf{t}} \leftarrow \mathsf{NTT}(\mathbf{s}')$
  $\hat{\mathbf{u}} \leftarrow \hat{\mathbf{a}} \circ \hat{\mathbf{t}} + \mathsf{NTT}(\mathbf{e}')$
  $\mathbf{v} \leftarrow \mathsf{Encode}(\mu)$
  $\mathbf{v}' \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{b}} \circ \hat{\mathbf{t}}) + \mathbf{e}'' + \mathbf{v}$
  $h \leftarrow \mathsf{Compress}(\mathbf{v}')$
  return $c = \mathsf{EncodeC}(\hat{\mathbf{u}}, h)$

Algorithm 3 NewHope-CPA-PKE Decryption

function NewHope-CPA-PKE.Decrypt($c \in \{0,\ldots,255\}^{7\frac{n}{4}+3\frac{n}{8}}, sk \in \{0,\ldots,255\}^{7\cdot n/4}$)
  $(\hat{\mathbf{u}}, h) \leftarrow \mathsf{DecodeC}(c)$
  $\hat{\mathbf{s}} \leftarrow \mathsf{DecodePolynomial}(sk)$
  $\mathbf{v}' \leftarrow \mathsf{Decompress}(h)$
  $\mu \leftarrow \mathsf{Decode}(\mathbf{v}' - \mathsf{NTT}^{-1}(\hat{\mathbf{u}} \circ \hat{\mathbf{s}}))$
  return μ

is 0), has variance $k/2$ and we set $k = 8$ in all instantiations. This gives a standard deviation of $\varsigma = \sqrt{8/2}$. We describe sampling from ψ_8 in Algorithm 4 as the function Sample that takes as input a 32 byte seed *seed* and an integer parameter $0 \leq nonce < 2^8$ for domain separation. This way one seed can be used to sample multiple polynomials. The output is a polynomial $\mathbf{r} \in \mathcal{R}_q$ where all n coefficients are independently distributed according to ψ_8.

Polynomials and the NTT. The main mathematical objects that are manipulated in NewHope are polynomials in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ like $\mathbf{s}, \mathbf{e}, \hat{\mathbf{s}}, \hat{\mathbf{a}}, \hat{\mathbf{b}}, \mathbf{s}', \mathbf{e}', \mathbf{e}'', \hat{\mathbf{t}}, \hat{\mathbf{u}}, \hat{\mathbf{e}}, \mathbf{v}, \mathbf{v}'$. For a polynomial $\mathbf{c} \in \mathcal{R}_q$ where $\mathbf{c} = \sum_{i=0}^{n-1} c_i X^i$ we denote by c_i the i-th coefficient of $\mathbf{c}$ for integer $i \in \{0,\ldots,n-1\}$. We use the same notation to access elements of vectors that are not necessary in $\mathcal{R}_q$. Addition or subtraction of polynomials in $\mathcal{R}_q$ (denoted as $+$ or $-$, respectively) is the usual coefficient-wise addition or subtraction,

Algorithm 4 Deterministic sampling of polynomials in $\mathcal{R}_q$ from ψ_8^n

function SAMPLE($seed \in \{0,\ldots,255\}^{32}$, positive integer $nonce$)
  $\mathbf{r} \leftarrow \mathcal{R}_q$
  $extseed \leftarrow \{0,\ldots,255\}^{34}$
  $extseed[0{:}31] \leftarrow seed[0{:}31]$
  $extseed[32] \leftarrow nonce$
  for i **from** 0 **to** $(n/64)-1$ **do**
    $extseed[33] \leftarrow i$
    $buf \leftarrow \mathsf{SHAKE256}(128, extseed)$
    for j **from** 0 **to** 63 **do**
      $a \leftarrow buf[2*j]$
      $b \leftarrow buf[2*j+1]$
      $r_{64*i+j} = \mathsf{HW}(a) + q - \mathsf{HW}(b) \bmod q$
  return $\mathbf{r} \in \mathcal{R}_q$

such that for $\mathbf{a} = \sum_{i=0}^{n-1} a_i X^i \in \mathcal{R}_q$ and $\mathbf{b} = \sum_{i=0}^{n-1} b_i X^i \in \mathcal{R}_q$ we get $\mathbf{a}+\mathbf{b} = \sum_{i=0}^{n-1}(a_i + b_i \mod q)X^i$ and $\mathbf{a}-\mathbf{b} = \sum_{i=0}^{n-1}(a_i - b_i \mod q)X^i$. In general, fast quasi-logarithmic algorithms exist for polynomial multiplication. We explicitly specify how to use the Number Theoretic Transform (NTT); some polynomials are also transmitted in a transformed representation. However, an implementer may choose a different algorithm for polynomial multiplication, like Karatsuba or Schoolbook multiplication, and then transform the result into the NTT domain such that it is compliant with this specification. Moreover, here we just describe the basic definition and refer to [76, 2] and Section 2 for details on the efficient implementation of the NTT.

With the NTT, a polynomial multiplication for elements in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$ can be performed by computing $\mathbf{c} = \mathsf{NTT}^{-1}(\mathsf{NTT}(\mathbf{a}) \circ \mathsf{NTT}(\mathbf{b}))$ for $\mathbf{a},\mathbf{b},\mathbf{c} \in \mathcal{R}_q$. The $\circ$ operator denotes coefficient-wise multiplication of two polynomials $\mathbf{a},\mathbf{b} \in \mathcal{R}_q$ such that $\mathbf{a} \circ \mathbf{b} = \sum_{i=0}^{n-1}(a_i \cdot b_i \mod q)X^i$. The NTT defined in $\mathcal{R}_q$ can be implemented very efficiently if n is a power of two and q is a prime for which it holds that $q \equiv 1 \mod 2n$. This way a primitive n-th root of unity ω and its square root $\gamma = \sqrt{\omega} \bmod q$ exist. By multiplying coefficient-wise by powers of γ before the NTT computation and after the reverse transformation by powers of $\gamma^{-1} \bmod q$, no zero padding is required and an n-point NTT can be used to transform a polynomial with n coefficients.

For a polynomial $\mathbf{g} = \sum_{i=0}^{n-1} g_i X^i \in \mathcal{R}_q$ we define

$$\mathsf{NTT}(\mathbf{g}) = \hat{\mathbf{g}} = \sum_{i=0}^{n-1} \hat{g}_i X^i, \text{ with}$$
$$\hat{g}_i = \sum_{j=0}^{n-1} \gamma^j g_j \omega^{ij} \bmod q,$$

where ω is an n-th primitive root of unity and $\gamma = \sqrt{\omega} \bmod q$.

Note that most implementations will use an in-place NTT algorithm which usually requires bit-reversal operations that are not included in the previously given straightforward description of the NTT. As an optimization, we allow implementations to skip these bit-reversals for the forward transformation as all inputs are only random noise. Thus, and slightly counter-intuitive, we define bit-reversal and perform it on polynomials that go into the NTT. With bit-reversal and our straightforward NTT definition, implementers do not need to apply a reversal when using an in-place NTT. Note that in key generation this optimization is transparent to the protocol, but due to the re-encryption implementers have to follow our instructions. For a positive integer v and a power of two n we formally define bit-reversal as $\mathsf{BitRev}(v) = \sum_{i=0}^{\log_2(n)-1}(((v \gg i)\&1) \ll (\log_2(n)-1-i))$. For polynomials $\mathbf{s},\mathbf{z} \in \mathcal{R}_q$ the bit-reversal of a polynomial $\mathbf{s}$ is $\mathbf{z} = \mathsf{PolyBitRev}(\mathbf{s}) = \sum_{i=0}^{n-1} s_i X^{\mathsf{BitRev}(i)}$.

The function NTT^{-1} is the inverse of the function NTT. The computation of NTT^{-1} is essentially the same as the computation of NTT, except that it uses $\omega^{-1} \mod q$, multiplies by powers of $\gamma^{-1} \mod q$ after

the summation, and also multiplies each coefficient by the scalar $n^{-1} \mod q$ so that

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

Note that we define the $x \mod q$ operation for integers x, q to always produce an output in the range $[0, q-1]$. Unless otherwise stated, when we access an element a_i of a polynomial $\mathbf{a} \in \mathcal{R}_q$ we always assume that a_i is reduced modulo q and in the range $[0, q-1]$.

Definition of GenA. The public parameter $\mathbf{a}$ is generated by GenA which takes as input a 32 byte array *seed*. The function is described in Algorithm 5. The resulting polynomial $\mathbf{a}$ (denoted as $\hat{\mathbf{a}}$) is considered to be in the NTT domain. This is possible because the NTT transforms uniform polynomials to uniform polynomials. Inside of GenA we use the SHAKE128 hash function [112] to expand the pseudorandom seed and we define a function that absorbs a byte array into the internal state of SHAKE128 and then we use another function to obtain pseudorandom data by squeezing the internal state. The $state \leftarrow \mathsf{SHAKE128Absorb}(d)$ functions takes as input a byte array d. It outputs a byte array of length 200 that represents the state after absorbing d. To obtain pseudorandom values $buf, state \leftarrow \mathsf{SHAKE128Squeeze}(j, state)$ is used. As input the function takes a positive integer j determining the amount of output blocks of SHAKE128 to be produced and the 200 byte state. It outputs a byte array buf of length $168 \cdot j$ and a byte array *state* of length 200.

Algorithm 5 Deterministic generation of $\hat{\mathbf{a}}$ by expansion of a seed

function GenA($seed \in \{0, \ldots, 255\}^{32}$)
  $\hat{\mathbf{a}} \leftarrow \mathcal{R}_q$
  $extseed \leftarrow \{0, \ldots, 255\}^{33}$
  $extseed[0:31] \leftarrow seed[0{:}31]$
  for i **from** 0 **to** $(n/64) - 1$ **do**
    $ctr \leftarrow 0$
    $extseed[32] \leftarrow i$
    $state \leftarrow \mathsf{SHAKE128Absorb}(extseed)$
    while $ctr < 64$ **do**
      $buf, state \leftarrow \mathsf{SHAKE128Squeeze}(1, state)$
      $j \leftarrow 0$
      for $j < 168$ **and** $ctr < 64$ **do**
        $val \leftarrow \mathsf{b2i}(buf[j]) \,|\, (\mathsf{b2i}(buf[j+1]) \ll 8)$
        if $val < 5 \cdot q$ **then**
          $\hat{a}_{i*64+ctr} \leftarrow val$
          $ctr \leftarrow ctr + 1$
        $j \leftarrow j + 2$
  return $\hat{\mathbf{a}} \in \mathcal{R}_q$

Encoding and decoding of the secret and public key. Note that polynomials are transmitted in the NTT domain and thus for interoperability our definition and parametrization of the NTT has to be used.

To encode a polynomial in $\mathcal{R}_q$ into an array of bytes we use EncodePolynomial as described in Algorithm 6. The function DecodePolynomial as described in Algorithm 7 converts a byte array into an element in $\mathcal{R}_q$. The secret key consists only of one polynomial $\mathbf{s} \in \mathcal{R}_q$ and thus we can directly apply $\mathsf{EncodePolynomial}(\hat{\mathbf{s}})$. The secret key is then either encoded into an array of 869 bytes ($n = 512$) or 1792 bytes ($n = 1024$). The public key is encoded as an array of 928 bytes ($n = 512$) or 1824 bytes ($n = 1024$) by $\mathsf{EncodePK}(\hat{\mathbf{b}}, seed)$ described in Algorithm 8. It takes as input a polynomial $\hat{\mathbf{b}} \in \mathcal{R}_q$ and a byte array *seed* with 32 elements. The $\mathsf{DecodePk}(pk)$ function decodes the public key and is provided in Algorithm 9.

Algorithm 6 Encoding of a polynomial in $\mathcal{R}_q$ to a byte array

function EncodePolynomial($\hat{\mathbf{s}}$)
  $r \leftarrow \{0,\ldots,255\}^{7\cdot n/4}$
  for i **from** 0 **to** $n/4-1$ **do**
    $t0 \leftarrow \hat{s}_{4*i+0} \mod q$
    $t1 \leftarrow \hat{s}_{4*i+1} \mod q$
    $t2 \leftarrow \hat{s}_{4*i+2} \mod q$
    $t3 \leftarrow \hat{s}_{4*i+3} \mod q$
    $r[7*i+0] \leftarrow t0\&\texttt{0xff}$
    $r[7*i+1] \leftarrow (t0 \gg 8)\,|(t1 \ll 6)\&\texttt{0xff}$
    $r[7*i+2] \leftarrow (t1 \gg 2)\&\texttt{0xff}$
    $r[7*i+3] \leftarrow (t1 \gg 10)\,|(t2 \ll 4)\&\texttt{0xff}$
    $r[7*i+4] \leftarrow (t2 \gg 4)\&\texttt{0xff}$
    $r[7*i+5] \leftarrow (t2 \gg 12)\,|(t3 \ll 2)\&\texttt{0xff}$
    $r[7*i+6] \leftarrow (t3 \gg 6)\&\texttt{0xff}$
  return $r \in \{0,\ldots,255\}^{7\cdot n/4}$

Algorithm 7 Decoding of a polynomial represented as a byte array into an element in $\mathcal{R}_q$

function DecodePolynomial($v \in \{0,\ldots,255\}^{7\cdot n/4}$)
  for i **from** 0 **to** $n/4-1$ **do**
    $r \leftarrow \mathcal{R}_q$
    $r_{4*i+0} \leftarrow \mathsf{b2i}(v[7*i+0])\|((\mathsf{b2i}(v[7*i+1])\,\&\,\texttt{0x3f}) \ll 8)$
    $r_{4*i+1} \leftarrow (\mathsf{b2i}(v[7*i+1]) \gg 6)\|(\mathsf{b2i}(v[7*i+2]) \ll 2)\|((\mathsf{b2i}(v[7*i+3])\,\&\,\texttt{0x0f}) \ll 10)$
    $r_{4*i+2} \leftarrow (\mathsf{b2i}(v[7*i+3]) \gg 4)\|(\mathsf{b2i}(v[7*i+4]) \ll 4)\|((\mathsf{b2i}(v[7*i+5])\,\&\,\texttt{0x03}) \ll 12)$
    $r_{4*i+3} \leftarrow (\mathsf{b2i}(v[7*i+5]) \gg 2)\|(\mathsf{b2i}(v[7*i+6]) \ll 6)$
  return $\mathbf{r} \in \mathcal{R}_q$

Algorithm 8 Encoding of the public key

function EncodePK($\hat{\mathbf{b}} \in \mathcal{R}_q, publicseed \in \{0,\ldots,255\}^{32}$)
  $r \leftarrow \{0,\ldots,255\}^{7\cdot n/4+32}$
  $r[0:7\cdot n/4-1] \leftarrow \mathsf{EncodePolynomial}(\hat{\mathbf{b}})$
  $r[7\cdot n/4 : 7\cdot n/4+31] \leftarrow publicseed[0:31]$
  return $r \in \{0,\ldots,255\}^{7\cdot n/4+32}$

Algorithm 9 Decoding of the public key

function DecodePk($pk \in \{0,\ldots,255\}^{7\cdot n/4+32}$)
  $\hat{\mathbf{b}} \leftarrow \mathsf{DecodePolynomial}(pk[0:7\cdot n/4-1])$
  $seed \leftarrow pk[7\cdot n/4 : 7\cdot n/4+31])$
  return $(\hat{\mathbf{b}} \in \mathcal{R}_q, seed \in \{0,\ldots,255\}^{32})$

Encoding and decoding of the ciphertext. The ciphertext encoding is described in Algorithm 13. The ciphertext c is encoded as an array of 1088 bytes ($n = 512$) or 2176 bytes ($n = 1024$) by $\mathsf{EncodeC}(\hat{\mathbf{u}}, h)$ that takes as input a polynomial in $\hat{\mathbf{u}} \in \mathcal{R}_q$ and an array h of $3 \cdot n/8$ bytes that was generated by $\mathsf{Compress}(\mathbf{v}')$ as given in Algorithm 12. The compression and decompression functions simply perform coefficient-wise modulus switching between modulus q and modulus 8 by multiplying by the new modulus and then performing a rounding division by the old modulus. To decode the ciphertext the $\mathsf{DecodeC}$ function is used that outputs $\hat{\mathbf{u}} \in \mathcal{R}_q$ and a byte array h that is then given to $\mathsf{Decompress}$ to obtain $\mathbf{v}' \in \mathcal{R}_q$.

In NewHope-CPA-PKE the 256-bit message μ represented as an array of 32 bytes has to be encoded into an element in $\mathcal{R}_q$ during encryption and decoded from an element in $\mathcal{R}_q$ into a byte array during decryption. To allow robustness against errors each bit of the 256-bit message $\mu \in \{0, \ldots, 255\}^{32}$ is encoded into $\lfloor n/256 \rfloor$ coefficients by Encode (see Algorithm 10). The decoding function Decode (see Algorithm 11) maps from $\lfloor n/256 \rfloor$ coefficients back to the original key bit. For example, for $n = 1024$, take $4 = \lfloor 1024/256 \rfloor$ coefficients (each in the range $\{0, \ldots, q-1\}$, subtract $\lfloor q/2 \rfloor$ from each of them, accumulate their absolute values, and set the key bit to 0 if the sum is larger than q or to 1 otherwise.

Algorithm 10 Message encoding

```
1:  function ENCODE(μ ∈ {0,...,255}^32)
2:      v ← R_q
3:      for i from 0 to 31 do
4:          for j from 0 to 7 do
5:              mask ← −((msg[i] ≫ j) & 1)
6:              v_{8*i+j+0} ← mask &(q/2)
7:              v_{8*i+j+256} ← mask &(q/2)
8:              if n equals 1024 then
9:                  v_{8*i+j+512} ← mask &(q/2)
10:                 v_{8*i+j+768} ← mask &(q/2)
11:     return v ∈ R_q
```

Algorithm 11 Message decoding

```
1:  function DECODE(v ∈ R_q)
2:      μ ← {0,...,255}^32
3:      for i from 0 to 255 do
4:          t ← |(v_{i+0}  mod q) − (q − 1)/2|
5:          t ← t + |(v_{i+256}  mod q) − (q − 1)/2|
6:          if n equals 1024 then
7:              t ← t + |(v_{i+512}  mod q) − (q − 1)/2|
8:              t ← t + |(v_{i+768}  mod q) − (q − 1)/2|
9:              t ← ((t − q))
10:         else
11:             t ← ((t − q/2))
12:         t ← t ≫ 15
13:         μ[i ≫ 3] ← μ[i ≫ 3] |(t ≪ (i & 7))
14:     return μ ∈ {0,...,255}^32
```

1.2.2 Interconversion to IND-CPA KEM

NewHope-CPA-PKE can be converted to an IND-CPA-secure key encapsulation mechanism by using the public key encryption scheme to convey a secret, K. The PKE's coins and the secret K are computed by

Algorithm 12 Ciphertext compression

function COMPRESS($\mathbf{v}' \in \mathcal{R}_q$)
  $k \leftarrow 0$
  $t \leftarrow \{0,\ldots,255\}^8$
  $h \leftarrow \{0,\ldots,255\}^{3\cdot n/8}$
  for ℓ **from** 0 **to** $n/8-1$ **do**
    $i \leftarrow 8\cdot\ell$
    for j **from** 0 **to** 7 **do**
      $t[j] \leftarrow v'_{i+j} \mod q$
      $t[j] \leftarrow ((\mathsf{b2i}(t[j] \ll 3) + q/2)/q)\,\&\,\mathtt{0x7}$
    $h[k+0] \leftarrow t[0]\,|(t[1] \ll 3)\,|(t[2] \ll 6)$
    $h[k+1] \leftarrow (t[2] \gg 2)\,|(t[3] \ll 1)\,|(t[4] \ll 4)\,|(t[5] \ll 7)$
    $h[k+2] \leftarrow (t[5] \gg 1)\,|(t[6] \ll 2)\,|(t[7] \ll 5)$
    $k+ \leftarrow 3$
  return $r\ in\{0,\ldots,255\}^{3\cdot n/8}$

Algorithm 13 Ciphertext encoding

function ENCODEC($\hat{\mathbf{u}} \in \mathcal{R}_q, h \in \{0,\ldots,255\}^{3\cdot n/8}$)
  $c[0:(7\cdot n/4-1)] \leftarrow \mathsf{EncodePolynomial}(\hat{\mathbf{u}})$
  $c[(7\cdot n/4):(7\cdot n/4+3\cdot n/8-1)] \leftarrow h$
  return $c \in \{0,\ldots,255\}^{7\cdot n/4+3\cdot n/8}$

Algorithm 14 Ciphertext decoding

function DECODEC($c \in \{0,\ldots,255\}^{7\cdot n/4+3\cdot n/8}$)
  $\hat{\mathbf{u}} \leftarrow \mathsf{DecodePolynomial}(c[0:(7\cdot n/4-1)])$
  $h \leftarrow c[(7\cdot n/4):(7\cdot n/4+3\cdot n/8-1)]$
  return $(\hat{\mathbf{u}} \in \mathcal{R}_q, h \in \{0,\ldots,255\}^{3\cdot n/8})$

Algorithm 15 Ciphertext decompression

function DECOMPRESS($h \in \{0,\ldots,255\}^{3\cdot n/8}$)
  $k \leftarrow 0$
  for ℓ **from** 0 **to** $n/8-1$ **do**
    $i \leftarrow 8\cdot\ell$
    $r_{i+0} \leftarrow a[k+0]\,\&\,7$
    $r_{i+1} \leftarrow (a[k+0] \gg 3)\,\&\,7$
    $r_{i+2} \leftarrow (a[k+0] \gg 6)\,|((a[1] \ll 2)\,\&\,4)$
    $r_{i+3} \leftarrow (a[k+1] \gg 1)\,\&\,7$
    $r_{i+4} \leftarrow (a[k+1] \gg 4)\,\&\,7$
    $r_{i+5} \leftarrow (a[k+1] \gg 7)\,|((a[2] \ll 1)\,\&\,6)$
    $r_{i+6} \leftarrow (a[k+2] \gg 2)\,\&\,7$
    $r_{i+7} \leftarrow (a[k+2] \gg 5)$
    $k \leftarrow k+3$
    for j **from** 0 **to** 7 **do**
      $r_{i+j} \leftarrow (r_{i+j} * q + 4) \gg 3$
  return $\mathbf{v}' \in \mathcal{R}_q$

hashing random coins, rather than using random coins directly, to protect against attacks involving disclosure of system randomness. The final shared secret is derived from the secret K by hashing. The resulting algorithms for NewHope-CPA-KEM are shown in Algorithms 16, 17, and 18.

Algorithm 16 NewHope-CPA-KEM Key Generation

function NewHope-CPA-KEM.Gen()
  $(pk, sk) \xleftarrow{\$}$ NewHope-CPA-PKE.Gen()
  return (pk, sk)

Algorithm 17 NewHope-CPA-KEM Encapsulation

function NewHope-CPA-KEM.Encaps(pk)
  $coin \xleftarrow{\$} \{0,\ldots,255\}^{32}$
  $K \| coin' \leftarrow \mathsf{SHAKE256}(64, coin) \in \{0,\ldots,255\}^{32+32}$
  $c \leftarrow$ NewHope-CPA-PKE.Encrypt$(pk, K; coin')$
  $ss \leftarrow \mathsf{SHAKE256}(32, K)$
  return (c, ss)

Algorithm 18 NewHope-CPA-KEM Decapsulation

function NewHope-CPA-KEM.Decaps(c, sk)
  $K' \leftarrow$ NewHope-CPA-PKE.Decrypt(c, sk)
  return $ss = \mathsf{SHAKE256}(32, K')$

1.2.3 Transform from IND-CPA PKE to IND-CCA KEM[1]

The Fujisaki–Okamoto transform [59] constructs an IND-CCA2-secure public-key encryption scheme from a one-way-secure public key encryption scheme in the classical random oracle model (with an assumption on the distribution of ciphertexts for each plaintext being sufficiently close to uniform). Targhi and Unruh [139] gave a variant of the Fujisaki–Okamoto transform and showed its IND-CCA2 security against a quantum adversary in the quantum random oracle model under similar assumptions. The results of both FO and TU proceed under the assumption that the public key encryption scheme has perfect correctness, which is not the case for lattice-based schemes. Hofheinz, Hövelmanns, and Kiltz [85] gave a variety of constructions in a modular fashion. We apply their $\mathsf{QFO}^{\not\perp}_m$ transform which constructs an IND-CCA-secure key encapsulation mechanism from an IND-CPA public key encryption scheme and three hash functions; following [32], we make the following modifications, denoting the resulting transform $\mathsf{QFO}^{\not\perp\prime}_m$:

- A single hash function (with longer output) is used to compute K, $coin'$, and d.
- The computation of K, $coin'$, and d also takes the public key pk as input.
- The computation of the shared secret ss also takes the encapsulation c and d as input.

$\mathsf{QFO}^{\not\perp\prime}_m$ transform Let $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a public key encryption scheme with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$, where the randomness space of $\mathsf{Encrypt}$ is $\mathcal{R}^E$. Let $\mathsf{len}_s, \mathsf{len}_K, \mathsf{len}_d, \mathsf{len}_{ss}$ be parameters. Let $G : \{0,\ldots,255\}^* \to \{0,\ldots,255\}^{\mathsf{len}_K} \times \mathcal{R}^E \times \{0,\ldots,255\}^{\mathsf{len}_d}$ and $F : \{0,\ldots,255\}^* \to \{0,\ldots,255\}^{\mathsf{len}_{ss}}$ be hash functions. Define $\mathsf{QKEM}^{\not\perp\prime}_m = \mathsf{QFO}^{\not\perp\prime}_m[\mathsf{PKE}, G, F]$ be the key encapsulation mechanism with $\mathsf{QKEM}^{\not\perp\prime}_m.\mathsf{KeyGen}$, $\mathsf{QKEM}^{\not\perp\prime}_m.\mathsf{Encaps}$ and $\mathsf{QKEM}^{\not\perp\prime}_m.\mathsf{Decaps}$ as shown in Figure 1.

[1] The text in this section is shared with the FrodoKEM submission.

function $\mathsf{QKEM}_m^{\not\perp\prime}.\mathsf{KeyGen}()$
  $(pk, sk) \xleftarrow{\$} \mathsf{PKE.KeyGen}()$
  $s \xleftarrow{\$} \{0,\ldots,255\}^{\mathsf{len}_s}$
  $\overline{sk} \leftarrow (sk, pk, s)$
  return $(pk, \overline{sk})$

function $\mathsf{QKEM}_m^{\not\perp\prime}.\mathsf{Encaps}(pk)$
  $\mu \xleftarrow{\$} \mathcal{M}$
  $(K, coin', d) \leftarrow G(pk\|\mu)$
  $c \leftarrow \mathsf{PKE.Encrypt}(pk, \mu; coin')$
  $ss \leftarrow F(K\|c\|d)$
  $\overline{c} \leftarrow (c, d)$
  return $(\overline{c}, ss)$

function $\mathsf{QKEM}_m^{\not\perp\prime}.\mathsf{Decaps}((c,d),(sk,pk,s))$
  $\mu' \leftarrow \mathsf{PKE.Decrypt}(c, sk)$
  $(K', coin'', d') \leftarrow G(pk\|\mu')$
  if $c = \mathsf{PKE.Encrypt}(pk, \mu'; coin'')$ **and** $d = d'$ **then**
    return $ss' \leftarrow F(K'\|c\|d)$
  else
    return $ss' \leftarrow F(s\|c\|d)$

Figure 1: Construction of an IND-CCA-secure key encapsulation mechanism $\mathsf{QKEM}_m^{\not\perp\prime} = \mathsf{QFO}_m^{\not\perp\prime}[\mathsf{PKE}, G, F]$ from a public key encryption scheme PKE and hash functions G and F.

1.2.4 IND-CCA-secure key encapsulation mechanism

NewHope-CCA-KEM is derived from NewHope-CPA-PKE by applying the $\mathsf{QFO}_m^{\not\perp\prime}$ transformation. The hash functions G and F are both taken to be SHAKE256. The length parameters are taken as $\mathsf{len}_s = \mathsf{len}_K = \mathsf{len}_d = \mathsf{len}_{ss} = 32$. The randomness space is $\{0,\ldots,255\}^{32}$. The message space $\mathcal{M}$ is $\{0,\ldots,255\}^{32}$. The ciphertext component c to F is instead computed as $\mathsf{SHAKE256}(c)$ for efficiency (a new buffer does not need to be allocated). The KEM public key also caches a hash of the public key to save on computation. Finally, some random values ($seed, rand$ in NewHope-CCA-KEM.Gen and μ in NewHope-CCA-KEM.Encaps) are computed by hashing random coins, rather than using random coins directly, to protect against attacks involving disclosure of system randomness. The resulting algorithms for NewHope-CCA-KEM are shown in Algorithms 19, 20, and 21.

1.2.5 Interconversion to IND-CCA PKE

NewHope-CCA-KEM can be converted to an IND-CCA-secure public key encryption scheme using standard conversion techniques as specified by NIST. In particular, shared secret ss can be used as the encryption key in an appropriate data encapsulation mechanism in the KEM/DEM (key encapsulation mechanism / data encapsulation mechanism) framework [46].

1.3 Design rationale

Currently, we see two main approaches to build practical lattice-based PKEs. One is to base the schemes on NTRU or a related assumption [83, 137]. The other approach we are using in this work is the LWE/RLWE assumption [102]. However, usage of the LWE assumption comes with a cost as keys become rather large. This can be avoided by relying on ideal lattices and the Ring-Learning With Errors (RLWE) assumption. While RLWE certainly features more structure than LWE, no algorithms are known that can exploit this structure and that are thus working more efficiently on RLWE than on LWE. When restricting the design space to ideal lattices due to smaller key sizes, then the seminal work by Lyubashevsky, Peikert and Regev [103, 102] (from now on referred to as LPR10) can be considered as the core basis for follow-up work like [31, 50, 120].

The traditional approach for passively secure LWE-based (and Ring-LWE-based) key encapsulation (KEM) or key exchange is derived straight-forwardly from LWE-based (or Ring-LWE-based) encryption schemes like the ones described in [127, 99, 70]: Alice generates a key pair (sk_A, pk_A), sends pk_A to Bob; Bob chooses a (symmetric) key k, encrypts this key under pk_A and sends it to Alice; Alice decrypts to obtain k. See, for example, [118, Sec. 4.2] for an adaptation of the passively secure lattice-based cryptosystem from [69] to this KEM setting. We will in the following refer to this approach as a Key Transport Mechanism (KTM) or as the *encryption-based* approach for RLWE-based key exchange.

Algorithm 19 NewHope-CCA-KEM Key Generation

function NewHope-CCA-KEM.Gen()
  $(pk, sk) \xleftarrow{\$}$ NewHope-CPA-PKE.Gen()
  $s \xleftarrow{\$} \{0, \ldots, 255\}^{32}$
  return $(pk, \overline{sk} = sk \| pk \| \mathsf{SHAKE256}(32, pk) \| s)$

Algorithm 20 NewHope-CCA-KEM Encapsulation

function NewHope-CCA-KEM.Encaps(pk)
  $coin \xleftarrow{\$} \{0, \ldots, 255\}^{32}$
  $\mu \leftarrow \mathsf{SHAKE256}(32, coin) \in \{0, \ldots, 255\}^{32}$
  $K \| coin' \| d \leftarrow \mathsf{SHAKE256}(96, \mu \| \mathsf{SHAKE256}(32, pk)) \in \{0, \ldots, 255\}^{32+32+32}$
  $c \leftarrow$ NewHope-CPA-PKE.Encrypt$(pk, \mu; coin')$
  $ss \leftarrow \mathsf{SHAKE256}(32, K \| \mathsf{SHAKE256}(32, c \| d))$
  return $(\overline{c} = c \| d, ss)$

Algorithm 21 NewHope-CCA-KEM Decapsulation

function NewHope-CCA-KEM.Decaps($\overline{c}, \overline{sk}$)
  $c \| d \leftarrow \overline{c} \in \{0, \ldots, 255\}^{32+32}$
  $sk \| pk \| h \| s \leftarrow \overline{sk} \in \{0, \ldots, 255\}^{32+32+32+32}$
  $\mu' \leftarrow$ NewHope-CPA-PKE.Decrypt(c, sk)
  $K' \| coin'' \| d' \leftarrow \mathsf{SHAKE256}(96, \mu' \| h) \in \{0, \ldots, 255\}^{32+32+32}$
  if $c =$ NewHope-CPA-PKE.Encrypt$(pk, \mu'; coin'')$ **and** $d = d'$ **then**
    $fail \leftarrow 0$
  else
    $fail \leftarrow 1$
  $K_0 \leftarrow K'$
  $K_1 \leftarrow s$
  return $ss = \mathsf{SHAKE256}(32, K_{fail} \| \mathsf{SHAKE256}(32, c \| d))$

A slightly different approach is what we will in the following call the *reconciliation-based approach*. Instead of letting Bob choose a secret key, Alice and Bob compute a noisy shared secret value and then use some reconciliation mechanism that allows them to agree on the same shared key (often also referred to as key agreement mechanism in the literature). This idea of a reconciliation mechanism to extract an exact shared value from noisy data is essentially the idea of a fuzzy extractor [52], known, for example, from physically unclonable functions. See, for example, [34, 80].

The reconciliation-based approach for Ring-LWE-based key agreement was listed as a special instance of "Noisy Diffie Hellman" by Gaborit (presenting joint work with Aguilar, Lacharme, Schrek, and Zémor) in his talk at PQCrypto 2010 [61, Slide 6]. It was also described by Lindner and Peikert as "(approximate) key agreement" [99, Sec. 3.1] and was already mentioned vaguely in an invited lecture of Peikert at TCC 2009 [119, Slide 14]. In [48] Ding described an instantiation of a reconciliation-based approach of LWE-based key exchange. The reconciliation mechanism can be used on top of a *matrix form of LWE* (as already used earlier, for example in [68] and [99, Sec. 2.2 and 3.1]) or on top of RLWE [102]. Neither of [61, 99, 119] describe a concrete reconciliation mechanism; the first reconciliation mechanism was the one described by Ding in [48, Sec 1.3] and [50]. Note that later and revised versions of [50] also list Lin [49] and Xie and Lin [51] as authors. Peikert in [120] tweaked Ding's reconciliation mechanism to obtain unbiased keys; the approach by Ding inevitably produces slightly biased key bits.

The main reason for the reconciliation-based approach is a reduced bandwidth requirement. For exam-

ple, [120] advertises "nearly halving the ciphertext size". This estimate comes from the fact that (in a naive version of the encryption-based approach) the second ciphertext polynomial has coefficients in $\{0, \ldots, q-1\}$ whereas the coefficients are in $\{0, 1\}$ (for [50, 49, 51] and [120]) or in $\{0, 1, 2, 3\}$ (for [9]) when using the reconciliation approach.

Our proposal: NewHope. Our submission, NewHope, is based on NewHope-Simple [8] which is a variant of NewHope-Usenix [9]. The main difference is that NewHope-Simple uses the encryption-based approach while NewHope-Usenix is based on the reconciliation-based approach. Alternatively, our submission could also be described as a variant of the scheme by Lyubashevsky, Peikert and Regev [103, 102] to which we apply the modifications from NewHope-Usenix [9] and the ciphertext size reduction technique from [123].

The basic NewHope-CPA-PKE scheme is a semantically secure public-key encryption with respect to adaptive chosen plaintext attacks. This allows us to apply standard transformations to build passively and actively secure KEMs and PKEs. This enables the use of our submission in unauthenticated key-exchange protocol but also in settings where a CCA-secure KEM or PKE is required. As a consequence, in this section we mainly focus on the properties of NewHope-CPA-PKE for which we define key generation in Algorithm 1 encryption in Algorithm 2 and decryption in Algorithm 3.

Parameter choices. We fix $q = 12289$ and $k = 8$ and provide two parameter sets that differ in only one parameter. For our NewHope512 with a bit-security level of 101 we set $n = 512$ and for NewHope1024 with a bit-security level of 233 we choose $n = 1024$. However, for long-term security we recommend NewHope1024. As k is fixed the same binomial sampler can be used for implementations of both parameter sets. Due to the fact that the security level grows with the noise-to-modulus ratio, it makes sense to choose the modulus as small as possible, improving compactness and efficiency together with security. As noise parameter k of the binomial distribution $\psi_k = \sum_{i=1}^{k} b_i - b'_i$ we set $k = 8$ for both parameter sets. This way we achieve a negligible error probability for both parameter sets. We chose the modulus $q = 12289$ as it is the smallest prime for which it holds that $q \equiv 1 \mod 2n$ so that the number-theoretic transform (NTT) can be realized efficiently and that we can transfer polynomials in NTT encoding (see Section 1.2.1). The choice is also appealing as the prime is already used by some implementations of Ring-LWE encryption [132, 47, 100] and BLISS signatures [54, 122]; thus sharing of some code (or hardware modules) between our proposal and an implementation of BLISS would be possible.

Error correction and reconciliation. The reconciliation technique of NewHope-Usenix [9] is generalizing and improving the previous approaches and extracts a single key bit from multiple polynomial coefficients. It relies on non-trivial *lattice-codes* and *lattice-quantizers* [43]. It is efficient, but fairly complex. Due to the complexity of the reconciliation approach in NewHope-Usenix [9] we propose the usage of the encryption-based approach. The difference in bandwidth requirements for NewHope-Usenix and NewHope is much smaller than one might expect. Specifically, the message from Bob to Alice (the ciphertext) in NewHope-Simple requires only 2176 bytes (compared to 2048 bytes in NewHope-Usenix); the message from Alice to Bob (the public key) has the same size (1824 bytes) for both variants. We obtain this result by carefully analyzing and optimizing a technique that is known since at least [118, Sec. 4.2] and has also been used in [123], for lattice-based signatures in [75], in the context of fully homomorphic encryption in [36, Sec. 4.2] and [33, Sec. 5.4] and for lattice-based PRFs in [18] and that was also applied in works like [88]. The idea of this technique is that the low bits of each coefficient of $\mathbf{v}$ mainly carry noise and contribute very little to the successful recovery of the plaintext. One can thus decide to "discard" (i.e, not transmit) those bits and thus shorten the length of $\mathbf{v}$. This can also be seen as switching to a smaller modulus and is therefore also called "modulus switching".

We combine this technique with a simple technique to encode one key bit into 4 coefficients that was first described by Güneysu and Pöppelmann in [123] and minimize the ciphertext size under the constraint that the failure probability of NewHope-Simple does not exceed the failure probability of NewHope.

Noise distribution. We do not use discrete Gaussians as the noise distribution but instead use the centred binomial distribution ψ_k of parameter $k = 8$ for the secret and error term. The reason is that it turns out to be challenging to implement a discrete Gaussian sampler efficiently *and* protected against timing attacks (see [31, 9]). On the other hand, sampling from the centered binomial distribution is easy and does not require high-precision computations or large tables as one may sample from ψ_k by computing $\sum_{i=0}^{k-1} b_i - b'_i$, where the

$b_i, b'_i \in \{0,1\}$ are uniform independent bits. The distribution ψ_k is centered (its mean is 0), has variance $k/2$ and for $k = 8$ this gives a standard deviation of $\varsigma = \sqrt{8/2}$. In Section 4.1.1 a justification of the security of this design decision is given. As explained in Section 4.2, our choice of parameters in NewHope1024 leaves a comfortable margin to the targeted 128 bits of post-quantum security (NIST level 5), which accommodates for the slight loss in security indicated by Theorem 4.1 due to the use of the binomial distribution. Even more important from a practical point of view is that no known attack makes use of the difference in error distribution; what matters for attacks are entropy and standard deviation.

No backdoor. One serious concern in lattice-based cryptography may be the presence of constant polynomials, e.g., the fixed system parameter $\mathbf{a}$ in [31]. As described in Section 1.2.1, our proposal includes pseudorandom generation of this parameter for every key exchange. In the following we discuss the reasons for this decision.

In the worst scenario, the fixed parameter $\mathbf{a}$ could be backdoored. For example, inspired by NTRU trapdoors [83, 137], a dishonest authority may choose mildly small $\mathbf{f}, \mathbf{g}$ such that $\mathbf{f} = \mathbf{g} = 1 \bmod p$ for some prime $p \geq 4 \cdot 8 + 1$ and set $\mathbf{a} = \mathbf{g}\mathbf{f}^{-1} \bmod q$. Then, given $(\mathbf{a}, \mathbf{b} = \mathbf{a}\mathbf{s} + \mathbf{e})$, the attacker can compute $\mathbf{b}\mathbf{f} = \mathbf{a}\mathbf{f}\mathbf{s} + \mathbf{f}\mathbf{e} = \mathbf{g}\mathbf{s} + \mathbf{f}\mathbf{e} \bmod q$, and, because $\mathbf{g}, \mathbf{s}, \mathbf{f}, \mathbf{e}$ are small enough, compute $\mathbf{g}\mathbf{s} + \mathbf{f}\mathbf{e}$ in $\mathbb{Z}$. From this he can compute $\mathbf{t} = \mathbf{s} + \mathbf{e} \bmod p$ and, because the coefficients of $\mathbf{s}$ and $\mathbf{e}$ are smaller than 8, their sums are in $[-2 \cdot 8, 2 \cdot 8]$: knowing them modulo $p \geq 4 \cdot 8 + 1$ is knowing them in $\mathbb{Z}$. It now only remains to compute $(\mathbf{b} - \mathbf{t}) \cdot (\mathbf{a} - 1)^{-1} = (\mathbf{a}\mathbf{s} - \mathbf{s}) \cdot (\mathbf{a} - 1)^{-1} = \mathbf{s} \bmod q$ to recover the secret $\mathbf{s}$.

One countermeasure against such backdoors is the "nothing-up-my-sleeve" process, which would, for example, choose $\mathbf{a}$ as the output of a hash function on a common universal string like the digits of π. Yet, even this process may be partially abused [22], and when not strictly required it seems preferable to avoid it.

All-for-the-price-of-one attacks. Even if this common parameter has been honestly generated, it is still rather uncomfortable to have the security of all connections rely on a single instance of a lattice problem. The scenario is an entity that discovers an unforeseen cryptanalytic algorithm, making the required lattice reduction still very costly, but say, not impossible in a year of computation, given its outstanding computational power. By finding *once* a good enough basis of the lattice $\Lambda = \{(a,1)x + (q,0)y | x, y \in \mathcal{R}\}$, this entity could then compromise *all* communications, using for example Babai's decoding algorithm [15].

This idea of massive precomputation that is only dependent on a fixed parameter $\mathbf{a}$ and then afterwards can be used to break all key exchanges is similar in flavor to the 512-bit "Logjam" DLP attack [1]. This attack was only possible in the required time limit because most TLS implementations use fixed primes for Diffie-Hellman. One of the recommended mitigations by the authors of [1] is to avoid fixed primes.

Against all authority. Fortunately, all those pitfalls can be avoided by having the communicating parties generate a fresh $\mathbf{a}$ at each instance of the protocol (as we propose). If in practice it turns out to be too expensive to generate $\mathbf{a}$ for every connection, it is also possible to cache $\mathbf{a}$ on the server side[2] for, say a few hours without significantly weakening the protection against all-for-the-price-of-one attacks. Additionally, the performance impact of generating $\mathbf{a}$ is reduced by sampling $\mathbf{a}$ uniformly directly in NTT format (recalling that the NTT is a one-to-one map), and by transferring only a short 256-bit seed for $\mathbf{a}$.

A subtle question is to choose an appropriate primitive to generate a "random-looking" polynomial $\mathbf{a}$ out of a short seed. For a security reduction, it seems to the authors that there is no way around the (non-programmable) random oracle model (ROM). It is argued in [62] that such a requirement is in practice an overkill, and that any pseudorandom generator (PRG) should also work. And while it is an interesting question how such a reasonable pseudo-random generator would interact with our lattice assumption, the cryptographic notion of a PRG *is not* helpful to argue security. Indeed, it is an easy exercise[3] to build (under the NTRU assumption) a "backdoored" PRG that is, formally, a legitimate PRG, but that makes our scheme insecure. Instead, we prefer to base ourselves on a standard cryptographic hash-function, which is the typical choice of an "instantiation" of the ROM. As a suitable option we see Keccak [27], which has recently been standardized as SHA3 in FIPS-202 [112], and which offers extendable-output functions (XOF) named SHAKE. This avoids costly external iteration of a regular hash function and directly fits our needs. We use SHAKE128 for the generation of $\mathbf{a}$, which offers 128-bits of (post-quantum) security against collisions and preimage attacks. With only a small performance penalty we could have also chosen SHAKE256, but we do not see any

[2] But recall that the secrets $\mathbf{s}, \mathbf{e}, \mathbf{s}', \mathbf{s}', \mathbf{e}''$ have to be sampled fresh for every connection.

[3] Consider a secure PRG p, and parse its output $p(\text{seed})$ as two small polynomial $(\mathbf{f}, \mathbf{g})$: an NTRU secret-key. Define $p'(\text{seed}) = \mathbf{g}\mathbf{f}^{-1} \bmod q$: under the decisional NTRU assumption, p' is still a secure PRG. Yet revealing the seed does reveal $(\mathbf{f}, \mathbf{g})$ and provides a backdoor as detailed above.

Table 2: Parameters of NewHope512 and NewHope1024 and derived high-level properties.

Parameter Set	NewHope512	NewHope1024
Dimension n	512	1024
Modulus q	12289	12289
Noise parameter k	8	8
NTT parameter γ	49	7
Decryption error probability	2^{-213}	2^{-216}
Claimed post-quantum bit-security	101	233
NIST Security Strength Category	1	5

Table 3: Sizes of public keys, secret keys, and ciphertexts of our NewHope instantiations in bytes.

Parameter Set	$\|pk\|$	$\|sk\|$	$\|ciphertext\|$
NewHope512-CPA-KEM	928	869	1088
NewHope1024-CPA-KEM	1824	1792	2176
NewHope512-CCA-KEM	928	1888	1120
NewHope1024-CCA-KEM	1824	3680	2208

reason for such a choice, in particular because neither collisions nor preimages lead to an attack against the proposed scheme.

Short-term public parameters. NewHope does not rely on a globally chosen public parameter **a** as the efficiency increase in doing so is not worth the measures that have to be taken to allow trusted generation of this value and the defense against backdoors [22]. Moreover, this approach avoids the rather uncomfortable situation that all connections rely on a single instance of a lattice problem (see Section 1.3) in the flavor of the "Logjam" DLP attack [1].

1.4 Parameters

1.4.1 NewHope512 and NewHope1024

For our NewHope cryptosystem we specify the two parameter sets NewHope512 and NewHope1024 in Table 2. These parameter sets are used to instantiate the NewHope-CPA-KEM or NewHope-CCA-KEM scheme. In case the security level should be specified together with the scheme we use the exemplary notation NewHope1024-CPA-KEM to refer to the NewHope-CPA-KEM scheme instantiated with the NewHope1024 parameter set. In Table 3 we provide public key, secret key, and ciphertext sizes for our two KEMs that support the transmission of a 256-bit message or key. For the justification of the NIST level we refer to Section 5 and for the justification of the post-quantum bit-security we refer to Section 4.2.

The parameters in Table 2 fully define NewHope and all other intermediary parameters can be calculated from there. For convenience, we list intermediary parameters:

- NewHope512: $\gamma = \sqrt{\omega} = 49$; $\omega = 2401$; $\omega^{-1} \bmod q = 11813$; $\gamma^{-1} \bmod q = 1254$; $n^{-1} \bmod q = 12265$
- NewHope1024: $\gamma = \sqrt{\omega} = 7$; $\omega = 49$; $\omega^{-1} \bmod q = 1254$; $\gamma^{-1} \bmod q = 8778$; $n^{-1} \bmod q = 12277$

Note that the parameters of NewHope cannot be freely chosen. The dimension n has to be an integer power of two to support efficient NTT algorithms and to maintain the security properties of RLWE. Degrees that are not power of 2 are also possible, but come with several complications [104, 121], in particular the defining polynomial of the ring can not have the form $X^n + 1$ anymore.

Additionally, n has to be greater or equal than 256 due to our choice of the encoding function that needs to embed a 256-bit message into an n-dimensional polynomial in NewHope-CPA-PKE. The modulus q has to be chosen as integer prime q such that $q \equiv 1 \mod 2n$ to support efficient NTT algorithms. The integer parameter k of the noise distribution has to be chosen such that the probability of decryption errors is negligible. On a high-level, the final security of NewHope depends on (q, n, k) where a larger n and a

larger $\frac{k}{q}$ lead to a higher security level. The choice of γ does not have an impact on the security but is need for correctness (see Section 1.2) and is simply the smallest possible value.

In the unlikely case that a higher security level is required while confidence in the RLWE assumption remains, it is straightforward to choose a NewHopeLudicrous parameter set with dimension $n = 2048$ and $k = 8$. This would basically double execution times and the size of public keys, ciphertexts, secrete keys (maybe). A small increase in security for the NewHope-CPA-KEM is also possible. As the scheme should in practice only be used in an ephemeral setting where decryption errors are less critical it might be possible to slightly increase k (e.g., $k = 16$ as in NewHope-Usenix).

We do not belief that a larger modulus q will result in a performance benefit or better performance/security tradeoff. However, in case q is increased, the parameter k has to be adapted as well. Choosing n not as a power of two would render the scheme insecure. In general, RLWE-based schemes do not requires a prime modulus q for security or performance. However, as NewHope directly uses properties of a negacyclic NTT, parameters have to be chosen so that q is prime and that $q \equiv 1 \mod 2n$. A scheme without restrictions regarding the modulus q would look quite different than NewHope from an implementers perspective.

1.4.2 Toy/challenge parameters

We do no encourage the use of smaller dimensions that $n = 512$ for practical applications. As toy parameter set for cryptanalysis we propose NewHopeToy1 with $n = 256, q = 7681, k = 4$. An even smaller toy parameter NewHopeToy2 set with $n = 128, q = 256, k = 1$ could also be a target for cryptanalysis but would require the reduction of the length of the message supported by NewHope-CPA-PKE to 128-bit.

1.4.3 Cryptographic primitives

NewHope relies on the SHAKE hash function [112] for several purposes:

- NewHope-CPA-PKE uses SHAKE128 to generate the public parameters $\hat{\mathbf{a}}$ from a public seed *seed*. In this instance the assumption is that SHAKE128 acts as a public random function for the given output length. Additionally, SHAKE256 is used to hash and extend the output of the random number generator in key generation.
- NewHope-CPA-KEM uses SHAKE256 to derive intermediate random values and the shared secret. The assumption is that SHAKE256 is a pseudorandom function.
- NewHope-CCA-KEM uses SHAKE256 to derive random keys, intermediate random values, and the shared secret; and to hash the public key and ciphertext. When hashing the public key pk and ciphertext c, the assumption is that SHAKE256 is collision-resistant. When deriving *seed*, *rand*, s, μ, and ss, the assumption is that SHAKE256 is a pseudorandom function.

No padding is used in the derivations above. Multiple inputs are combined by concatenating bitstrings; lengths of the concatenated values are fixed.

1.4.4 Provenance of constants and tables

The following constants are used in NewHope:

- Dimension n: Selected as a power of two to support efficient NTT algorithms and to maintain the security of RLWE.
- Modulus q: Selected as the smallest prime such that $q \equiv 1 \mod 2n$ so that the number-theoretic transform (NTT) can be realized efficiently.
- Noise parameter k: Selected so that the probability of decryption errors is negligible.
- NTT parameter γ: Must be n-th primitive root of unity; we select the smallest such value.
- Domain separation in calls to SHAKE: $\hat{\mathbf{a}}$ is generated pseudorandomly using SHAKE128 from a seed; domain separators are used internally in this generation, and are simply selected as counters.

2 Performance analysis

2.1 Estimated performance on the NIST PQC reference platform

In this section we provide details on our reference implementation written in C and estimate its performance on the NIST PQC reference platform. In Table 4 we list the directory in which the code of our reference implementation located in the submission file. The respective Gen (`crypto_kem_keypair`), Encaps (`crypto_kem_enc`), and Decaps (`crypto_kem_dec`) functions for each instantiation are defined in the file `kem.c` in the respective directory. Note that the code of reference and optimized implementation is identical but we provide two directory structures for completeness. The size of the produced keys and ciphertexts is not dependent on the platform and the numbers can be found in Table 3.

Table 4: Directories of the code of our reference implementation.

Reference Implementation	
NewHope512-CPA-KEM	Reference_Implementation/crypto_kem/newhope512cpa
NewHope512-CCA-KEM	Reference_Implementation/crypto_kem/newhope512cca
NewHope1024-CPA-KEM	Reference_Implementation/crypto_kem/newhope1024cpa
NewHope1024-CCA-KEM	Reference_Implementation/crypto_kem/newhope1024cca

The main emphasis in the C reference implementation is on simplicity and portability. It does not use any floating-point arithmetic and outside of the Keccak (SHAKE256 and SHAKE128) implementation only needs 16-bit and 32-bit integer arithmetic.

NTT Implementation. All polynomial coefficients are represented as unsigned 16-bit integers. Our in-place NTT implementation transforms from bit-reversed to natural order using Gentleman-Sande butterfly operations [67, 42]. One would usually expect that each NTT is preceded by a bit-reversal, but all inputs to NTT are noise polynomials that we can simply consider as being already bit-reversed. This is supported in the description of the algorithm. As explained earlier, the NTT^{-1} operation still involves a bit-reversal. For $n = 1024$ the core of the NTT and NTT^{-1} operation consists of 10 layers of transformations, each consisting of 512 butterfly operations of the form described in Listing 2.

Montgomery arithmetic and lazy reductions. The performance of operations on polynomials is largely determined by the performance of NTT and NTT^{-1}. The main computational bottleneck of those operations are 2304 ($n = 512$) or 5120 ($n = 1024$) butterfly operations, each consisting of one addition, one subtraction and one multiplication by a precomputed constant. Those operations are in $\mathbb{Z}_q$; recall that q is a 14-bit prime. To speed up the modular-arithmetic operations, we store all precomputed constants in Montgomery representation [111] with $R = 2^{18}$, i.e., instead of storing ω^i, we store $2^{18}\omega^i \pmod q$. After a multiplication of a coefficient g by some constant $2^{18}\omega^i$, we can then reduce the result r to $g\omega^i \pmod q$ with the fast Montgomery reduction approach. In fact, we do not always fully reduce modulo q, it is sufficient if the result of the reduction has at most 14 bits. The fast Montgomery reduction routine given in Listing 1a computes such a reduction to a 14-bit integer for any unsigned 32-bit integer in $\{0, \ldots, 2^{32} - q(R-1) - 1\}$. Note that the specific implementation does not work for *any* 32-bit integer; for example, for the input $2^{32} - q(R-1) = 1073491969$ the addition `a=a+u` causes an overflow and the function returns 0 instead of the correct result 4095. In the following we establish that this is not a problem for our software.

Aside from reductions after multiplication, we also need modular reductions after addition, which, for the sake of simplicity and readability, are written as the `C` modulo operator `%`). An alternative and faster approach is to use use "short Barrett reduction" [19] as detailed in Listing 1b. Again, this routine does not fully reduce modulo q, but reduces any 16-bit unsigned integer to an integer of at most 14 bits which is congruent modulo q.

In the context of the NTT and NTT^{-1}, we make sure that inputs have coefficients of at most 14 bits. This allows us to avoid reductions after addition on every second level, because coefficients grow by at most one bit per level and the short Barrett reduction (and the `%` operator) can handle 16-bit inputs. Let us turn our focus to the input of the Montgomery reduction (see Listing 2). Before subtracting `a[j+d]` from `t` we need to add a

multiple of q to avoid unsigned underflow. Coefficients never grow larger than 15 bits and $3 \cdot q = 36867 > 2^{15}$, so adding $3 \cdot q$ is sufficient. An upper bound on the expression `((uint32_t)t + 3*12289 - a[j+d])` is obtained if `t` is $2^{15} - 1$ and `a[j+d]` is zero; we thus obtain $2^{15} + 3 \cdot q = 69634$. All precomputed constants are in $\{0, \ldots, q-1\}$, so the expression `(W * ((uint32_t)t + 3*12289 - a[j+d])`, the input to the Montgomery reduction, is at most $69634 \cdot (q-1) = 855662592$ and thus safely below the maximum input that the Montgomery reduction can handle.

Listing 1 Reduction routines used in the reference implementation.

(a) Montgomery reduction ($R = 2^{18}$).

```
uint16_t mred(uint32_t a) {
  uint32_t u;
  u = (a * 12287);
  u &= ((1 << 18) - 1);
  a += u * 12289;
  return a >> 18;
}
```

(b) Short Barrett reduction.

```
uint16_t bred(uint16_t a) {
  uint32_t u;
  u = ((uint32_t) a * 5) >> 16;
  a -= u * 12289;
  return a;
}
```

Listing 2 The Gentleman-Sande butterfly inside odd levels of our NTT computation. All `a[j]` and `W` are of type `uint16_t`.

```
  W = omega[jTwiddle++];
  t = a[j];
  a[j] = bred(t + a[j+d]);
  a[j+d] = mred(W * ((uint32_t)t + 3*12289 - a[j+d]));
```

Fast random sampling. As a first step before performing any operations on polynomials, both Alice and Bob need to expand the seed to the polynomial **a** using SHAKE256. The implementation we use is based on the "simple" implementation by Van Keer for the Keccak permutation and slightly modified code taken from the "TweetFIPS202" implementation [26] for everything else.

Implementation of GenA. The public parameter **a** is generated from a 32-byte seed through the extendable-output function SHAKE128 [112, Sec. 6.2]. The approach described here slightly differs from the approach described in [9]. Specifically, the coefficients of **a** are generated in $n/64$ independent blocks of 64 coefficients each. To generate block i (of coefficients ranging from a_{64i} to a_{64i+63}) is generated by concatenating the $32 - byte$ seed with a one-byte value of i and feeding the resulting 33-byte extended seed to SHAKE128 is then considered as an array of 16-bit, unsigned, little-endian integers. Each of those integers is used as a coefficient of **a** if it is smaller than $5q$ and rejected otherwise. The first such 16-bit integer is used as the coefficient a_{64i}, the next one as coefficient of a_{64i+1} and so on. Each block needs a total of 64 coefficients, so at least 128 bytes of output from SHAKE128. The probability that a 16-bit value is smaller than $5q$ is 93.75%, so the expected number of bytes of SHAKE128 output per block of **a** is 137. One block of output of SHAKE128 has 168 bytes, so with very large probability only one block of SHAKE128 output is required for each block of **a**.

Performance results. Benchmark results for our reference implementation are reported in Table 5 and were obtained on an Intel Core i7-4770K (Haswell) running at 3491.953 MHz with Turbo Boost and Hyperthreading disabled. We compiled our C reference implementation with gcc-4.9.2 and flags `-O3 -fomit-frame-pointer -march=native`. For all other routines we report the median of 1000 runs.

2.2 Performance on x86 processors using vector extensions

Intel processors since the "Sandy Bridge" generation support Advanced Vector Extensions (AVX) that operate on vectors of 8 single-precision or 4 double-precision floating-point values in parallel. With the introduction of the "Haswell" generation of CPUs, this support was extended also to 256-bit vectors of integers of various sizes (AVX2). It is not surprising that the enormous computational power of these vector instructions has been used before to implement very high-speed crypto (see, for example, [23, 74, 24]) and also our optimized reference implementation targeting Intel Haswell processors uses those instructions to speed up multiple

Table 5: Cycle counts of our NewHope C reference implementation compiled with gcc-4.9.2 on an Intel Core i7-4770K (Haswell) with Turbo Boost and Hyperthreading disabled.

Operation	NH-512-CPA-KEM	NH-512-CCA-KEM	NH-1024-CPA-KEM	NH-1024-CCA-KEM
NTT	21,772	21,772	49,920	49,772
NTT^{-1}	23,384	23,420	53,596	53,408
GenA	16,012	16,052	32,248	32,240
Gen	106,820	117,128	222,922	244,944
Encaps	155,840	180,648	330,828	377,092
Decaps	40,988	206,244	87,080	437,056

Table 6: Cycle counts of an additional NewHope implementation using AVX extensions compiled with gcc-4.9.2 on an Intel Core i7-4770K (Haswell) with Turbo Boost and Hyperthreading disabled.

Operation	NH-512-CPA-KEM	NH-512-CCA-KEM	NH-1024-CPA-KEM	NH-1024-CCA-KEM
NTT	4888	4820	8416	8496
NTT^{-1}	6352	6344	11,708	11,680
GenA	10,804	10,808	21,308	21,480
Gen	56,236	68,080	107,032	129,670
Encaps	85,144	109,836	163,332	210,092
Decaps	19,472	114,176	35,716	220,864

components of the key exchange. We have done an implementation of NewHope targeting such a vectorized architecture.

NTT optimizations. The AVX instruction set has been used before to speed up the computation of lattice-based cryptography, and in particular the number-theoretic transform. Most notably, Güneysu, Oder, Pöppelmann and Schwabe achieve a performance of only 4 480 cycles for a dimension-512 NTT on Intel Sandy Bridge [76]. For arithmetic modulo a 23-bit prime, they represent coefficients as double-precision integers.

We experimented with multiple different approaches to speed up the NTT in AVX. For example, we vectorized the Montgomery arithmetic approach of our C reference implementation and also adapted it to a 32-bit-signed-integer approach. In the end it turned out that floating-point arithmetic beats all of those more sophisticated approaches, so we are now using an approach that is very similar to the approach in [76]. One computation of a dimension-1024 NTT takes $\approx 8\,450$ cycles, unlike the numbers in [76] this does include multiplication by the powers of γ and unlike the numbers in [76], this excludes a bit-reversal.

Fast sampling. For the computation of SHAKE-128 we use the same code as in the C reference implementation. One might expect that architecture-specific optimizations (for example, using AVX instructions) are able to offer significant speedups, but the benchmarks of the eBACS project [25] indicate that on Intel Haswell, the fastest implementation is the "simple" implementation by Van Keer that our C reference implementation is based on. The reasons that vector instructions are not very helpful for speeding up SHAKE (or, more generally, Keccak) are the inherently sequential nature and the 5×5 dimension of the state matrix that makes internal vectorization hard.

Performance results. Benchmark results for our AVX implementation are reported in Table 6 and were obtained on an Intel Core i7-4770K (Haswell) running at 3491.953 MHz with Turbo Boost and Hyperthreading disabled. `-no-pie -O3 -fomit-frame-pointer -msse2avx -mavx2 -march=native`

2.3 Performance estimation on ARM Cortex-M0 and M4

For the performance estimation of NewHope on ARM Cortex-M0 and M4 micrcontrollers we refer to the implementation of NewHope-Usenix by Alkim, Jakubeit, and Schwabe [10]. Their work shows that an implementation of NewHope-Usenix can outperform an implementation of Curve25519 [21] for the

Table 7: Cycle counts of a NewHope-Usenix implementation on a Cortex-M0 and Cortex-M4 microcontroller obtained from [10].

Operation	Cortex-M0	Cortex-M4
NTT	148,517	87,223
NTT^{-1}	167,405	97,789
Generation of $\mathbf{a}$	380,855	293,975
Key generation (equiv. to Gen)	1,168,224	964,440
Key gen + shared key (equiv. to Encaps)	1,738,922	1,418,124
Shared key (equiv. to Decaps)	298,877	178,874
ROM usage (bytes)	30,178	22,828

Table 8: Cycle counts of an implementation of a CPA or CCA-secure public-key encryption scheme that is similar to NewHope (both use $n = 1024, q = 12289, k = 8$) on a Cortex-M0 and Cortex-M4 microcontroller obtained from [116].

Operation	Cycle Counts	
	Unmasked	Masked
Key generation	2,669,559	-
CCA2-secure encryption	4,176,684	-
CCA2-secure decryption	4,416,918	25,334,493
CPA-secure encryption	3,910,871	19,315,432
CPA-secure decryption	163,887	550,038
SHAKE128	87,738	201,997
NTT	83,906	-
NTT^{-1}	104,010	-
Uniform sampling (TRNG)	60,014	-
Noise sampling (PRNG)	1,142,448	6,031,463
PRNG (64 bytes)	88,778	202,454

Cortex-M0, like the one presented in [55], by more than a factor of two. Their cycle counts are given in Table 7.

Additionally, we also refer to the work by Oder, Schneider, Pöppelmann, and Güneysu [116] who describe an implementation of CCA2-secure public-key encryption that is similar to NewHope with and without side-channel countermeasures. In Table 8 we provide their results on a Cortex-M4. The instantiated scheme is ring-LWE public-key encryption ($n = 1024$, $q = 12289$, and binomial distribution with parameter $k = 8$) parametrized for negligible decryption errors so that the Fujisaki-Okamoto [59] transformation by Targhi and Unruh can be used [139]. The CCA2-secure encryption takes 4,176,684 cycles, which translates to 25 milliseconds when operating at a clock frequency of 168 MHz. Key generation takes 16 ms at 168 MHz. The application of the CCA2-conversion to the decryption causes a much higher overhead due to the necessary re-encryption. In the unmasked case, it requires 27 times more cycles.

2.4 Performance on MIPS64

Starting as an academic project in Stanford in the 1980s, the MIPS architecture is nowadays, typically utilized in network equipment, laser printers and consumer electronics. It was formerly part of superscalar processors (e.g. the MIPS I R2000 and R3000 of 32-bits, the R4000 of 64 bits (MIPS III) and the R10000 (MIPS IV)) and video game consoles (e.g. PlayStation 1–2 and Nintendo64). Developed by MIPS Technologies, Inc. (now Imagination Technologies[4]), the MIPS architecture is based on the RISC instruction set.

[4] https://www.imgtec.com/mips/

From year 2000, several synthesizable cores has appeared such as the 32-bit 4k, 24k and 64-bit 5k. Afterwards the MIPS32[5] and MIPS64[6] specifications were created. Nowadays, companies such as Loongson Technology, Cavium, Broadcom and Toshiba have licenses for MIPS64. Given the availability of the MIPS64 in the market in a myriad of different network routers, we have selected the MIPS64r3 release for optimizing NewHope1024 and NewHope512. Besides, the size of the available registers makes it ideal for vectorizing the polynomial arithmetic of the algorithm as well as for reducing memory access (this is the case, for instance, of our implementation of the Keccak-f[1600] permutation).

Our target device is a 28 nm cnMIPS III core from a CN7130 SoC, based on the MIPS64r3 architecture, clocked at 1.6 GHz and equipped with 78K instruction cache, 32K data cache and a floating point unit.

The MIPS64r3 architecture. The MIPS64r3 architecture has 32 64-bits registers and standardizes three co-processor encoding regions: CP0 for CPU configuration, cache control, interrupt control and memory management, CP1 is the FPU and CP2 available to other peripherals. The register file is comprised of 64-bit 32 registers where 27 can be used: v0-v1 (value returned by subroutines), a0-a3 (subroutine parameters), t0-t9 (temporary), s0-s8 (subroutine registers), gp (global pointer) and ra (return address). Further, co-processor CP1 provides an additional set of 32 64-bit registers, typically part of the FPU.

NTT optimization. As mentioned before, our choice for a 64-bits architecture instead of the MIPS32 one is largely based on the idea of vectorizing the NTT implementation. In so doing, we apply the following strategy: first, we parallelize the NTT by vectorizing the butterfly operation, second, and directly related to the former idea, we merge the layers of the NTT. Also both the Montgomery and Barrett reductions have been adapted for dealing with 64-bit integers.

Our approach is to parallelize the execution of the NTT by processing more than one coefficient (16 bits) in the architecture registers of 64-bit length. Out of the 32 registers of the MIPS64 architecture, only 27 are available to us for processing the 1,024 or 512 coefficients of the NTT. This way, we could execute one NTT's butterfly operation for n pairs of coefficients in parallel. However, due to the overflow bits of the butterfly operations (addition and multiplication) we can store 2 coefficients in 1 register. In our implementation, we use 16 registers for storing the coefficients, meaning that is actually possible to process $2 \cdot 16 = 32$ coefficients at a time, merging at most $log_2\ 32 = 5$ layers. Nonetheless due to the fact that after the 5^{th} layer the blocks of coefficients must be chosen from a different block of coefficients, that is, not from an adjacent one, we merge 4 layers.

Table 9: Performance figures of the NTT on MIPS64 in number of cycles.

Implementation	no optimization (#cycles)	-O2 (#cycles)	-O3 (#cycles)	opt (#cycles)
NTT1024 (NewHope1024 c32)	439,970	196,989	196,990	-
NTT512 (NewHope512 c32)	197,296	86,651	86,647	-
NTT1024 (vectorized)	-	-	-	85,348
NTT512 (vectorized)	-	-	-	38,755

Polynomial arithmetic vectorization. The same approach we utilized for optimizing the NTT can be applied to implement the polynomial arithmetic operations of NewHope1024 and NewHope512. However, in order to perform coefficients multiplication and pointwise multiplication via vectorization, both 128-bit registers and respective SIMD instructions are required in order to avoid overflows. Since our target architecture lacks both components we have only addressed the vectorization of the coefficient addition whose overflow can be controlled. Further, in order to reduce the impact of our strategy, we reused the coefficient storage method describe in the prior subsection. In this respect, we can perform two coefficient additions using a single addition instruction after an NTT has been performed.

Keccak. The XOF SHAKE used in NewHope relies on the Keccak-f[1600] permutation [27]. Since the main operations against internal state of $5x5$ are performed on 64-bit words, MIPS64 is ideal for doing the permutation directly on 64-bit registers. Besides, loading words into the state is also done using 64-bit words

[5] https://www.imgtec.com/mips/architectures/mips32/
[6] https://www.imgtec.com/mips/architectures/mips64/

and cycle reduction is achieved by (1) maintaining the 25 64-bit words of the state in registers exploiting the large amount of registers available in the MIPS64 architecture during the ρ and χ layers and (2) reducing memory access in the computation of the ρ layer by storing the input values directly from the θ layer.

However, when comparing our implementation with other architectures, the lack of a Bitwise Bit Clear BIC instruction in the MIPS64r3 ISA (available for instance in the ARMv7-M architecture[7]) creates a small performance penalty, since every operation of the χ layer requires three instructions (that is, one *and*, one *or* and one *xor* operation).

Performance results. In order to estimate the number of cycles required by each primitive of NewHope1024 and NewHope512 we rely on the performance counters of the coprocessor 0 (CP0). In so doing, we set up the performance counters in the before the start of of the primitive and measure again after the execution has finished.

Table 10: Performance figures of NewHope1024 and NewHope512 (CPA) on MIPS64 in number of cycles.

Implementation	no optimization (#cycles)	`-O2` (#cycles)	`-O3` (#cycles)	`opt` (#cycles)
NewHope1024	8,421,677	3,099,586	2,456,793	1,705,203
NewHope-CPA-KEM Key Generation	2,948,707	1,114,148	857,679	589,613
NewHope-CPA-KEM Encapsulation	4,378,938	1,645,474	1,277,601	882,443
NewHope-CPA-KEM Decapsulation	1,093,115	339,177	322,128	232,543
NewHope512	4,079,865	1,518,736	1,186,890	864,812
NewHope-CPA-KEM Key Generation	1,425,656	544,466	413,041	299,922
NewHope-CPA-KEM Encapsulation	2,123,245	806,941	617,037	448,791
NewHope-CPA-KEM Decapsulation	530,569	167,074	156,203	115,767

Table 11: Performance figures of NewHope1024 and NewHope512 (CCA) on MIPS64 in number of cycles.

Implementation	no optimization (#cycles)	`-O2` (#cycles)	`-O3` (#cycles)	`opt` (#cycles)
NewHope1024	14,466,351	5,524,430	4,290,417	2,871,081
NewHope-CCA-KEM Key Generation	3,329,101	1,298,382	981,655	651,810
NewHope-CCA-KEM Encapsulation	5,078,734	1,481,606	1,517,853	1,021,702
NewHope-CCA-KEM Decapsulation	6,058,512	2,244,514	1,791,563	1,197,556
NewHope512	7,029,073	2,702,808	2,063,742	1,473,698
NewHope-CCA-KEM Key Generation	1,611,787	633,120	470,209	334,100
NewHope-CCA-KEM Encapsulation	2,485,018	979,139	734,888	530,373
NewHope-CCA-KEM Decapsulation	2,931,568	1,089,937	858,111	609,107

Our performance figures suggest that it is possible to achieve a reduction of 196,990 - 85,348 = 111,642 cycles in the NTT computation from NewHope1024 (that is, a speed up of factor 2.3) and a reduction of 86,647 - 38,755 = 47,892 cycles in the NTT computation from NewHope1024 (that is, a speed up of factor 1.8) (See Table 9). With `opt` we refer to the implementations based on the optimization techniques described in this section. Besides, we noticed an overall reduction of factor 1.49 and factor 1.40 in the computation of the whole protocol (NewHope1024 and NewHope512 respectively (CCA-KEM)). These improvements were obtained by comparing our results with a compilation of NewHope1024/ NewHope512 using an aggressive optimization option (`-O3`)[8] (Table 11).

[7] http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0406c/index.html

[8] Using `gcc` version 4.7.0 for cross-compiling on Linux `4.12.0-2-amd64`

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 NewHope512-CPA-KEM are in the subdirectory `KAT/newhope512cpa`, the KAT values of NewHope512-CCA-KEM are in the subdirectory `KAT/newhope512cca`, the KAT values of NewHope1024-CPA-KEM are in the subdirectory `KAT/newhope1024cpa`, and the KAT values of NewHope1024-CCA-KEM are in the subdirectory `KAT/newhope1024cca`. Each of those directories contains the KAT values as generated by the `PQCgenKAT_kem` program provided by NIST. Specifically, those files are:

- `KAT/newhope512cpa/PQCkemKAT_896.req`
- `KAT/newhope512cpa/PQCkemKAT_896.rsp`
- `KAT/newhope512cca/PQCkemKAT_1888.req`
- `KAT/newhope512cca/PQCkemKAT_1888.rsp`
- `KAT/newhope1024cpa/PQCkemKAT_1792.req`
- `KAT/newhope1024cpa/PQCkemKAT_1792.rsp`
- `KAT/newhope1024cca/PQCkemKAT_3680.req`
- `KAT/newhope1024cca/PQCkemKAT_3680.rsp`

4 Justification of security strength

4.1 Provable security reductions

A summary of the provable security reductions underlying the security of NewHope-CCA-KEM is as follows:

1. Using the centred binomial distribution ψ_k instead of a discrete Gaussian distribution provides negligible advantage to an adversary. Section 4.1.1 gives a justification.
2. Using a pseudorandomly generated $\hat{\mathbf{a}}$ in NewHope-CPA-PKE instead of a uniformly random $\hat{\mathbf{a}}$ provides no advantage to an adversary, under the assumption that SHAKE128 is a random oracle.
3. NewHope-CCA-KEM is an IND-CCA-secure KEM under the assumption that NewHope-CPA-PKE is an IND-CPA-secure public key encryption scheme and that G and F (both instantiated as SHAKE256 are random oracles. Theorem 4.2 gives a tight, classical reduction against classical adversaries in the classical random oracle model. Theorem 4.3 gives a non-tight, classical reduction against quantum adversaries in the quantum random oracle model.
4. NewHope-CPA-PKE is an IND-CPA-secure public key encryption scheme under the assumption that the decision ring learning with errors problem is hard. Theorem 4.4 gives a tight, classical reduction against classical or quantum adversaries in the standard model.
5. The decision ring learning with errors problem is hard under the assumption that the search version of the approximate shortest vector problem is hard (in the worst case) on ideal lattices in $\mathcal{R}$, for appropriate parameters. Lyubashevsky et al. [102, Thm. 3.6] give a polynomial-time quantum reduction against classical or quantum adversaries in the standard model. See also [120, Thm. 2.7] for a simplified version of this result.

4.1.1 Binomial noise distribution

The original worst-case to average-case reductions for LWE [127] and Ring-LWE [103] state hardness for *continuous Gaussian* distributions (and therefore also trivially apply to *rounded Gaussians*, which differ from discrete Gaussians). This also extends to discrete Gaussians [35] but such proofs are not necessarily intended for direct implementations. The use of discrete Gaussians (or other distributions with very high-precision sampling) is only crucial for signatures [101] and lattice trapdoors [70], to provide zero-knowledgeness.

The following theorem states that choosing ψ_k as error distribution in NewHope-CPA-KEM (i.e., using the algorithm Sample) does not significantly decrease security compared to a rounded Gaussian distribution with the same standard deviation $\sigma = \sqrt{8/2}$.

Theorem 4.1 *Let ξ be the rounded Gaussian distribution of parameter $\sigma = \sqrt{4}$, that is, the distribution of $\lfloor \sqrt{4} \cdot x \rceil$ where x follows the standard normal distribution. Let $\mathcal{P}$ be the idealized version of* NewHope-CPA-KEM*, where outputs from* Sample *are replaced by samples from ξ. If an (unbounded) algorithm, given as input the public key and ciphertext of* NewHope-CPA-KEM *succeeds in recovering the shared secret ss with probability p, then it would also succeed against $\mathcal{P}$ with probability at least*

$$q \geq p^{9/8} \cdot 2^{-14}.$$

The result also holds for NewHope-CCA-KEM*.*

In [17], Bai et al. identify Rényi divergence as a powerful tool to improve or generalize security reductions in lattice-based cryptography. We review the key properties. The Rényi divergence [129, 17] is parametrized by a real $a > 1$, and defined for two distributions P, Q by:

$$R_a(P\|Q) = \left(\sum_{x \in \mathrm{Supp}(P)} \frac{P(x)^a}{Q(x)^{a-1}} \right)^{\frac{1}{a-1}}.$$

It is multiplicative: if P, P' are independent, and Q, Q' are also independent, then $R_a(P \times P'\|Q \times Q') \leq R_a(P\|Q) \cdot R_a(P'\|Q')$. Finally, Rényi divergence relates the probabilities of the same event E under two different distributions P and Q:

$$Q(E) \geq P(E)^{a/(a-1)} / R_a(P||Q).$$

Proof For our argument, recall that because the final shared key ss is obtained through hashing as $ss \leftarrow \mathsf{SHAKE256}(K)$ before being used, then, in the random oracle model (ROM), any successful attacker must recover K exactly. We call this event E. We also define ξ to be the rounded Gaussian distribution of parameter $\sigma = \sqrt{k/2} = \sqrt{4}$, that is the distribution of $\lfloor \sqrt{4} \cdot x \rceil$ where x follows the standard normal distribution.

A simple script in [9] computes $R_9(\psi_8\|\xi) \approx 1.002$. Yet because $5n$ samples are used per instance of the protocol, we need to consider the divergence $R_9(P\|Q) = R_9(\psi_8, \xi)^{5n}$ where $P = \psi_8^{5n}$ and $Q = \xi^{5n}$. For $n = 512$ we get $R_9(P\|Q) \approx 164$, and for $n = 1024$, we get $R_9(P\|Q) \approx 26889$. In both cases, $R_9(P\|Q) \leq 2^{14}$. ∎

The choice $a = 9$ is rather arbitrary but seemed a good trade-off between the coefficient $1/R_a(\psi_8\|\xi)$ and the exponent $a/(a-1)$. This reduction is provided as a safeguard: switching from Gaussian to binomial distributions can not dramatically decrease the security of the scheme. With practicality in mind, we will simply ignore the loss factor induced by the above reduction, since the best-known attacks against LWE do not exploit the structure of the error distribution, and seem to depend only on the standard deviation of the error (except in extreme cases [14, 90]).

4.1.2 Security of IND-CCA KEM

Theorem 4.2 (IND-CPA PKE $\implies$ IND-CCA KEM in classical ROM) *We define a public key encryption scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with message space $\mathcal{M}$ and which is δ-correct. Let G and F be independent random oracles. Let $\mathsf{QKEM}_m^{\not\perp'} = \mathsf{QFO}_m^{\not\perp'}[\mathsf{PKE}, G, F]$ be the KEM obtained by applying the $\mathsf{QFO}_m^{\not\perp'}$ transform as in subsubsection 1.2.3. For any classical algorithm $\mathcal{A}$ against the IND-CCA security of $\mathsf{QKEM}_m^{\not\perp'}$ that makes q_G and q_F queries to its G and F oracles, there exists a classical algorithm $\mathcal{B}$ against the IND-CPA security of* PKE *such that*

$$\mathrm{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{QKEM}_m^{\not\perp'}}(\mathcal{A}) \leq \frac{4 \cdot q_{\mathsf{RO}} + 1}{|\mathcal{M}|} + q_{\mathsf{RO}} \cdot \delta + 3 \cdot \mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{PKE}}(\mathcal{B})$$

where $q_{\mathsf{RO}} = q_G + q_F$. Moreover, the running time of $\mathcal{B}$ is about that of $\mathcal{A}$.

Theorem 4.2 follows from Theorems 3.2 and 3.4 of Hofheinz, Hövelmanns, and Kiltz [85], with the following modifications. In the application of HHK's Theorem 3.2, we take $q_V = 0$. Note that Theorems 3.2 and 3.4 of HHK are about the $\mathsf{FO}^{\not\perp}$ transform, which differs from the $\mathsf{QFO}_m^{\not\perp'}$ in the following ways. 1) $\mathsf{QFO}_m^{\not\perp'}$

uses a single hash function (with longer output) to compute K and $coin'$ whereas $\mathsf{FO}^{\not\perp}$ uses two; but this is equivalent in the random oracle model with appropriate output lengths. 2) $\mathsf{QFO}_m^{\not\perp\prime}$'s computation of K and $coin'$ also takes the public key pk as input whereas $\mathsf{FO}^{\not\perp}$ does not; this does not negatively affect any of the theorems, and has the potential to provide multi-target security. 3) $\mathsf{QFO}_m^{\not\perp\prime}$ includes the d value in the ciphertext, whereas $\mathsf{FO}^{\not\perp}$ does not; since d is computed by applying a random oracle G to the secret $\mu \in \mathcal{M}$, taking advantage of d requires querying G on μ, which occurs with the additional $\frac{q}{|\mathcal{M}|}$ probability term added in the theorem.

Theorem 4.3 (IND-CPA PKE $\implies$ IND-CCA KEM in quantum ROM) *We define a public key encryption scheme* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *with message space* $\mathcal{M}$ *and which is* δ*-correct. Let* G *and* F *be independent random oracles. Let* $\mathsf{QKEM}_m^{\not\perp\prime} = \mathsf{QFO}_m^{\not\perp\prime}[\mathsf{PKE}, G, F]$ *be the KEM obtained by applying the* $\mathsf{QFO}_m^{\not\perp\prime}$ *transform as in subsubsection 1.2.3. For any quantum algorithm* $\mathcal{A}$ *against the IND-CCA security of* $\mathsf{QKEM}_m^{\not\perp\prime}$ *that makes* q_G *and* q_F *queries to its quantum* G *and* F *oracles, there exists a quantum algorithm* $\mathcal{B}$ *against the IND-CPA security of* PKE *such that*

$$\mathrm{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{QKEM}_m^{\not\perp\prime}}(\mathcal{A}) \leq 9 \cdot q_{\mathsf{RO}} \cdot \sqrt{q_{\mathsf{RO}}^2 \cdot \delta + q_{\mathsf{RO}} \cdot \sqrt{\mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{PKE}}(\mathcal{B}) + \frac{1}{|\mathcal{M}|}}}$$

where $q_{\mathsf{RO}} = q_G + q_F$*. Moreover, the running time of* $\mathcal{B}$ *is about that of* $\mathcal{A}$*.*

Theorem 4.3 follows from Lemma 2.3 and Theorems 4.4 and 4.6 of Hofheinz, Hövelmanns, and Kiltz [85], with the following modifications. Note that Theorems 4.4 and 4.6 of HHK are about the $\mathsf{QFO}_m^{\not\perp}$ transform, which differs from the $\mathsf{QFO}_m^{\not\perp\prime}$ in the following ways. 1) $\mathsf{QFO}_m^{\not\perp\prime}$ uses a single hash function (with longer output) to compute K, $coin'$, and d whereas $\mathsf{FO}^{\not\perp}$ uses two; but this is equivalent in the random oracle model with appropriate output lengths. 2) $\mathsf{QFO}_m^{\not\perp\prime}$'s computation of K, $coin'$, and d also takes the public key pk as input whereas $\mathsf{FO}^{\not\perp}$ does not; this does not negatively affect any of the theorems, and has the potential to provide multi-target security. 3) $\mathsf{QFO}_m^{\not\perp\prime}$'s computation of the shared secret ss also takes the encapsulation $\overline{c}$ as input; this does not negatively affect any of the theorems, and provides robustness against ciphertext modification.

4.1.3 Security of IND-CPA PKE

Theorem 4.4 (dRLWE $\implies$ IND-CPA security of NewHope-CPA-PKE) *Let* n *and* q *be integers. Let* χ *be a probability distribution on* $\mathcal{R}_q$*. For any quantum algorithm* $\mathcal{A}$ *against the IND-CPA security of* NewHope-CPA-PKE *(with uniformly random* $\hat{\mathbf{a}}$*), there exists quantum algorithms* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *against the decision ring-LWE problem such that*

$$\mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\text{NewHope-CPA-PKE}}(\mathcal{A}) \leq \mathrm{Adv}^{\text{dRLWE}}_{n,q,\chi}(\mathcal{B}_1) + \mathrm{Adv}^{\text{dRLWE}}_{n,q,\chi}(\mathcal{B}_2) \ .$$

Moreover, the running times of $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *are about that of* $\mathcal{A}$*.*

The proof of Theorem 4.4 is essentially the same as that of Lemma 4.1 of [120] or Theorem 1 of [31].

4.2 Cryptanalytic attacks

For our security analysis in this section we mainly rely on the (very pessimistic) concrete security analysis of Ring-LWE based cryptosystems from [9]. Additionally, we also estimate the security level with the approach presented in [7].

4.2.1 Methodology: the core SVP hardness

RLWE as LWE. We analyze the hardness of Ring-LWE as an LWE problem, since, so far, the best known attacks do not make use of the ring structure. Indeed, while some new quantum algorithms against Ideal-SVP recently appeared [56, 39, 28, 44, 45], they do not seem to affect Ring-LWE. Precisely, in [45] two obstacles are discussed. First the approximation factor reached are asymptotically sub-exponential and it is therefore unlikely to affect cryptographic parameters. Secondly, Ring-LWE is proven to be at least as hard as Ideal-SVP,

but the natural approach for a converse reduction seems to require the ring $\mathbb{Z}[X]/(X^n+1)$ to be Euclidean, which is only the case for $n \in \{1, 2, 4\}$ (see [97]).

Attacks against LWE. There are many algorithms to consider in general (see the survey [7]), yet many of those are irrelevant for our parameter set. In particular, because there are only $m = n$ samples available one may rule out BKW types of attacks [90] and linearization attacks [14]. This essentially leaves us with two BKZ [136, 41] attacks, usually referred to as primal and dual attacks that we will briefly recall below.

The algorithm BKZ proceeds by reducing a lattice basis using an SVP oracle in a smaller dimension b. It is known [78] 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 [41, 40, 12]. 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.

4.2.2 Enumeration versus quantum sieve

Typical implementations of BKZ [65, 41, 38] use an enumeration algorithm as its SVP oracle, yet this algorithm runs in super-exponential time $2^{\Theta(n \log n)}$. On the other hand, the sieve algorithms are known to run in exponential time, but are so far slower in practice for accessible dimensions $b \approx 130$. In recent work Ducas [53] has shown that sieving techniques (in the classical setting) can be used in practice for exact-SVP, being now less than an order of magnitude slower than enumeration already in dimension 60 to 80.

For simplicity and conservatism, we will choose a reasonable lower bound for both enumeration and sieving. Namely, our bounds follow the asymptotic complexity of sieving algorithms, yet ignoring sub-exponential factors, when calculating cost in those attacks. According to the prediction of [41], even with Grover acceleration, the cost of enumeration is also lower-bounded by our estimates for blocksizes $b \geq 250$.

Quantum sieve. A lot of recent work has pushed the efficiency of the original lattice sieve algorithms [114, 109], 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 [93, 20]. The hidden sub-exponential factor is known to be much greater than one in practice, so again, estimating the cost ignoring this factor leaves us with a significant pessimistic margin.

Most of those algorithms have been shown [94, 92] to benefit from Grover's quantum search algorithm, bringing the complexity down to $2^{0.265b}$. It is unclear if further improvements are to be expected, yet, because all those algorithms require classically building lists of size $\sqrt{4/3}^{b+o(b)} \approx 2^{0.2075b}$. It is thus very plausible that the best quantum SVP algorithm would run in time greater than $2^{0.2075b}$.

Discarding enumeration for our analysis. In [41], predictions of the cost of solving SVP classically using sophisticated heuristic enumeration algorithms are given. For example, solving SVP in dimension 100 requires visiting about 2^{39} nodes, and 2^{134} nodes in dimension 250. Note that the cost for enumeration are here given in term of visited nodes in the enumeration tree, and visiting each of those nodes require about 100 cycles according to [41]. For simplicity and conservatism, we will assume that each node requires only one cycle.

Because this enumeration is a backtracking algorithm, it does benefit from the recent quasi-quadratic speedup [110], decreasing the quantum cost to about at least 2^{20} to 2^{67} operations as the dimension increases from 100 to 250. Again, this is quite conservative as the quantum version of this backtracking algorithm is subject to slowdowns polynomial in the depth of the tree.

On the other hand, our best-known attack bound $2^{0.265b}$ gives a cost of 2^{66} in dimension 250, and the best plausible attack bound $2^{0.2075b} \approx 2^{39}$. Because enumeration is super-exponential (both in theory and practice), its cost will be worse than our bounds in dimension larger than 250 and we may safely ignore this algorithm.[9]

We note that a recent technique formalized as discrete pruning [60, 11] seems to outperform the previous pruned enumeration of [41]. Unfortunately, no tools are currently available to fully predict the cost of this new techniques. We hope that future work will clarify these issues. Our current understanding is that this methods visits more nodes of the enumeration tree, but visit them much faster by removing the intricate backtracking steps. By counting the number of visited nodes rather than the count of CPU cycles, our lower bound should therefore also apply to this new discrete pruning technique.

[9] The numbers are taken from the latest full version of [41] available at http://www.di.ens.fr/~ychen/research/Full_BKZ.pdf.

4.2.3 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 $\Lambda = \{\mathbf{x} \in \mathbb{Z}^{m+n+1} : (\mathbf{A}|-\mathbf{I}_m|-\mathbf{b})\mathbf{x} = \mathbf{0} \bmod q\}$ of dimension $d = m + n + 1$, volume q^m, and with a unique-SVP solution $\mathbf{v} = (\mathbf{s}, \mathbf{e}, 1)$ of norm $\lambda \approx \varsigma\sqrt{n+m}$. Note that the number of used samples m may be chosen between 0 and $2n$ 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 $\delta = ((\pi b)^{1/b} \cdot b/2\pi e)^{1/2(b-1)}$ [40, 7]. 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

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

We note that this analysis introduced in [9] differs and is more conservative than prior works, which were typically based on the hardness of unique-SVP estimates of [63]. The validity of the new analysis has been confirmed by further analysis and experiments in [6].

4.2.4 Dual attack

The dual attack consists of finding a short vector in the dual lattice $\mathbf{w} \in \Lambda' = \{(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^m \times \mathbb{Z}^n : \mathbf{A}^t\mathbf{x} = \mathbf{y} \bmod q\}$. 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.

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

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

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 provide $2^{0.2075b}$ 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.

4.2.5 Security analysis

The cost of the primal attack and dual attacks are given in Table 12. They were obtained by executing our script in `scripts/PQsecurity.py`. According to our analysis, we claim that our proposed parameters for NewHope1024 offer 233 bits of security. Thus we are stronger (and quite likely with a large margin) than a post-quantum security level of 128 bits. In particular, NewHope1024 could even withstand a dimension-halving attack in the line of [66, Sec 8.8.1] based on the Gentry-Szydlo algorithm [71, 98] or the subfield approach of [5]. Note that so far, such attacks are only known for principal ideal lattices or NTRU lattices, and there are serious obstructions to extend them to Ring-LWE, but such precaution seems reasonable until lattice cryptanalysis stabilizes. For our NewHope512 parameter set we claim 101 bits of security.

In addition to our own analysis, we have used a freely available tool to evaluate the concrete security of LWE instances [7]. This approach is less pessimistic than our original security analysis, in particular it takes

Attack	m	b	Known Classical	Known Quantum	Best Plausible
BCNS proposal [31]: $q = 2^{32} - 1$, $n = 1024$, $\varsigma = 3.192$					
Primal	1062	296	86	78	61
Dual	1055	296	86	78	61
NTRUEncrypt [82]: $q = 2^{12}$, $n = 743$, $\varsigma \approx \sqrt{2/3}$					
Primal	613	603	176	159	125
Dual	635	600	175	159	124
JarJar-Usenix [9]: $q = 12289$, $n = 512$, $\varsigma = \sqrt{12}$					
Primal	623	449	131	119	93
Dual	602	448	131	118	92
NewHope-Usenix [9]: $q = 12289$, $n = 1024$, $\varsigma = \sqrt{8}$					
Primal	1100	967	282	256	200
Dual	1099	962	281	255	199
NewHope512: $q = 12289$, $n = 1024$, $\varsigma = \sqrt{4}$					
Primal	540	384	112	101	79
Dual	545	383	112	101	79
NewHope1024: $q = 12289$, $n = 1024$, $\varsigma = \sqrt{4}$					
Primal	999	886	259	235	183
Dual	1048	881	257	233	182

Table 12: Core hardness of NewHope512 and NewHope1024 and selected other proposals from the literature as well as previous instantiations of NewHope. The value b denotes the block dimension of BKZ, and m the number of used samples. Cost is given in $\log_2$ of CPU operations and is the smallest cost for all possible choices of m and b. Note that our estimation is very optimistic about the abilities of the attacker so that our result for the parameter set from [31] *does not* indicate that it can be broken with $\approx 2^{80}$ bit operations, given today's state-of-the-art in cryptanalysis.

account for the number of SVP calls, and estimate the cost of classical sieving to $2^{.292b+16}$. In Table 13 we provide the results for NewHope512 and NewHope1024. These values have been obtained by executing for different values of n and k using the sage module as follows:

```
load("https://bitbucket.org/malb/lwe-estimator/raw/HEAD/estimator.py")
k = 8.0; n = 1024; q = 12289; stddev = sqrt(k/2); alpha = alphaf(sigmaf(stddev), q)
_ = estimate_lwe(n, alpha, q, reduction_cost_model=BKZ.sieve)
```

The estimation in Table 13 also leads to the conclusion that NewHope1024, with a security level of 289 bits, reaches well beyond a security level of 128 bits. For NewHope512 a bit-security level of 142 bits is obtained. Most other (R)LWE-based or NTRU-based proposals achieve considerably lower security than NewHope1024. For comparison we also give a lower bound on the security of [31] and do notice a significantly improved security in our proposal. Yet, because of the numerous pessimistic assumption made in our analysis, we do not claim any quantum attacks reaching those bounds. The highest-security parameter set used for RLWE encryption in [72] is very similar to the parameters of JarJar-Usenix. The situation is different for NTRUEncrypt, which has been instantiated with parameters that achieve about 128 bits of security according to our analysis[10]. Specifically, we refer to NTRUEncrypt with $n = 743$ as suggested in [82]. A possible advantage of NTRUEncrypt compared to NewHope is somewhat smaller message sizes, however, this advantage becomes very small when scaling parameters to achieve a similar security margin as NewHope.

[10]For comparison we view the NTRU key-recovery as an homogeneous Ring-LWE instance. We do not take into account the combinatorial vulnerabilities [86] induced by the fact that secrets are ternary. We note that NTRU is a potentially a weaker problem than Ring-LWE: it is in principle subject to a subfield-lattice attack [5], but the parameters proposed for NTRUEncrypt are immune.

	usvp	dec	dual
JarJar-Usenix	161	198	185
NewHope-Usenix	313	410	356
NewHope512	142	171	163
NewHope1024	289	373	334

Table 13: Hardness of NewHope512 and NewHope1024 in the model of [7]. The analysis is based on a cost of $2^{.292b+16.4}$ for each call to the (classical) sieve of [20], an estimate that lies between our lower bound of $2^{.292b}$, and the measured cost in practice [20, 105]. This models also account for the number of calls to the sieve inside BKZ. The 'usvp' attack is similar what we call the 'primal' attack, while the 'dec' attack is a weaker variation trying to solve BDD without embedding it to a unique-SVP problem, by first reducing the primal lattice and then decoding the target vector using Babai decoding. Please refer to [7] for details. The dual attack they consider is also similar to the one described above.

4.2.6 Cost model and margins

Considering that lattice cryptanalysis is not a fully matured research area and that substantial improvement are still appearing, it seems preferable to leave significant margins in our security claims. The state-of-the art is unfortunately not as refined for lattice algorithm as it is can be for a memory-less brute-force attack on AES [73]. An analysis based on the current state of the art, using a model as refined as suggested by the call for proposal seems, in our case, way too intricate, prone to mistakes, and would likely become irrelevant within a few year.

We prefer to perform our analysis in simpler model; yet all the simplifications are done in favor of the attacker, and serve as margins. Those simplifications also contribute to make our analysis and scripts easier to verify. We list them here.

Asymptotic versus concrete. We used theoretical complexity of Sieving omitting sub-exponential factors. For the best asymptotic sieve algorithm [20] with complexity $2^{.292b+o(b)}$, it is typically reported that the fitted practical complexity $f \cdot 2^{cb}$ is quite larger than $2^{.292b}$, including a large constant factor f and a constant c. For example, the initial implementation [20] reports a fit of about $2^{.387b+16}$ clock-cycles on a `x86-64` CPU, in the range of dimensions $60 - 80$.

We prefer not to conclude on a quantified margin considering further works. Indeed, the implementation of [105] reports significant speed-ups using fine tuning and low-level optimizations (up to $\times 50$), but unfortunately no fit is provided. Further improvements are expected by combining those techniques with the very recent SubSieve algorithm of [53]: quantified claims seems premature.

Core SVP hardness versus BKZ. We also ignored the fact that the attacks actually require polynomially many calls to BKZ, and the best concrete predictions are typically based on simulations [41, 7]. Those simulation gets more complicated to perform as we include more techniques such as [12]. One may doubt the reliability of such simulations and fear further improvement of BKZ strategies. Moreover, some amortization strategies of sieving in BKZ sketched in [53] remains to be studied. Our core SVP-hardness approach dismisses those concerns.

CPU cycles versus gates. We also note that the concrete cost measured above is expressed in numbers of `x86-64` CPU cycles, rather than gate count. If one wishes to evaluate the gate count, we warn against naive implementations whose main cost are derived from 'Multiply-and-Add' operations inside inner-product loops. Some recent works [57, 53] show how to avoid most inner-products, resorting primarily to 'xor-popcount' operations.

RAM model versus circuits. The complexity of those algorithms have been analyzed in the quantumly accessible RAM model, but considering the amount of memory they require, it is not clear whether realistic architecture would scale well. Even for classical algorithm, it is not clear that the complexity $2^{.292b+o(b)}$ can be achieved by a circuit. On the contrary, for the simplest version of the sieve algorithm with complexity $2^{0.415b}$, it seems possible to design an efficient circuit with area = time = $2^{.2075b+o(b)}$.[11] It is plausible that

[11] While this has not been studied in details for now, this was pointed out by Paul Kirchner on a public mailing-list `https://groups.google.com/d/msg/cryptanalytic-algorithms/BoSRLOuHIjM/wAkZQlwRAgAJ`.

the LSH techniques can also be applied in the circuit model to some extent, but these techniques will likely be more costly than in the RAM model, if not by exponential factors, at least by a substantial polynomial factor.

Amount of memory. Even without considering the issues with (quantumly) accessing such large amount of memory, mobilizing the required amount of memory can already be considered completely infeasible for the parameters of NewHope1024. Indeed, the fastest algorithm requires $2^{0.265b+o(b)}$ bits of storage, estimated at 2^{233} bits. It is claimed in [20] that the amount of memory may be reduced down to $2^{.2075b+o(b)}$ without affecting the asymptotic running time, but it may affect it significantly in practice. Moreover, even this amount $2^{.2075b+o(b)}$ of memory, concretely lower bounded by 2^{182} bits for NewHope1024 already exceed the numbers of atoms on earth. For NewHope512, the memory lower bound of 2^{79} bits is comparable to the total amount of data storage available world-wide in 2017 (estimated to 295 exabytes $\approx 2^{71}$ in 2007 [81], and growing at a rate of 58% per year).

We note that some variants of sieving require less memory at the cost of being significantly slower both asymptotically and in practice [16, 79].

MAXDEPTH for quantum computation. While the maximal depth of a quantum computation have a direct impact on the security level of primitives like AES, we note that this may not be the case for the lattice attacks considered here. Indeed, while sieving is subject to Grover accelerations [94, 92], these Grover search are applied to a rather small search space, and many of them may be ran in parallel. In that respect, it seems that setting MAXDEPTH to 2^{64} or even 2^{40} may not affect the efficiency of a quantum attack. For simplicity and conservatism, we prefer to not account for such a limitation on the adversary computational resources.

4.2.7 Failure analysis and attack exploiting failure

For our analysis of the failure rate of NewHope512 and NewHope1024 we follow the approach from [9]. The script in `scripts/failure-1024k8.py` gives a failure rate of less than 2^{-216} for NewHope1024. For our NewHope512 instance we obtain a similar failure rate of less than 2^{-213} as provided in the script `scripts/failure-512k8.py`.

Attacks exploiting failure have been studied in [58] against a CPA version of NewHope, and required generating about 4000 decryption requests. The attack consist of using much larger errors than defined by the protocol.

One may fear that an attacker using Grover search could produce a failing ciphertext in time about $2^{-216/2}$ for the CCA versions of our scheme. Yet, this would require the adversary to decide offline whether a ciphertext triggers failure. This is not possible, since triggering failure also involves the randomness of the decryptor's secret. Moreover, the failure rate given above are upper bounds, that we do not expect to be so tight. In conclusion, we do not expect decapsulation failures to induce any weaknesses.

5 Expected security strength

In the light of the analysis of Section 4.2 we estimate the following security levels for the two versions of our scheme, according to the 1 to 5 scale provided in Section 4.A.5 (Security Strength Categories) of the call provided by the NIST:

- NewHope512: Level 1 (equivalent to AES128, *i.e.* 2^{170}/MAXDEPTH quantum gates or 2^{143} classical gates) with a claimed post-quantum bit-security of 101 bits.
- NewHope1024: Level 5 (equivalent to AES256, *i.e.* 2^{298}/MAXDEPTH quantum gates or 2^{272} classical gates) with a claimed post-quantum bit-security of 233 bits.

The above claims are meant for *any* value of MAXDEPTH $\geq 2^{40}$. Indeed, we note that this level are easier to achieve as MAXDEPTH increase (since the security of AES decrease as MAXDEPTH increase, while MAXDEPTH does not affect the security analysis of our scheme).

In more details, the cost given in Table 13 following the methodology of [7] are directly corroborating these security strength, despite optimistic cost of Sieving, and counting CPU cycles rather than gates. The much more conservative lower bounds of Table 12 remain somewhat below the gatecounts associated to these level. Yet, in the lights of the margins discussed in Section 4.2.6, this security strength should still

be comfortably conservative. In particular, in unlike attacks against AES, the fastest attacks against our scheme resort to very large amounts of memory (at least 2^{79} bits for NewHope512, and at least 2^{182} bits for NewHope1024), which makes a direct comparison of gatecounts less relevant.

6 Advantages and limitations

6.1 Summary

From our point of view, NewHope is a fast, efficient, and simple scheme that is a suitable replacement of RSA and ECC. The main advantages of NewHope and our parameter choices are:

- **High performance.** NewHope has been implemented on a wide range of platforms and showed very good performance and features reasonable sized key and ciphertexts. Even for a category 5 scheme with 233 bits of security, performance seems to be similar to currently used elliptic curve based cryptosystems.
- **Simplicity and ease of implementation.** A basic NewHope implementation is very simple and can be done with only few lines of code in a tool like SageMath or other mathematical software. The complexity of the final reference implementation mostly stems from encoding and decoding functions that are unavoidable as well as the particular NTT implementation. Additionally, the difference between parameter sets is kept minimal as only n and γ change between NewHope512 and NewHope1024. Moreover, the NewHope-Usenix code has already been ported into various programming languages[12]. A successful integration of NewHope-Usenix into Google Chrome [95] and OpenSSL/Apache [30] shows the suitability for usage in a hybrid setup.
- **Memory efficiency.** The implicit usage of the NTT allows for memory efficient in place computation. No big temporary data structures are required.
- **Conservative design.** We claim that NewHope1024 has a considerable security margin and is based on a conservative security analysis that leaves room for improvements in cryptanalysis. Moreover, the scheme is designed to be somewhat misuse resistant: for example, the leakage of information from the system random number generator more difficult because we always hash random coins before using them.
- **Implementation security.** While more effort on implementation security is needed, some works already exist that deal with lattice-based cryptography and schemes similar to NewHope.

Some of our design choices lead to certain trade-off and we came to the conclusion we accept some disadvantages in our design due to the benefits we gain in other areas (e.g., speed, performance, simplicity). The disadvantages of NewHope we would like to point out are:

- **Small noise distribution.** The choice of $k = 8$ was made as a tradeoff for both parameter sets, to achieve negligible decryption error rates, and to simplify sampling as we can access the randomness byte-wise. However, some security could be gained by optimizing k for $n = 512$ and $n = 1024$ and for more security in an ephemeral Diffie-Hellman variant where correctness is less important (like the original NewHope-Usenix).
- **Ring-LWE**. The usage of the Ring-LWE problem is the basis for the good performance and simplicity of NewHope and currently no attacks are known that can exploit the addition structure. However, the standard LWE assumption could be considered more conservative and thus a better choice in case the next years lead to progress in the cryptanalysis of RLWE.
- **Limited Parametrization**. It is hard with the current structure of NewHope to construct a scheme that achieves NIST security category 2,3, or 4 as either ring dimension $n = 512$ or $n = 1024$ has to be used.
- **Restrictions due to usage of the NTT.** We use the NTT in our basic CPA-secure scheme for efficiency reasons and we output elements in the NTT domain. Past research shows that the NTT is a very suitable way to implement polynomial multiplication on various platforms, especially for large dimensions n. However, this design choice also somewhat restricts the implementer from choosing a polynomial multiplication algorithm of their choice, like Nussbaumer, Karatsuba, or Schoolbook multiplication, or at least leads to a performance impact of doing so. If a different polynomial

[12] See https://ianix.com/pqcrypto/pqcrypto-deployment.html.

multiplication algorithm is used, it is still required to transform elements into the NTT domain with our exact parameters.

6.2 Compatibility with existing deployments and hybrid schemes

The original IND-CPA-secure key encapsulation mechanism NewHope-Usenix has been demonstrated as suitable for use in existing network protocol deployments and in hybrid schemes by several works.

An experiment conducted by Google [95] used NewHope-Usenix alongside ephemeral elliptic curve Diffie–Hellman (ECDH) key exchange in a hybrid TLS 1.2 ciphersuite in an experimental version of the Chrome browser. In their report at the end of a 4 month experiment, Google engineers reported that they "did not find any unexpected impediment to deploying something like NewHope. There were no reported problems caused by enabling it." They further elaborated that "the median connection latency only increased by a millisecond, the latency for the slowest 5% increased by 20ms and, for the slowest 1%, by 150ms", and speculated the the latency increase was due primarily to increased communication sizes, not computational overhead due to the low computational cost of NewHope-Usenix.

Bos et al. [30] compared the performance of several post-quantum key exchange methods, including NewHope-Usenix, within TLS 1.2 using OpenSSL and Apache, measuring latency and throughput of HTTPS connections using either only post-quantum key exchange or hybrid post-quantum key exchange with ECDH in a local network environment. They found that, despite the larger communication size of NewHope-Usenix, it could support more connections per second and had lower latency than ECDH. Hybrid ECDH + NewHope-Usenix did result in a small decrease in throughput and latency compared to only ECDH connections, but only a 3–5% decrease. For details, see Table 5 of [30].

While the above results were about NewHope-Usenix, NewHope-CCA-KEM should not behave very differently. As noted earlier, the primary difference is that NewHope-CCA-KEM uses key transport to establish its shared secret, rather than reconciliation like in NewHope-Usenix, and that NewHope-CCA-KEM achieves IND-CCA security by using a variant of the Fujisaki–Okamoto transform to reconstruct and check ciphertexts. Communication sizes (ciphertexts) increase by 32 bytes, which should have minimal effect. Computation costs of NewHope-CCA-KEM are higher than NewHope-Usenix, primarily to the re-encryption in the decapsulation operation. NewHope-CCA-KEM's extremely fast performance means the cost of this re-encryption is still quite small.

6.3 Ease of implementation and hardware implementations

Implementations of the RLWE scheme of Lyubashevsky et al. [102] (LPR10) on microcontrollers are given in [100, 124]. Additionally, Infineon has announced the successful implementation of a variant of NewHope on a smart card microcontroller [87].

Several implementations of lattice-based cryptography on reconfigurable hardware have been provided so far. Instantiations of the basic LPR10 scheme on FPGAs are given in [72, 132, 123]. Works that implement NewHope-Usenix on FPGAs are [91] and [115].

6.4 Side-channel resistance

Several works already consider side-channel attacks on lattice-based primitives and the construction of countermeasures. Basic mechanism to protect the NTT and arithmetic of lattice-based schemes can be found in [133]. Works that deal proposed protected implementations of CPA-secure Ring-LWE-based scheme are [131] and [130]. Simple power analysis (SPA) attacks are proposed in [125] and [117]. The first work dealing with side-channel protection of a CCA-secure Ring-LWE-based scheme can be found in [116] (see Table 8 and Section 2.3). They provide a provably first-order secure masking scheme and its non-trivial integration into a CCA2 conversion. An interesting result is that for full protection of the secret key and message in the probing model, a masked noise sampler is required for re-encryption and a first design of corresponding protected binomial sampler is provided. The implementation and measurements were carried out on an ARM Cortex-M4F were experimentally verified by using the common non-specific t-test methodology. The masked CCA2-decryption, with very similar parameters and construction as proposed for NewHope, takes 25,334,493 cycles which is an overhead factor of 5.7 compared to the CCA2-secure

decryption without masking. Thus decryption requires roughly to 152 milliseconds runtime at 168 MHz. The overhead cost for the masking of the CCA2-secure decryption is mainly due to the high cost of the sampling. The sampling in turn heavily depends on the performance of the PRNG, in this case SHAKE128. An insecure approach with an unmasked re-encryption would require around 2 million cycles only. However, such an implementation would not provide sufficient protection against a side-channel adversary in a chosen-ciphertext scenario. Due to the high similarity of [116] and NewHope we expect very similar results for a side-channel secured implementation of our proposal.

References

[1] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Z. Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 5–17. ACM, 2015. https://weakdh.org/.

[2] C. Aguilar-Melchor, J. Barrier, L. Fousse, and M.-O. Killijian. Xpir: Private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies*, 2016(2):155–174, 2016. URL: https://eprint.iacr.org/2014/1025.

[3] M. Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions (extended abstract). In J. S. Vitter, editor, *Proceedings of the 30th ACM Symposium on the Theory of Computing*, pages 10–19. ACM, 1998. doi:10.1145/276698.276705.

[4] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In J. S. Vitter, P. G. Spirakis, and M. Yannakakis, editors, *Proceedings of the 33rd ACM Symposium on Theory of Computing*, pages 601–610. ACM, 2001. doi:10.1145/380752.380857.

[5] M. Albrecht, S. Bai, and L. Ducas. A subfield lattice attack on overstretched ntru assumptions. In *Advances in Cryptology – CRYPTO 2016*, volume 9814 of *LNCS*, pages 153–178. Springer, 2016. URL: https://eprint.iacr.org/2016/127.

[6] M. R. Albrecht, F. Göpfert, F. Virdia, and T. Wunderer. Revisiting the expected cost of solving usvp and applications to LWE. In *Advances in Cryptology – ASIACRYPT 2017*. Springer, 2017. To appear.

[7] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015. URL: http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml.

[8] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. NewHope without reconciliation. Cryptology ePrint Archive, Report 2016/1157, 2016. URL: https://eprint.iacr.org/2016/1157.

[9] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A new hope. In T. Holz and S. Savage, editors, *Proceedings of the 25th USENIX Security Symposium*, pages 327–343. USENIX Association, 2016. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/alkim.

[10] E. Alkim, P. Jakubeit, and P. Schwabe. NewHope on ARM Cortex-M. In C. Carlet, M. A. Hasan, and V. Saraswat, editors, *Proceedings of the 6th International Conference on Security, Privacy, and Applied Cryptography Engineering (SPACE)*, volume 10076 of *LNCS*, pages 332–349. Springer, 2016. doi:10.1007/978-3-319-49445-6_19.

[11] Y. Aono and P. Q. Nguyen. Random sampling revisited: lattice enumeration with discrete pruning. In *Advances in Cryptology – EUROCRYPT 2017*, volume 10211 of *LNCS*, pages 65–102. Springer, 2017. doi:10.1007/978-3-319-56614-6_3.

[12] Y. Aono, Y. Wang, T. Hayashi, and T. Takagi. Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. In M. Fischlin and J. Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, volume 9665 of *LNCS*, pages 789–819. Springer, 2016. doi:10.1007/978-3-662-49890-3_30.

[13] B. Applebaum, D. Cash, C. Peikert, and A. Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *LNCS*, pages 595–618. Springer, 2009. doi:10.1007/978-3-642-03356-8_35.

[14] S. Arora and R. Ge. New algorithms for learning in presence of errors. In L. Aceto, M. Henzingeri, and J. Sgall, editors, *Automata, Languages and Programming*, volume 6755 of *LNCS*, pages 403–415. Springer, 2011. https://www.cs.duke.edu/~rongge/LPSN.pdf.

[15] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986. http://www.csie.nuk.edu.tw/~cychen/Lattices/On%20lovasz%20lattice%20reduction%20and%20the%20nearest%20lattice%20point%20problem.pdf.

[16] S. Bai, T. Laarhoven, and D. Stehlé. Tuple lattice sieving. *LMS Journal of Computation and Mathematics*, 19(A):146–162, 2016.

[17] S. Bai, A. Langlois, T. Lepoint, D. Stehlé, and R. Steinfeld. Improved security proofs in lattice-based cryptography: using the Rényi divergence rather than the statistical distance. In T. Iwata and J. H. Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015*, volume 9452 of *LNCS*, pages 3–24. Springer, 2015. doi:10.1007/978-3-662-48797-6_1.

[18] A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 719–737. Springer, 2012. doi:10.1007/978-3-642-29011-4_42.

[19] P. Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In A. M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *LNCS*, pages 311–323. Springer, 1987.

[20] A. Becker, L. Ducas, N. Gama, and T. Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the 27th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2016.

[21] D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Proceedings of the 9th International Conference on Public Key Cryptography (PKC)*, volume 3958 of *LNCS*, pages 207–228. Springer, 2006. doi:10.1007/11745853_14.

[22] D. J. Bernstein, T. Chou, C. Chuengsatiansup, A. Hülsing, T. Lange, R. Niederhagen, and C. van Vredendaal. How to manipulate curve standards: a white paper for the black hat. Cryptology ePrint Archive report 2014/571, 2014. URL: https://eprint.iacr.org/2014/571/.

[23] D. J. Bernstein, C. Chuengsatiansup, T. Lange, and P. Schwabe. Kummer strikes back: new DH speed records. In T. Iwata and P. Sarkar, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 8873 of *LNCS*, pages 317–337. Springer, 2014. Full version: http://cryptojedi.org/papers/#kummer.

[24] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn. SPHINCS: practical stateless hash-based signatures. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 368–397. Springer, 2015. URL: https://cryptojedi.org/papers/#sphincs.

[25] D. J. Bernstein and T. Lange. eBACS: ECRYPT benchmarking of cryptographic systems. URL: http://bench.cr.yp.to.

[26] D. J. Bernstein, P. Schwabe, and G. V. Assche. Tweetable FIPS 202, 2015. URL: http://keccak.noekeon.org/tweetfips202.html.

[27] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Keccak. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 313–314. Springer, 2013.

[28] J.-F. Biasse and F. Song. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In *Proceedings of the 27th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 893–902. SIAM, 2016.

[29] A. Blum, M. L. Furst, M. J. Kearns, and R. J. Lipton. Cryptographic primitives based on hard learning problems. In D. R. Stinson, editor, *Advances in Cryptology – CRYPTO*, volume 773 of *LNCS*, pages 278–291. Springer, 1993. doi:10.1007/3-540-48329-2_24.

[30] J. W. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1006–1018, 2016. doi:10.1145/2976749.2978425.

[31] J. W. Bos, C. Costello, M. Naehrig, and D. 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. URL: https://eprint.iacr.org/2014/599.

[32] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé. CRYSTALS - Kyber: a CCA-secure module-lattice-based KEM. Cryptology ePrint Archive, report 2017/634, 2017. URL: https://eprint.iacr.org/2017/634.

[33] J. W. Bos, K. E. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In M. Stam, editor, *Proceedings of the 14th IMA International Conference on Cryptography and Coding*, volume 8308 of *LNCS*, pages 45–64. Springer, 2013. doi:10.1007/978-3-642-45239-0_4.

[34] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls. Efficient helper data key extractor on FPGAs. In *Proceedings of the 10th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 5154 of *LNCS*, pages 181–197, 2008. doi:10.1007/978-3-540-85053-3_12.

[35] Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé. Classical hardness of learning with errors. In *Proceedings of the 45th ACM Symposium on Theory of Computing (STOC)*, pages 575–584. ACM, 2013. URL: http://arxiv.org/pdf/1306.0281.

[36] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. Cryptology ePrint Archive, report 2011/344, 2011. Preprint of [37]. URL: https://eprint.iacr.org/2011/344.

[37] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM Journal on Computing*, 43(2):831–871, 2014.

[38] D. Cadé, X. Pujol, and D. Stehlé. fplll 4.0.4, 2013. URL: https://github.com/dstehle/fplll.

[39] P. Campbell, M. Groves, and D. Shepherd. Soliloquy: A cautionary tale. In *ETSI 2nd Quantum-Safe Crypto Workshop*, pages 1–9, 2014.

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

[41] Y. Chen and P. Q. Nguyen. BKZ 2.0: Better lattice security estimates. In D. H. Lee and X. 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.

[42] E. Chu and A. George. *Inside the FFT Black Box Serial and Parallel Fast Fourier Transform Algorithms*. CRC Press, Boca Raton, FL, USA, 2000.

[43] J. Conway and N. J. A. Sloane. *Sphere packings, lattices and groups*, volume 290 of *Grundlehren der mathematischen Wissenschaften*. Springer Science & Business Media, 3rd edition, 1999.

[44] R. Cramer, L. Ducas, C. Peikert, and O. Regev. Recovering short generators of principal ideals in cyclotomic rings. In *Advances in Cryptology – EUROCRYPT 2016*, volume 9666 of *LNCS*, pages 559–585. Springer, 2016. doi:10.1007/978-3-662-49896-5_20.

[45] R. Cramer, L. Ducas, and B. Wesolowski. Short Stickelberger class relations and application to Ideal-SVP. In *Advances in Cryptology – EUROCRYPT 2017*, volume 10210 of *LNCS*, pages 324–348. Springer, 2017. doi:10.1007/978-3-319-56620-7_12.

[46] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1):167–226, 2003. doi:10.1137/S0097539702403773.

[47] R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede. Efficient software implementation of ring-LWE encryption. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2015*, pages 339–344. EDA Consortium, 2015. URL: https://eprint.iacr.org/2014/725.

[48] J. Ding. New cryptographic constructions using generalized learning with errors problem. Cryptology ePrint Archive report 2012/387, 2012. URL: https://eprint.iacr.org/2012/387.

[49] J. Ding and X. Lin. A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive report 2012/688, versions 20130303:142425 and 20130303:142813, 2013. URL: https://eprint.iacr.org/2012/688.

[50] J. Ding, X. Xie, and X. Lin. A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive report 2012/688, 2012. URL: https://eprint.iacr.org/2012/688.

[51] J. Ding, X. Xie, and X. Lin. A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive report 2012/688, version 20140729:180116, 2013. URL: https://eprint.iacr.org/2012/688.

[52] Y. Dodis, L. Reyzin, and A. D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540. Springer, 2004. doi:10.1007/978-3-540-24676-3_31.

[53] L. Ducas. Shortest vector from lattice sieving: a few dimensions for free. Cryptology ePrint Archive, Report 2017/999, 2017. https://eprint.iacr.org/2017/999.

[54] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. Lattice signatures and bimodal Gaussians. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, volume 8042 of *LNCS*, pages 40–56. Springer, 2013. https://eprint.iacr.org/2013/383/.

[55] M. Düll, B. Haase, G. Hinterwälder, M. Hutter, C. Paar, A. H. Sánchez, and P. Schwabe. High-speed Curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers. *Designs, Codes and Cryptography*, 77(2-3):493–514, 2015. URL: https://doi.org/10.1007/s10623-015-0087-1, doi:10.1007/s10623-015-0087-1.

[56] K. Eisenträger, S. Hallgren, A. Kitaev, and F. Song. A quantum algorithm for computing the unit group of an arbitrary degree number field. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC)*, pages 293–302. ACM, 2014.

[57] R. Fitzpatrick, C. Bischof, J. Buchmann, Ö. Dagdelen, F. Göpfert, A. Mariano, and B.-Y. Yang. Tuning GaussSieve for speed. In *Progress in Cryptology – LATINCRYPT 2014*, volume 8895 of *LNCS*, pages 288–305. Springer, 2014. doi:10.1007/978-3-319-16295-9_16.

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

[59] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology – CRYPTO 1999*, volume 1666 of *LNCS*, pages 537–554. Springer, 1999. doi:10.1007/3-540-48405-1_34.

[60] M. Fukase and K. Kashiwabara. An accelerated algorithm for solving SVP based on statistical analysis. *Journal of Information Processing*, 23(1):67–80, 2015. URL: https://www.jstage.jst.go.jp/article/ipsjjip/23/1/23_67/_article/-char/ja/.

[61] P. Gaborit. Noisy Diffie-Hellman protocols. Slides of a talk in the recent results session at *Post-Quantum Cryptography - 3rd International Workshop, PQCrypto 2010*, 2010. https://pqc2010.cased.de/rr/03.pdf.

[62] S. D. Galbraith. Space-efficient variants of cryptosystems based on learning with errors, 2013. https://www.math.auckland.ac.nz/~sgal018/compact-LWE.pdf.

[63] N. Gama and P. Nguyen. Predicting lattice reduction. In *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 31–51. Springer, 2008. doi:10.1007/978-3-540-78967-3_3.

[64] N. Gama and P. Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In C. Dwork, editor, *Proceedings of the 40th ACM Symposium on Theory of Computing (STOC)*, pages 207–216. ACM, 2008. doi:10.1145/1374376.1374408.

[65] N. Gama, P. Q. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 257–278. Springer, 2010. doi:10.1007/978-3-642-13190-5_13.

[66] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, 2013. doi:10.1007/978-3-642-38348-9_1.

[67] W. M. Gentleman and G. Sande. Fast Fourier transforms: for fun and profit. In *Fall Joint Computer Conference*, volume 29 of *AFIPS Proceedings*, pages 563–578, 1966. http://cis.rit.edu/class/simg716/FFT_Fun_Profit.pdf.

[68] C. Gentry, S. Halevi, and V. Vaikuntanathan. A simple BGN-type cryptosystem from LWE. In *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 506–522. Springer, 2010. doi:10.1007/978-3-642-13190-5_26.

[69] C. Gentry, C. Peikert, and V. Vaikuntanathan. How to use a short basis: Trapdoors for hard lattices and new cryptographic constructions, 2008. http://web.eecs.umich.edu/~cpeikert/pubs/trap_lattice.pdf (full version of [70]).

[70] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In C. Dwork, editor, *Proceedings of the 40th ACM Symposium on Theory of Computing (STOC)*, pages 197–206. ACM, 2008. doi:10.1145/1374376.1374407.

[71] C. Gentry and M. Szydlo. Cryptanalysis of the revised NTRU signature scheme. In L. R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 299–320. Springer, 2002. doi:10.1007/3-540-46035-7_20.

[72] N. Göttert, T. Feller, M. Schneider, J. A. Buchmann, and S. A. Huss. On the design of hardware building blocks for modern lattice-based encryption schemes. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *LNCS*, pages 512–529. Springer, 2012. doi:10.1007/978-3-642-33027-8_30.

[73] M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt. Applying Grover's algorithm to AES: quantum resource estimates. In *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, volume 9606 of *LNCS*, pages 29–43. Springer, 2016. doi:10.1007/978-3-319-29360-8_3.

[74] S. Gueron. Parallelized hashing via j-lanes and j-pointers tree modes, with applications to SHA-256. Cryptology ePrint Archive report 2014/170, 2014. https://eprint.iacr.org/2014/170.

[75] T. Güneysu, V. Lyubashevsky, and T. Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *LNCS*, pages 530–547. Springer, 2012. doi:10.1007/978-3-642-33027-8_31.

[76] T. Güneysu, T. Oder, T. Pöppelmann, and P. Schwabe. Software speed records for lattice-based signatures. In P. Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*, volume 7932 of *LNCS*, pages 67–82. Springer, 2013. http://cryptojedi.org/papers/#lattisigns.

[77] G. Hanrot, X. Pujol, and D. Stehlé. Algorithms for the shortest and closest lattice vector problems. In Y. M. Chee, Z. Guo, S. Ling, F. Shao, Y. Tang, H. Wang, and C. Xing, editors, *Coding and Cryptology – Third International Workshop, IWCC 2011*, volume 6639 of *LNCS*, pages 159–190. Springer, 2011. doi:10.1007/978-3-642-20901-7_10.

[78] G. Hanrot, X. Pujol, and D. Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *LNCS*, pages 447–464. Springer, 2011. URL: https://eprint.iacr.org/2011/198.pdf.

[79] G. Herold and E. Kirshanova. Improved algorithms for the approximate k-list problem in euclidean norm. In *20th International Conference on Public Key Cryptography (PKC)*, volume 10174 of *LNCS*, pages 16–40. Springer, 2017. URL: https://eprint.iacr.org/2017/017.pdf.

[80] A. V. Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A. Sadeghi, I. Verbauwhede, and C. Wachsmann. Reverse fuzzy extractors: Enabling lightweight mutual authentication for PUF-enabled RFIDs. In A. D. Keromytis, editor, *16th International Conference on Financial Cryptography and Data Security*, volume 7397 of *LNCS*, pages 374–389. Springer, 2012. doi:10.1007/978-3-642-32946-3_27.

[81] M. Hilbert and P. López. The world's technological capacity to store, communicate, and compute information. *Science*, 332(6025):60–65, 2011.

[82] J. Hoffstein, J. Pipher, J. M. Schanck, J. H. Silverman, W. Whyte, and Z. Zhang. Choosing parameters for NTRUEncrypt. Cryptology ePrint Archive report 2015/708, 2015. URL: https://eprint.iacr.org/2015/708.

[83] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: a ring-based public key cryptosystem. In J. P. Buhler, editor, *Algorithmic number theory Symposium (ANTS)*, volume 1423 of *LNCS*, pages 267–288. Springer, 1998. https://www.securityinnovation.com/uploads/Crypto/ANTS97.ps.gz.

[84] J. Hoffstein, J. Pipher, and J. H. Silverman. *An introduction to mathematical cryptography*. Springer Verlag, 2008.

[85] D. Hofheinz, K. Hövelmanns, and E. Kiltz. A modular analysis of the Fujisaki–Okamoto transformation. In *Theory of Cryptography – 15th International Conference, TCC 2017*, volume 10677 of *LNCS*, pages 341–371. Springer, 2017. URL: https://eprint.iacr.org/2017/604.

[86] N. Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In A. Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *LNCS*, pages 150–169. Springer, 2007. doi:10.1007/978-3-540-74143-5_9.

[87] Infineon Technologies. Ready for tomorrow: Infineon demonstrates first post-quantum cryptography on a contactless security chip. Press release, May 2017. URL: https://www.infineon.com/cms/en/about-infineon/press/press-releases/2017/INFCCS201705-056.html.

[88] Z. Jin and Y. Zhao. Optimal key consensus in presence of noise. CoRR, report abs/1611.06150, 2016. URL: http://arxiv.org/abs/1611.06150.

[89] R. Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 193–206, 1983. doi:10.1145/800061.808749.

[90] P. Kirchner and P.-A. Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In R. Gennaro and M. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, volume 9215 of *LNCS*, pages 43–62. Springer, 2015. doi:10.1007/978-3-662-47989-6_3.

[91] P. Kuo, W. Li, Y. Chen, Y. Hsu, B. Peng, C. Cheng, and B. Yang. Post-quantum key exchange on FPGAs. Cryptology ePrint Archive, report 2017/690, 2017. URL: https://eprint.iacr.org/2017/690.

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

[93] T. Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In R. Gennaro and M. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, volume 9216 of *LNCS*, pages 3–22. Springer, 2015. doi:10.1007/978-3-662-47989-6_1.

[94] T. Laarhoven, M. Mosca, and J. van de Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77(2):375–400, 2015. URL: https://eprint.iacr.org/2014/907/.

[95] A. Langley. CECPQ1 results. Blog post on imperialviolet.org, 2016. URL: https://www.imperialviolet.org/2016/11/28/cecpq1.html.

[96] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982. doi:10.1007/bf01457454.

[97] H. W. Lenstra. Euclid's algorithm in cyclotomic fields. *Journal of the London Mathematical Society*, 2(4):457–465, 1975. URL: https://openaccess.leidenuniv.nl/bitstream/handle/1887/2121/346_016.pdf.

[98] H. W. Lenstra and A. Silverberg. Revisiting the Gentry-Szydlo algorithm. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, volume 8616 of *LNCS*, pages 280–296. Springer, 2014. https://eprint.iacr.org/2014/430.

[99] R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. In A. Kiayias, editor, *Topics in Cryptology - CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, 2011. https://eprint.iacr.org/2010/613/.

[100] Z. Liu, H. Seo, S. S. Roy, J. Großschädl, H. Kim, and I. Verbauwhede. Efficient Ring-LWE encryption on 8-bit AVR processors. In T. Güneysu and H. Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015*, volume 9293 of *LNCS*, pages 663–682. Springer, 2015. https://eprint.iacr.org/2015/410/.

[101] V. Lyubashevsky. Lattice signatures without trapdoors. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755. Springer, 2012. https://eprint.iacr.org/2011/537/.

[102] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, 2010. http://www.di.ens.fr/~lyubash/papers/ringLWE.pdf.

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

[104] V. Lyubashevsky, C. Peikert, and O. Regev. A toolkit for ring-LWE cryptography. In *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, 2013.

[105] A. Mariano, T. Laarhoven, and C. Bischof. A parallel variant of ldsieve for the svp on lattices. In *Parallel, Distributed and Network-based Processing (PDP), 2017 25th Euromicro International Conference on*, pages 23–30. IEEE, 2017. URL: https://eprint.iacr.org/2016/890.pdf.

[106] D. Micciancio. CSE206A: Lattices algorithms and applications (spring 2014), 2014. Lecture notes of a course given in UCSD. See http://cseweb.ucsd.edu/classes/sp14/cse206A-a/.

[107] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems: a cryptographic perspective*, volume 671 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, 2002.

[108] D. Micciancio and O. Regev. Lattice-based cryptography. In D. J. Bernstein, J. Buchmann, and E. Dahmen, editors, *Post-quantum Cryptography*, pages 147–191. Springer, 2009.

[109] D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1468–1480. SIAM, 2010. http://dl.acm.org/citation.cfm?id=1873720.

[110] A. Montanaro. Quantum walk speedup of backtracking algorithms. arXiv preprint arXiv:1509.02374, 2015. http://arxiv.org/pdf/1509.02374v2.

[111] P. 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.

[112] 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.

[113] P. Q. Nguyen and B. Valle. *The LLL Algorithm: Survey and Applications*. Springer, 1st edition, 2009.

[114] P. Q. Nguyen and T. 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.

[115] T. Oder and T. Güneysu. Implementing the NewHope-Simple key exchange on low-cost FPGAs. In *Progress in Cryptology – LATINCRYPT 2017*, 2017. To appear. URL: http://www.cs.haifa.ac.il/~orrd/LC17/paper51.pdf.

[116] T. Oder, T. Schneider, T. Pöppelmann, and T. Güneysu. Practical CCA2-secure and masked Ring-LWE implementation. Cryptology ePrint Archive, report 2016/1109, 2016. URL: https://eprint.iacr.org/2016/1109.

[117] A. Park and D. Han. Chosen ciphertext simple power analysis on software 8-bit implementation of ring-LWE encryption. In *2016 IEEE Asian Hardware-Oriented Security and Trust, (AsianHOST)*, pages 1–6. IEEE Computer Society, 2016. doi:10.1109/AsianHOST.2016.7835555.

[118] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *Proceedings of the 41st ACM Symposium on Theory of Computing (STOC)*, pages 333–342, 2009. doi:10.1145/1536414.1536461.

[119] C. Peikert. Some recent progress in lattice-based cryptography. Slides of an invited tutorial at the 6th IACR Theory of Cryptography Conference – TCC 2009, 2009. URL: http://www.cc.gatech.edu/fac/cpeikert/pubs/slides-tcc09.pdf.

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

[121] C. Peikert, O. Regev, and N. Stephens-Davidowitz. Pseudorandomness of ring-LWE for any ring and modulus. In *Proceedings of the 49th ACM Symposium on Theory of Computing (STOC)*, pages 461–473. ACM, 2017. URL: https://eprint.iacr.org/2017/258.

[122] T. Pöppelmann, L. Ducas, and T. Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In L. Batina and M. Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *LNCS*, pages 353–370. Springer, 2014. https://eprint.iacr.org/2014/254/.

[123] T. Pöppelmann and T. Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. In T. Lange, K. Lauter, and P. 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.

[124] T. Pöppelmann, T. Oder, and T. Güneysu. High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers. In K. E. Lauter and F. Rodríguez-Henríquez, editors, *Progress in Cryptology – LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 346–365. Springer, 2015. doi:10.1007/978-3-319-22174-8_19.

[125] R. Primas, P. Pessl, and S. Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In W. Fischer and N. Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017*, volume 10529 of *LNCS*, pages 513–533. Springer, 2017. doi:10.1007/978-3-319-66787-4_25.

[126] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In H. N. Gabow and R. Fagin, editors, *Proceedings of the 37th ACM Symposium on Theory of Computing (STOC)*, pages 84–93. ACM, 2005. doi:10.1145/1060590.1060603.

[127] O. 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.

[128] O. Regev. The learning with errors problem (invited survey). In *Proceedings of the 25th Annual IEEE Conference on Computational Complexity - CCC 2010*, pages 191–204. IEEE Computer Society, 2010. doi:10.1109/CCC.2010.26.

[129] A. Rényi. On measures of entropy and information. In *Fourth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 547–561, 1961. http://projecteuclid.org/euclid.bsmsp/1200512181.

[130] O. Reparaz, R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede. Additively homomorphic Ring-LWE masking. In *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, volume 9606 of *LNCS*, pages 233–244, 2016. doi:10.1007/978-3-319-29360-8_15.

[131] O. Reparaz, S. S. Roy, F. Vercauteren, and I. Verbauwhede. A masked Ring-LWE implementation. In T. Güneysu and H. Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop*, volume 9293 of *LNCS*, pages 683–702. Springer, 2015. doi:10.1007/978-3-662-48324-4_34.

[132] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede. Compact Ring-LWE cryptoprocessor. In L. Batina and M. Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *LNCS*, pages 371–391. Springer, 2014. https://eprint.iacr.org/2013/866/.

[133] M. O. Saarinen. Arithmetic coding and blinding countermeasures for Ring-LWE. Cryptology ePrint Archive, report 2016/276, 2016. URL: https://eprint.iacr.org/2016/276.

[134] C. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994. doi:10.1007/BF01581144.

[135] C.-P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical computer science*, 53(2):201–224, 1987. URL: http://dx.doi.org/10.1016/0304-3975(87)90064-8.

[136] C.-P. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, 1994. doi:10.1007/BF01581144.

[137] D. Stehlé and R. Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In K. G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 27–47. Springer, 2011. doi:10.1007/978-3-642-20465-4_4.

[138] D. Stehlé, R. Steinfeld, K. Tanaka, and K. Xagawa. Efficient public key encryption based on ideal lattices. In *Advances in Cryptology - ASIACRYPT 2009*, volume 5912, pages 617–635. Springer, 2009. URL: https://eprint.iacr.org/2009/285.

[139] E. E. Targhi and D. Unruh. Post-quantum security of the Fujisaki–Okamoto and OAEP transforms. In M. Hirt and A. D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B*, volume 9986 of *LNCS*, pages 192–216, 2016. doi:10.1007/978-3-662-53644-5_8.

[140] P. van Emde Boas. Another NP-complete partition problem and the complexity of computing short vectors in a lattice. Technical Report 81-04, Universiteit van Amsterdam. Mathematisch Instituut, 1981.

[141] K. Xagawa. *Cryptography with Lattices*. PhD thesis, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2010. http://xagawa.net/pdf/2010Thesis.pdf.