SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Post-Quantum Lattice-Based Key Encapsulation Mechanisms

av

Jennifer Chamberlain

2019 - No M2

Post-Quantum Lattice-Based Key Encapsulation Mechanisms

Jennifer Chamberlain

Självständigt arbete i matematik 30 högskolepoäng, avancerad nivå

Handledare: Jonas Bergström, John Mattson

2019

Abstract

Lately there has been increased interest in post-quantum cryptography, and NIST is in the process of standardizing one or more quantum-resistant cryptosystems. Among the many submissions to their call for proposals, lattice-based cryptosystems are popular, and this thesis looks at a number of the lattice problems these cryptosystems can be based on, with particular focus on different versions of the learning with errors problem (LWE). I give an overview and comparison of three key encapsulation mechanisms (KEMs) based on different versions of this problem (FrodoKEM, NewHope and CRYSTALS-Kyber), and I also adapt CRYSTALS-Kyber, which is based on the module version of LWE (MLWE), to use only the module learning with rounding problem (MLWR), which makes the system more efficient, and discuss how this change affects the security of the system.

Acknowledgements

I want to thank my supervisors Jonas Bergström at SU and John Mattsson at Ericsson, as well as Erik Thormarker, also of Ericsson, for their support and ideas throughout my work on this thesis. I also want to thank my referee Sven Raum for quick and useful feedback.

Contents

1 Introduction

Since ancient times, cryptography has been used to send messages in such a way that even if intercepted, they cannot be read by anyone but the intended receiver. One way to achieve this is to have some secret key known to both the sender and receiver, with which the message can be encrypted to a ciphertext, from which the message can only be recovered with the secret key. This is called symmetric cryptography, and one example of this is the one-time pad cipher, which is impossible to crack but requires single-use keys of the same size as the message. In general, symmetric encryption relies on both parties having access to a secret key that no one else knows (it cannot simply be sent over an open channel, as it might then be intercepted), and this becomes especially inconvenient when it comes to communication over the internet.

An alternative that has only been around since the 1970s is asymmetric cryptography, where each party has a public key and a private key. The public key is published, and anyone can use it to, for instance, encrypt a message to a ciphertext from which the message can only be recovered using the private key. This is called a public key encryption scheme, or PKE. Asymmetric cryptography is also used for key exchanges, which are used to agree on a key that is shared between two parties while keeping it secret from any potential eavesdroppers, and for signature schemes, which are used to send a message along with a token that the receiver can use to confirm that the message came from the correct sender.

A public key encryption scheme is usually not as fast or as memory efficient as a symmetric scheme, but it has the advantage of requiring only two keys per party, one public and one private, whereas a symmetric scheme requires one key per communication link which is inconvenient for someone who needs to send or receive encrypted information to many others. Therefore if there is a sufficiently efficient public key encryption scheme it might be used for communication, but otherwise asymmetric encryption is used only for the key exchange, and then symmetric encryption is used for sending actual messages.

RSA schemes, whose security relies on the difficulty of factoring large numbers, are especially popular for public key encryption. The most well known key exchanges are Diffie-Hellman type schemes, which rely on the hardness of finding discrete logarithms in finite groups, often elliptic curves. However, things are changing with the advent of quantum computers.

Quantum computers have different capabilities than classical computers, and can query functions on inputs in superposition, evaluating the functions at several positions at once. Such computers are not in general faster than classical computers, but there are some problems which they can solve especially well, and as Shor showed in [35] these include the problems of factoring large numbers and finding discrete logarithms in finite fields. At present, quantum computers actually able to implement Shor's algorithm are not available, but in 2015, Mosca estimated in [27] that by 2031 chances of breaking RSA with 2048-bit

modulus using a quantum computer will be 50%.[1] This means that sensitive information that needs to remain confidential for more than about a decade to come should even now be encrypted in some way less vulnerable to quantum computers.

The National Institute of Standards and Technology (NIST) has started a process to find and standardise schemes that will remain secure against quantum computers, and in 2016 they called for submissions to their "Post-Quantum Cryptography Standardization Project". By the end of 2017, 59 submissions for public key encryption schemes or key encapsulation mechanisms had been received (as well as 23 signature schemes). Key encapsulation mechanisms (KEMs) are another way to exchange a secret key, sometimes constructed from a public key encryption scheme (PKE) by deriving a shared secret key from a message which is sent from one party to the other using a PKE. These are of interest because it is often more efficient to exchange a key with asymmetric encryption and then use symmetric encryption, than to conduct an entire communication with only public key encryption.

A majority of the PKEs and KEMs submitted to NIST are either code-based (using error correcting codes) or lattice-based. Lattice-based schemes have the advantage of comparatively strong security proofs, not in the sense that they can be shown to be entirely unbreakable like for instance a one-time pad cipher but in the sense that breaking lattice-based schemes can (depending on the specific scheme) be shown to be as hard as finding short vectors in *any* lattice, a problem which is believed to be computationally very hard. So far there is no known algorithm with running time t that, in a general lattice, will find a vector that is no more than a factor γ longer than the shortest vector, without either γ of t growing exponentially (or nearly exponentially) in the dimension of the lattice. The disadvantage of lattice based schemes is that they have much larger keys and ciphertexts than for instance elliptic curve cryptography, which so far has not been broken for key sizes of 163 bits or more. As contrast, lattice based schemes may have keys many thousand bytes large. This, however, is not uncommon in post quantum cryptography, and code based schemes tend to have fairly similar sizes, with smaller ciphertexts but larger public keys. There have been several attempts to make lattice-based schemes more efficient, both in running time and in the size of keys and ciphertexts, by restricting them to certain more structured lattices, and while this restriction means that the proofs are likewise restricted to finding short vectors in more structured lattices there are so far no known attacks which make more structured lattice-based schemes unsuitable for public key cryptography.

[1] Matteo Mariantoni said in an invited talk at the 2014 PQCrypto conference that a quantum computed capable of factoring a 2000-bit number in 24 hours might be possible to build by about 2030 (though he said that this was a rough time estimate and would depend, among other things, on the money put into development), that it would require a dedicated nuclear power plant and the cost would be about a billion dollars.

1.1 Report outline

Section 2 contains background on lattices and cryptography, including hard lattice problems and an overview of different variants of the transform by Fujisaki and Okamoto [17] which takes a passively secure PKE and returns an actively secure PKE. Section 3 concerns the *short integer solution* and *learning with errors* problems, whose hardness relies on hard lattice problems and on which encryption (or signature) schemes can be based, and *learning with rounding* which is a variant of the learning with errors problem but without error sampling. In Section 4 we give some idea of three of the lattice-based KEMs submitted to NIST, one of which (Frodo [39]) uses general lattices while the other two (NewHope [40] and Kyber [41]) use more structured lattices. Section 5 contains a suggestion for a version of Kyber that relies on learning with rounding rather than learning with errors, and a discussion of the security of this adapted scheme.

2 Preliminaries

2.1 Notation

$\mathbb{Z}$ denotes the ring of integers, and similarly $\mathbb{Q}, \mathbb{R}, \mathbb{C}$ are the rationals, the reals and the complex numbers respectively. $\mathbb{Z}[X]$ denotes the polynomials in X with integer coefficients. For $f(X) \in \mathbb{Z}[X]$, $(f(X))$ denotes the ideal of $\mathbb{Z}[X]$ generated by $f(X)$. For any ring R and any integer q, $R_q = R/qR$.

Matrices are written in uppercase bold, e.g. $\mathbf{A}$, vectors in lowercase bold. All vectors are column vectors, and the transpose of a vector $\mathbf{a}$ is $\mathbf{a}^T$. Similarly the transpose of a matrix $\mathbf{A}$ is $\mathbf{A}^T$. Matrices and vectors can have entries in any ring.

- The inner product of two vectors $\mathbf{a} = (a_1, ..., a_n)^T$ and $\mathbf{b} = (b_1, ..., b_n)^T$ in $\mathbb{C}^n$ is $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{n} a_i \cdot b_i$.
- The Euclidean norm for a vector $\mathbf{a} = (a_1, ..., a_n)^T \in \mathbb{R}^n$ is defined as $||\mathbf{a}|| = \sqrt{a_1^2 + ... + a_n^2}$.
- For any real number r, $\lfloor r \rfloor$ is the largest integer such that $\lfloor r \rfloor \leq r$, $\lceil r \rceil$ is the smallest integer such that $\lceil r \rceil \geq r$, and rounding to the nearest integer (with ties broken upwards) is written as $\lfloor r \rceil = \lfloor r + 1/2 \rfloor$.
- Componentwise multiplication is denoted $\circ$. For polynomials $a = \sum_{i=1}^{n} a_i X^i$ and $b = \sum_{i=1}^{n} b_i X^i$ with coefficients in some ring, $a \circ b = \sum_{i=1}^{n} a_i b_i X^i$.
- Pr[E] denotes the probability of an event E, and for any E we always have $0 \leq \Pr[E] \leq 1$.
- If χ is some probability distribution, $e \leftarrow \chi$ means that e is sampled according to χ, and $\mathbf{e} \leftarrow \chi^n$ means that $\mathbf{e} = (e_1, ..., e_n)^T$ where $e_i \leftarrow \chi$ for $i = 1, ..., n$.

- For any algorithm A, $g \leftarrow A$ means that g is the output of A.
- In computer algorithms, two strings or tuples are concatenated using $||$, so $(a_1, ..., a_n)||(b_1, ..., b_n) = (a_1, ..., a_n, b_1, ..., b_n)$.

Let $f(x)$ and $g(x)$ be real-valued functions of x, defined on an unbounded set of the positive real numbers, such that $g(x)$ is positive for sufficiently high values of x. We have the following asymptotic notation.

- $f(x) = O(g(x))$: There is N and a positive constant C such that $|f(x)| \leq Cg(x)$ for all $x \geq N$.
- $f(x) = \tilde{O}(g(x))$: For some $k > 0$, $f(x) = O(g(x)\log^k(x))$, i.e., we ignore all logarithmic factors.
- $f(x) = o(g(x))$: For every positive constant ϵ there is N such that $|f(x)| \leq \epsilon g(x)$ for all $x \geq N$.
- $f(x) = \omega(g(x))$: For every positive constant ϵ there is N such that $|f(x)| \geq \epsilon g(x)$ for all $x \geq N$.
- $f(x) = \Theta(g(x))$: There is N and positive constants C_1, C_2 such that $C_1 g(x) \leq |f(x)| \leq C_2 g(x)$ for all $x \geq N$.
- $g(x) = \text{poly}(x)$: $g(x)$ is bounded by a polynomial in x.

2.2 Lattices

Much of the following material on lattices can be found in [21] by Hoffstein, Pipher and Silverman.

Definition 2.1. A *lattice* is a discrete additive subgroup of $\mathbb{R}^n$. The *dimension* of a lattice $\mathcal{L}$ is the maximum size of a set of linearly independent vectors in $\mathcal{L}$.

The dimension of a lattice is sometimes also called the *rank* of the lattice, and a lattice $\mathcal{L} \subset \mathbb{R}^n$ that has dimension n is known as a *full-rank* lattice. In this text, unless otherwise specified, all lattices are assumed to be full-rank.

A lattice $\mathcal{L} \subset \mathbb{R}^n$ always has a *basis*, a set of linearly independent vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ in $\mathbb{R}^n$ such that any vector $\mathbf{w} \in \mathcal{L}$ can be written as a linear combination with integer coefficients of $\mathbf{v}_1, \ldots, \mathbf{v}_n$. Given a basis we can also define the specific lattice generated by that basis.

Definition 2.2. Let $\mathcal{B} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ be a set of linearly independent vectors in $\mathbb{R}^n$. Then $\mathcal{L}(\mathcal{B}) = \{\sum_{i=1}^n a_i \mathbf{v}_i : a_i \in \mathbb{Z} \text{ for all } i\}$ is the *lattice generated by the basis* $\mathcal{B}$.

The basis of a lattice is not unique. If $\mathbf{v}_1, \ldots, \mathbf{v}_n$ is a basis for a lattice $\mathcal{L}$, then any other set of linearly independent vectors in $\mathbb{R}^n$ that generate $\mathcal{L}$ is also a basis for $\mathcal{L}$. A basis for a lattice of dimension n always consists of n basis vectors.

Assume $\mathbf{v}_1, \ldots, \mathbf{v}_n$ and $\mathbf{w}_1, \ldots, \mathbf{w}_n$ are two bases for $\mathcal{L}$. Then there are integers a_{ij} for $i, j \in \{1, ..., n\}$ such that

$$\begin{aligned} \mathbf{w}_1 &= a_{11}\mathbf{v}_1 + a_{12}\mathbf{v}_2 + \cdots + a_{1n}\mathbf{v}_n \\ \mathbf{w}_2 &= a_{21}\mathbf{v}_1 + a_{22}\mathbf{v}_2 + \cdots + a_{2n}\mathbf{v}_n \\ &\vdots \\ \mathbf{w}_n &= a_{n1}\mathbf{v}_1 + a_{n2}\mathbf{v}_2 + \cdots + a_{nn}\mathbf{v}_n, \end{aligned}$$

that is, $(\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_n) = (\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n)\mathbf{A}$ where

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{n1} \\ a_{12} & a_{22} & \cdots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{nn} \end{pmatrix}.$$

Since both $\mathbf{v}_1, \ldots, \mathbf{v}_n$ and $\mathbf{w}_1, \ldots, \mathbf{w}_n$ are bases for $\mathcal{L}$, $\mathbf{A}$ must be invertible, and $\mathbf{A}^{-1}$ must have integer entries. Therefore $\det \mathbf{A}$ and $\det(\mathbf{A}^{-1})$ are both integers, and since

$$1 = \det \mathbf{I} = \det \mathbf{A} \det(\mathbf{A}^{-1})$$

it follows that $\det \mathbf{A} = \pm 1$. This shows that if we have two bases for a lattice, there is some square integer matrix with determinant ± 1 (such a matrix is called *unimodular*, it is invertible and its inverse is also an integer matrix with determinant ± 1), which multiplied with the first basis will produce the second. The converse also holds, for if $\mathbf{v}_1, \ldots, \mathbf{v}_n$ is a basis for the lattice $\mathcal{L}$ and $\mathbf{A}$ is a unimodular $n \times n$ matrix, then $(\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n)\mathbf{A}$ is also a basis for $\mathcal{L}$.

In a vector space, an orthogonal basis can be created from any basis by using the Gram-Schmidt Algorithm. However, given a basis for a lattice, the Gram-Schmidt Algorithm will not in general yield an orthogonal basis for the lattice since the vectors produced by the algorithm are unlikely to belong to the lattice. We can still talk about more or less orthogonal bases, and a reasonably orthogonal basis for a lattice is sometimes called a "good" basis, while a basis that is far from orthogonal is called a "bad" basis. Note that these are vague and relative terms, and while one can sometimes say more specifically what is a "good enough" basis for a particular purpose there is no general definition.

Example. Let $\mathcal{L} \subset \mathbb{R}^2$ be the lattice spanned by $\mathbf{v}_1 = (-351, 122)$ and $\mathbf{v}_2 = (108, 447)$. The two unimodular matrices

$$\mathbf{A} = \begin{pmatrix} 3 & 4 \\ 5 & 7 \end{pmatrix} \text{ and } \mathbf{B} = \begin{pmatrix} 7 & 29 \\ 8 & 33 \end{pmatrix},$$

give two more bases $\mathbf{w}_1, \mathbf{w}_2$ and $\mathbf{u}_1, \mathbf{u}_2$ for $\mathcal{L}$, with

$$\begin{aligned} \mathbf{w}_1 &= 3\mathbf{v}_1 + 5\mathbf{v}_2 = (513, 2601) \\ \mathbf{w}_2 &= 4\mathbf{v}_1 + 7\mathbf{v}_2 = (-648, 3617) \\ \mathbf{u}_1 &= 7\mathbf{v}_1 + 8\mathbf{v}_2 = (-1593, 4430) \\ \mathbf{u}_2 &= 29\mathbf{v}_1 + 33\mathbf{v}_2 = (-6615, 18289). \end{aligned}$$

The angle between $\mathbf{v}_1$ and $\mathbf{v}_2$ is about 84.5 degrees, whereas that between $\mathbf{w}_1$ and $\mathbf{w}_2$ is about 21.3 degrees and that between $\mathbf{u}_1$ and $\mathbf{u}_2$ only about 0.1 degrees, so the vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ are nearly parallell. Probably most would agree that $\mathbf{u}_1, \mathbf{u}_2$ is a "bad" basis and $\mathbf{v}_1, \mathbf{v}_2$ a "good" one, though with no actual definition of these terms these are not objective truths. However, we can certainly say that $\mathbf{w}_1, \mathbf{w}_2$ is a better basis than $\mathbf{u}_1, \mathbf{u}_2$, and $\mathbf{v}_1, \mathbf{v}_2$ is better than either.

Dual lattices. For any lattice $\mathcal{L} \in \mathbb{R}^n$, the dual of $\mathcal{L}$ is

$$\mathcal{L}^* := \{\mathbf{y} \in \mathbb{R}^n \ : \ \langle \mathbf{x}, \mathbf{y} \rangle \in \mathbb{Z} \text{ for all } \mathbf{x} \in \mathcal{L}\}.$$

q-ary lattices. A lattice $\mathcal{L}$ is *q-ary* for some integer q if $q\mathbb{Z}^n \subseteq \mathcal{L} \subseteq \mathbb{Z}^n$. For positive integers q, m, n, and $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, we define two particular q-ary lattices

$$\Lambda_q(\mathbf{A}) = \{\mathbf{y} \in \mathbb{Z}^n \ : \ \mathbf{y} = \mathbf{A}^T\mathbf{s} \bmod q \text{ for some } \mathbf{s} \in \mathbb{Z}^m\}$$
$$\Lambda_q^\perp(\mathbf{A}) = \{\mathbf{y} \in \mathbb{Z}^n \ : \ \mathbf{A}\mathbf{y} = \mathbf{0} \bmod q\}.$$

Ideal and module lattices. Let ξ be an algebraic number, i.e. a complex root of a polynomial in $\mathbb{Q}[X]$, and K the number field $\mathbb{Q}(\xi)$. This is a $\mathbb{Q}$-vector space of dimension n, where n is the degree of the unique monic irreducible polynomial f such that ξ is one of its roots (the *minimal polynomial of f*). An algebraic number whose minimal polynomial is in $\mathbb{Z}[X]$ is an *algebraic integer*. The set of algebraic integers in K form a ring called the ring of integers of K. In lattice cryptography, we tend to consider only cases where ξ is a primitive ν-th root of unity, so that it is a root of the ν-th cyclotomic polynomial Φ_ν and K is a cyclotomic field. Then $n = \phi(\nu)$ (where ϕ is Euler's totient function) and the ring of integers is $\mathbb{Z}[\xi]$.

Let K be a number field and R its ring of integers. There is an embedding σ_H from K to $\mathbb{R}^n$ (it has to do with the canonical embeddings, field homomorphisms $\sigma_j : K \to \mathbb{C}$ defined by $\sigma_j : \xi \mapsto \xi^j$ for $j \in \mathbb{Z}_\nu^\times$, and Langlois and Stehlé write more about it in [25]) and for an ideal I of R, $\sigma_H(I)$ is a lattice called an *ideal lattice*. Similarly, $(\sigma_H, ..., \sigma_H)$ is an embedding from K^d to $\mathbb{R}^{nd}$, and it maps a finitely generated module $M \subseteq K^d$ of R to a lattice called a *module lattice*[2]. Note that the ideal lattice corresponding to the ideal I has dimension n and the module lattice corresponding to the module $M \subseteq K^d$ has dimension nd.

In lattice cryptography, lattice problems over ideal or module lattices are often described using polynomial rings instead, by considering polynomials in rings of the form $R := \mathbb{Z}[X]/f$ for some polynomial $f \in \mathbb{Z}[X]$ of degree n rather than vectors in the corresponding lattice.

[2]Langlois and Stehlé write in Section 2.1 of [25] that because K is a number field R is a Dedekind domain, and therefore any R-module $M \in K^d$ has a pseudo-basis in which elements of M are uniquely represented.

Fundamental domain. The *fundamental domain* of a lattice $\mathcal{L}$ together with a basis $\mathcal{B} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ is

$$\mathcal{F}_\mathcal{B} = \{t_1\mathbf{v}_1 + t_2\mathbf{v}_2 + \cdots + t_n\mathbf{v}_n \,:\, -\frac{1}{2} \leq t_i < \frac{1}{2} \text{ for all } i\}.$$

The fundamental domain is the generalisation to dimension n of a parallellepiped, and every vector in $\mathbb{R}^n$ can be written, uniquely, as a sum of a vector in $\mathcal{L}$ and a vector in $\mathcal{F}_\mathcal{B}$.

Lattice invariants. The n-dimensional volume of the fundamental domain is denoted $\mathrm{Vol}(\mathcal{F}_\mathcal{B})$. For $\mathcal{B} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$, let $\mathbf{V}$ be the matrix such that $(\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n) = (\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_n)\mathbf{V}$ where $\mathbf{e}_i$ is the vector with a 1 in entry i and zeros elsewhere. Then

$$\mathrm{Vol}(\mathcal{F}_\mathcal{B}) = |\det \mathbf{V}|.$$

The fundamental domain depends on the basis, but its volume does not because we change basis by multiplication with a unimodular matrix (and for two square matrices $\mathbf{M}$ and $\mathbf{N}$ of equal size, $\det(\mathbf{MN}) = \det \mathbf{M} \cdot \det \mathbf{N}$). Thus $\mathrm{Vol}(\mathcal{F}_\mathcal{B})$ does not depend on which basis for $\mathcal{L}$ is used to calculate it. We define the *determinant of* $\mathcal{L}$ by $\det \mathcal{L} = \mathrm{Vol}(\mathcal{F}_\mathcal{B})$.

The determinant of a lattice is a *lattice invariant*, meaning it is a property of the lattice that does not depend on the choice of basis. There are a number of other lattice invariants, for instance the length of the shortest nonzero vector in a lattice. Because a lattice is discrete, a shortest nonzero vector must exist (though it need not be unique). The length of a shortest vector in a lattice $\mathcal{L}$ is called the *minimum distance of* $\mathcal{L}$, and is denoted $\lambda_1(\mathcal{L})$.

Similarly, the *i-th successive minimum of* $\mathcal{L}$, which is the smallest value r such that $\mathcal{L}$ has i linearly independent vectors of norm at most r, is a lattice invariant. It is denoted $\lambda_i(\mathcal{L})$.

Finding a short vector in a lattice is not a straightforward problem, but we can at least get some idea of how long such a vector will be.

Theorem 2.1. *(Minkowski's Theorem) Let $\mathcal{L}$ be a lattice of dimension n, and $S \subset \mathbb{R}^n$ a symmetric convex set such that*

$$\mathrm{Vol}(S) > 2^n \det \mathcal{L}.$$

Then S contains a nonzero lattice vector. Moreover, if S is closed, the inequality need not be strict.

Applying Minkowski's theorem to a hypercube gives the following bound for the minimum distance of a lattice.

Theorem 2.2. *(Hermite's Theorem) For any lattice $\mathcal{L}$ of dimension n, there is some vector $\mathbf{v} \in \mathcal{L}$ such that*

$$||\mathbf{v}|| \leq \sqrt{n} \cdot (\det \mathcal{L})^{1/n}.$$

Minkowski's theorem can be applied to a hypersphere instead of a hypercube to get a better estimate. Let $\mathbb{B}_r(\mathbf{0})$ be a ball of radius r in $\mathbb{R}^n$, centered at $\mathbf{0}$. By Theorem 6.30 in [21], the volume of $\mathbb{B}_r(\mathbf{0})$ is

$$\mathrm{Vol}(\mathbb{B}_r(\mathbf{0})) = \frac{\pi^{n/2} r^n}{\Gamma(1+n/2)},$$

where $\Gamma(s) = \int_0^\infty t^{s-1} e^{-t} dt$ (for $s > 0$) is the gamma function. By Proposition 6.29 in [21],

$$\mathrm{Vol}(\mathbb{B}_r(\mathbf{0})) = \frac{\pi^{n/2} r^n}{\Gamma(1+n/2)} = \left(\sqrt{\frac{2\pi e}{n}} r\right)^n \cdot \frac{1}{\sqrt{\pi} e^{O(1)}} \text{ as } n \to \infty.$$

Thus for large enough n, it follows from Minkowski's theorem that an n-dimensional lattice $\mathcal{L}$ contains a vector of length at most $\sqrt{\frac{2n}{\pi e}} (\det(\mathcal{L}))^{1/n} \cdot \pi^{1/2n} e^{O(1/n)}$, which gives a better bound than Hermite's theorem when n is large.

It is also interesting to ask how long we might reasonably expect a shortest vector to be in a lattice that is somehow randomly chosen. Intuitively, the number of lattice points in $\mathbb{B}_r(\mathbf{0})$ is approximately the volume of $\mathbb{B}_r(\mathbf{0})$ divided by $\det(\mathcal{L}) = \mathrm{Vol}(\mathcal{F}_\mathcal{B})$ for any basis $\mathcal{B}$ of $\mathcal{L}$ (though lattice points near the boundary of $\mathbb{B}_r(\mathbf{0})$ will create an error). Therefore to estimate how large $\mathbb{B}_r(\mathbf{0})$ needs to be to contain one lattice point, we set $\mathrm{Vol}(\mathbb{B}_r(\mathbf{0})) = \det(\mathcal{L})$ and solve for r. Assuming n is large enough that

$$\mathrm{Vol}(\mathbb{B}_r(\mathbf{0})) \approx \left(\sqrt{\frac{2\pi e}{n}} r\right)^n,$$

it follows that $\mathrm{Vol}(\mathbb{B}_r(\mathbf{0})) = \det(\mathcal{L})$ for $r \approx \sqrt{\frac{n}{2\pi e}} \cdot (\det \mathcal{L})^{1/n}$.

This gives the *Gaussian expected shortest length*

$$\sigma(\mathcal{L}) = \sqrt{\frac{n}{2\pi e}} \cdot (\det \mathcal{L})^{1/n},$$

and the *Gaussian heuristic*, which says that for any "randomly chosen lattice" $\mathcal{L}$, $\lambda_1(\mathcal{L}) \approx \sigma(\mathcal{L})$.

2.2.1 Finding short vectors in a lattice

There are a number of lattice problems that are computationally hard, in that there are no known algorithms that can solve them to within some useful approximation in reasonable time, by which (at least in cryptography) we mean that neither the approximation factor nor the time should grow more quickly than a polynomial in the dimension of the lattice. Most of these are variants of two main problems, the *shortest vector problem* (SVP) and the *closest vector problem* (CVP). These are defined in Section 2.3, but informally, to solve SVP is to find a shortest vector in a given lattice (there can be several shortest vectors,

in which case it suffices to find one), and to solve CVP is to find, given a lattice and a vector not belonging to the lattice, a lattice vector that is closest to the given vector.

Both these problems are trivial given an orthogonal basis, but when working with lattices orthogonal bases are not common. Both SVP and CVP are still fairly easy to solve, at least up to some approximation, if given access to a sufficiently "good" basis (how good it needs to be depends on what approximation factor is acceptable), but they are difficult to solve given only a "bad" basis.

Since the volume of a parallellepiped with sides of fixed length is greatest when the sides are pairwise orthogonal, we have, for a fundamental domain $\mathrm{Vol}(\mathcal{F})$ of a lattice $\mathcal{L}$ and any basis $\mathcal{B} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ of $\mathcal{L}$,

$$\det \mathcal{L} = \mathrm{Vol}(\mathcal{F}_\mathcal{B}) \leq ||\mathbf{v}_1||\,||\mathbf{v}_2|| \cdots ||\mathbf{v}_n||.$$

This is called *Hadamard's inequality*.

Definition 2.3. Let $\mathcal{L}$ be a lattice with a chosen basis $\mathcal{B} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$. We define the *Hadamard ratio of* $\mathcal{B}$ to be

$$\mathcal{H}(\mathcal{B}) = \Big(\frac{\det(\mathcal{L})}{||\mathbf{v}_1||\,||\mathbf{v}_2|| \cdots ||\mathbf{v}_n||}\Big)^{1/n}.$$

By Hadamard's inequality, $0 < \mathcal{H}(\mathcal{B}) \leq 1$. The Hadamard ratio is not a lattice invariant, but can be used as a heuristic to judge how good a basis is, because it is closer to 1 when the basis is more orthogonal, and closer to 0 when it is less orthogonal. Thus the Hadamard ratio can be used to compare two bases for the same lattice and see which is "better", i.e., closer to being orthogonal.

In 1982, Lenstra, Lenstra and Lovász introduced the LLL-algorithm, which uses the above fact to produce a comparatively good basis in polynomial time. This new basis can be used to find approximate solutions to SVP and CVP in a small lattice, but it does not work as well for larger lattice dimensions because the approximation factors are exponential in the lattice dimension n. The LLL-algorithm, and some later variants of it, are described in Section 6.12 of [21].

Definition 2.4. A basis $\mathbf{v}_1, ..., \mathbf{v}_n$ for the lattice $\mathcal{L}$ is called *LLL-reduced* if it fulfills the two conditions

$$\textbf{(Size condition)} \quad |\mu_{i,j}| = \frac{|\langle \mathbf{v}_i, \mathbf{v}_j^* \rangle|}{||\mathbf{v}_j^*||^2} \leq \frac{1}{2} \quad \text{for all } 1 \leq j < i \leq n$$

$$\textbf{(Lovász condition)} \quad ||\mathbf{v}_i^*||^2 \geq \big(\frac{3}{4} - \mu_{i,i-1}^2\big)||\mathbf{v}_{i-1}^*||^2 \quad \text{for all } 1 < i \leq n,$$

where $\mathbf{v}_1^*, ..., \mathbf{v}_n^*$ is the Gram-Schmidt orthogonal basis associated to $\mathbf{v}_1, ..., \mathbf{v}_n$.

Notice that the order of the vectors affects whether the basis fulfills the size condition.

Any LLL-reduced basis $\mathbf{v}_1, ..., \mathbf{v}_n$ for the lattice $\mathcal{L}$ has the properties

$$\prod_{i=1}^{n} ||\mathbf{v}_i|| \leq 2^{n(n-1)/4} \det \mathcal{L},$$
$$||\mathbf{v}_j|| \leq 2^{(i-1)/2} ||\mathbf{v}_i^*|| \quad \text{for all } 1 \leq j \leq i \leq n.$$

Moreover, the first basis vector $\mathbf{v}_1$ fulfills

$$||\mathbf{v}_1|| \leq 2^{(n-1)/4} |\det \mathcal{L}|^{1/n}$$

and is a solution for the apprximate shortest vector problem SVP_γ for approximation factor $\gamma = 2^{(n-1)/2}$, meaning that it is longer than a shortest vector by at most a factor $\gamma = 2^{(n-1)/2}$.

Babai offers two procedures for finding an approximate solution to CVP in [5], the *rounding off procedure* and the *nearest plane procedure*. In the following, recall that for $a \in \mathbb{R}$, $\lfloor a \rceil$ is the closest integer to a (with ties broken upwards).

Given a lattice $\mathcal{L} \subset \mathbb{R}^n$ with basis $\mathbf{v}_1, \ldots, \mathbf{v}_n$, a vector $\mathbf{w} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_n\mathbf{v}_n$ with $a_1, ..., a_n \in \mathbb{R}$, and a function $\gamma(n)$, the challenge is to find a vector $\mathbf{x}$ such that $||\mathbf{w} - \mathbf{x}|| \leq \gamma(n)||\mathbf{w} - \mathbf{u}||$, where $\mathbf{u}$ is a closest lattice vector to $\mathbf{w}$.

The Rounding off Procedure. Set $b_i = \lfloor a_i \rceil$ for $i = 1, 2, ..., n$, and set $\mathbf{x} = b_1\mathbf{v}_1 + b_2\mathbf{v}_2 + \cdots + b_n\mathbf{v}_n$.

The Nearest Plane Procedure. Let U be the linear subspace of $\mathbb{R}^n$ spanned by $\mathbf{v}_1, \ldots, \mathbf{v}_{n-1}$, and $\mathcal{L}'$ the sublattice spanned by $\mathbf{v}_1, \ldots, \mathbf{v}_{n-1}$. Find a vector $\mathbf{z} \in \mathcal{L}$ such that the distance between $\mathbf{w}$ and $U + \mathbf{z}$ is minimal and let $\mathbf{w}'$ be the orthogonal projection of $\mathbf{w}$ on $U + \mathbf{z}$.[3]

Recursively, find $\mathbf{y} \in \mathcal{L}'$ near $\mathbf{w}' - \mathbf{z}$ and let $\mathbf{x} = \mathbf{y} + \mathbf{z}$.

Theorem 2.3. *(Theorem 3.1, [5]) If $\mathbf{v}_1, \ldots, \mathbf{v}_n$ is an LLL-reduced basis[4], then the nearest plane procedure produces a vector $\mathbf{x}$ closest to $\mathbf{w}$ to within a factor $\gamma = 2^{n/2}$.*

Theorem 2.4. *(Theorem 3.2, [5]) If $\mathbf{v}_1, \ldots, \mathbf{v}_n$ is an LLL-reduced basis, then the rounding off procedure produces a vector $\mathbf{x}$ closest to $\mathbf{w}$ to within a factor $\gamma = 1 + 2n(9/2)^{n/2}$.*

2.2.2 Example of Babai's rounding off procedure

The following example illustrates how Babai's rounding off procedure works in a good and a bad basis, respectively.

[3] To find $\mathbf{z}$ and $\mathbf{w}'$, write $\mathbf{w}$ as a linear combination $\alpha_1\mathbf{v}_1^* + \cdots + \alpha_n\mathbf{v}_n^*$ of the orthogonalised basis $\mathbf{v}_1^*, \cdots, \mathbf{v}_n^*$ and let $c = \lfloor \alpha_n \rceil$. Then $\mathbf{w}' = \alpha_1\mathbf{v}_1^* + \cdots + \alpha_{n-1}\mathbf{v}_{n-1}^* + c\mathbf{v}_n^*$ and $\mathbf{z} = c\mathbf{v}_n$.

[4] Babai uses a *Lovász-reduced* basis instead, where the second condition is

$$||\mathbf{v}_i^*|| \geq \frac{||\mathbf{v}_{i-1}^*||}{\sqrt{2}} \quad \text{for all } 1 < i \leq n.$$

Since the size condition in Definition 2.2 requires $\mu_{i,i-1}^2 \leq 1/4$, an LLL-reduced basis is also Lovász-reduced.

Let $\mathcal{L} \subset \mathbb{R}^2$ be the lattice spanned by $\mathbf{v}_1 = (-351, 122)$ and $\mathbf{v}_2 = (108, 447)$. This basis has a Hadamard ratio of $\mathcal{H}(\mathbf{v}_1, \mathbf{v}_2) = \sqrt{\frac{\det \mathcal{L}}{||\mathbf{v}_1|| \, ||\mathbf{v}_2||}} \approx 0.998$ so it is quite a good basis.

We want to find the closest lattice vector to $\mathbf{w} = (40119, 72324)$. Using the rounding off procedure, we write $\mathbf{w} \approx -59.52 \cdot \mathbf{v}_1 + 178.04 \cdot \mathbf{v}_2$, and get the approximate answer

$$\mathbf{x} = -60\mathbf{v}_1 + 178\mathbf{v}_2 = (40284, 72246),$$

with

$$||\mathbf{w} - \mathbf{x}|| \approx 183.$$

Now we try to solve the same problem using another basis for the lattice, with basis vectors $\mathbf{u}_1 = 7\mathbf{v}_1 + 8\mathbf{v}_2 = (-1593, 4430)$ and $\mathbf{u}_2 = 29\mathbf{v}_1 + 33\mathbf{v}_2 = (-6615, 18289)$. This basis has a Hadamard ratio of $\mathcal{H}(\mathbf{u}_1, \mathbf{u}_2) = \sqrt{\frac{\det \mathcal{L}}{||\mathbf{u}_1|| \, ||\mathbf{u}_2||}} \approx 0.043$ so is significantly worse than the previous one.

Again using the rounding off procedure, we write $\mathbf{w} \approx 7127.29 \cdot \mathbf{u}_1 - 1722.43 \cdot \mathbf{u}_2$, and the procedure gives the vector

$$\mathbf{x}' = 7127\mathbf{u}_1 - 1722\mathbf{u}_2 = (37719, 78952),$$

with

$$||\mathbf{w} - \mathbf{x}'|| \approx 7049.$$

2.2.3 The LLL-algorithm

Algorithm 1 The LLL algorithm

Input: Basis $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ for the lattice $\mathcal{L}$
Output: Produces an LLL-reduced basis for the lattice $\mathcal{L}$

```
k = 2
v*_1 = v_1
while k ≤ n do
    for j = 1, 2, ..., k − 1 do
        v_k = v_k − ⌊μ_{k,j}⌉ v*_j                    ▷ Size reduction
    if ||v*_k||^2 ≥ (3/4 − μ^2_{k,k−1}) ||v*_{k−1}||^2 then    ▷ Lovász condition
        k = k + 1
    else
        Swap v_{k−1} and v_k                          ▷ Swap step
        k = max(k − 1, 2)
return {v_1, ..., v_n}
```

At each step $\mathbf{v}_1^*, \ldots, \mathbf{v}_k^*$ is the set of orthogonal vectors obtained by applying Gram-Schmidt to the current $\mathbf{v}_1, \ldots, \mathbf{v}_k$, and $\mu_{i,j} = (\mathbf{v}_i \cdot \mathbf{v}_j^*)/||\mathbf{v}_j^*||^2$.

The LLL-algorithm, shown in Algorithm 1, produces an LLL-reduced basis for $\mathcal{L}$ in polynomial time (the main loop is executed no more than $\mathcal{O}(n^2 \log n +$

$n^2 \log B$) times, where B is the length of the longest vector of the basis which is to be reduced). Due to the swap step (line 9), the sublattices spanned by $\mathbf{v}_1, ..., \mathbf{v}_l$ for $1 \leq l < n$ change, and what the algorithm is attempting to do is to minimize the determinants of each of these sublattices, along with size reductions where possible.

The value 3/4 in the Lovász condition (line 6) can be replaced by any value strictly smaller than 1, and the algorithm will still terminate in polynomial time, but if it is replaced by 1 (which is needed to guarantee that the determinants of the sublattices will be minimized) this may not be the case (it is an open problem). In practice a value between 3/4 and 1 is usually used, though a larger value will not always give a better basis. The order of the vectors in the input basis affects the output basis.

There are some issues with the LLL-algorithm. Firstly, the Gram-Schmidt orthogonalisation is not always the same, so even if it is stored for repeated use when possible, it must still be calculated many times. Moreover, for high dimensions n the intermediate calculations involve huge numbers and it can be necessary, as Schnorr and Euchner suggest in [34], to use floating point approximations for the numbers $|\mu_{i,j}|$ and $||v_i^*||^2$, leading to round off errors.

Efficiently implemented the algorithm will terminate after no more than $\mathcal{O}(n^6(\log B)^3)$ basic operations.

Remark. *According to Section 6.11.2 of [21], LLL and other lattice reduction algorithms can easily find the shortest vector if it is significantly shorter than the Gaussian expected shortest length (say $O(2^n)$ shorter). Therefore, if it is important that it should be hard to find a shortest vector in a lattice, no vector should be too much shorter than the Gaussian expected shortest length.*

2.2.4 Variants of LLL

There are alternatives to the LLL algorithm which give better results, though at the cost of longer run time. The *deep insertion method*, presented by Schnorr and Euchner in 1994 [34], may not terminate in polynomial time, but according to [21] (Section 6.12.4) it will in practice run quite quickly on most lattices and tends to give a significantly better result than the LLL-algorithm. Instead of a swap step, the deep insertion method inserts the vector $\mathbf{v}_k$ between the vectors $\mathbf{v}_{i-1}$ and $\mathbf{v}_i$, where i is chosen to get a large size reduction. Specifically (for some chosen δ such that $1/4 < \delta < 1$), $i = 1$ if $\delta \cdot ||v_1^*||^2 > ||v_k^*||^2$, and otherwise i is chosen to be the largest $i \in [1, k-1]$ such that

$$\delta \cdot ||v_i^*||^2 \leq ||v_k^*||^2 - \sum_{j=1}^{i-1} \mu_{k,j} ||v_j^*||^2.$$

If this inequality holds for all $1 \leq i < k$, v_k is not moved at all.

In [34] floating point arithmetic is used for the vector norms.

Definition 2.5. Let $\mathbf{v}_1, ..., \mathbf{v}_n$ be a set of vectors. For $i = 0, ..., n$, let $\pi_i : \mathcal{L} \rightarrow$

$\mathbb{R}^n$ be the maps defined by

$$\pi_0(\mathbf{v}) = \mathbf{v} \text{ and } \pi_i(\mathbf{v}) = \mathbf{v} - \sum_{j=1}^{i} \frac{\langle \mathbf{v}, \mathbf{v}_j^* \rangle}{||\mathbf{v}_j^*||^2} \mathbf{v}_j^*,$$

where $\mathbf{v}_1^*, ..., \mathbf{v}_n^*$ is the Gram-Schmidt orthogonal basis associated to $\mathbf{v}_1, ..., \mathbf{v}_n$. A basis $\mathbf{v}_1, ..., \mathbf{v}_n$ for a lattice $\mathcal{L}$ is called *Korkin-Zolotarev reduced* if

1. $\mathbf{v}_1$ is a shortest nonzero vector of $\mathcal{L}$.
2. For $i = 2, 3, ..., n$, $\mathbf{v}_i$ is chosen so that $\pi_{i-1}(\mathbf{v}_i)$ is the shortest vector in $\pi_{i-1}(\mathcal{L})$.
3. For all $1 \leq i < j \leq n$, $|\langle \pi_{i-1}(\mathbf{v}_i), \pi_{i-1}(\mathbf{v}_j) \rangle| \leq \frac{1}{2} ||\pi_{i-1}(\mathbf{v}_i)||^2$.

In general a KZ-reduced basis is far better than an LLL-reduced basis, and by definition its first vector is always a solution to SVP. It is therefore not surprising that all known methods for finding such a basis require exponential time, in the dimension n.

The *BKZ-LLL algorithm*, where BKZ stands for *block Korkin-Zolotarev*, compromises by replacing the swap step of the LLL-algorithm with a block reduction, where a block of b vectors spanning some sublattice is reduced to a KZ-reduced basis for the same sublattice. Larger blocks give a better basis, but also slow down the algorithm, and the block-size b can be chosen with this in mind.

The BKZ-LLL algorithm gives a better basis for larger block size, but on the other hand the larger block size means that the algorithm takes longer to run. According to Remark 6.76 in [21], using BKZ-LLL to find a vector no more than a factor $\gamma = O(n^\delta)$ longer than the shortest vector (for some fixed δ) requires, both in theory and (according to experimental evidence) in practice, that as n grows the block size must grow linearly in n, and then the running time grows exponentially.

2.2.5 Hermite Normal Form

Lattice cryptography sometimes means working with large matrices $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, which may be expressed slightly more efficiently using the *Hermite normal form*.[5]

Definition 2.6. A matrix $\mathbf{H} \in \mathbb{Z}^{m \times n}$, with $m \geq n$, is in *(lower triangular) Hermite normal form* (HNF) if

- any columns consisting entirely of zeros are to the right,

[5]Because these matrices usually have entries in $\mathbb{Z}_q$ rather than $\mathbb{Z}$, they are not formally expressed using Hermite normal form, which is only defined for integer matrices, but they can still be written in a way that resembles Hermite normal form and by which they can be expressed more compactly.

- the pivot (first nonzero entry) of each nonzero column is positive, and strictly below the pivot of the column immediately to the left, and
- entries to the left of a pivot are nonnegative and strictly smaller than the pivot.

The third condition can be seen as requiring elements to the left of a pivot to be reduced modulo that pivot. The second condition together with $m \geq n$ implies that $\mathbf{H}$ is lower triangular, i.e., $h_{ij} = 0$ for $i < j$. A matrix $\mathbf{H} \in \mathbb{Z}^{n \times m}$, with $m \geq n$, has *upper triangular Hermite normal form* if its transpose has lower triangular Hermite normal form.

For any integer matrix $\mathbf{A} \in \mathbb{Z}^{m \times n}$, there exists a unique matrix $\mathbf{H} \in \mathbb{Z}^{m \times n}$ and a square unimodular matrix $\mathbf{U}$ such that $\mathbf{H} = \mathbf{AU}$.

2.2.6 Multiplying polynomials

Some problems based on more structured lattices can be described using polynomial rings, and it then becomes relevant to find efficient ways to multiply polynomials. A popular method is to use the *number-theoretic transform (NTT)*, a specialization over $\mathbb{Z}_q$ for some integer q of the discrete Fourier transform.

Number-theoretic Transform NTT

Let $R = \mathbb{Z}[X]/(X^n + 1)$ and $R_q = R/qR$. If n is a power of 2 and q is a prime such that $2n|(q-1)$, there exists a primitive n-th root of unity ω and its square root mod q, γ. For $\mathbf{g} = \sum_{i=0}^{n-1} g_i X^i \in R_q$, the NTT transform is the function $\mathsf{NTT} : R_q \to R_q$ defined by

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

NTT is invertible, and the inverse is denoted NTT^{-1}.

The point of the transform is that it allows us to multiply polynomials in R_q by coefficient-wise multiplication of their images under the NTT transform. That is, for $a, b \in R_q$,

$$ab = \mathsf{NTT}^{-1}(\mathsf{NTT}(a) \circ \mathsf{NTT}(b)),$$

where $\circ$ denotes coefficient-wise multiplication.

In many schemes, it is argued that the choice of n as a power of 2 makes the transform more efficient, but other choices of n are possible. In fact, it is not even necessary that q be prime, as long as a principal nth root of unity[6] ω exists.

Using the GNU Multiple Precision Arithmetic library

In [16], Fateman discussed encoding polynomials with integer coefficients as big numbers and then using GMP (GNU Multiple Precision) to multiply

[6] That is, $\omega^n = 1$ and $\sum_{j=0}^{n-1} \omega^{jk} = 0$ for $1 \leq k < n$.

them. This way we take advantage of the considerable effort that is put into maintaining and developing GMP. To multiply two polynomials in this way, we encode them as integers by evaluating them at one point, chosen depending on their degrees and sizes of their coefficients to be large enough to allow decoding (translating back into polynomials) without errors after multiplying the integers. Working in a finite field (as is the case for the lattice schemes discussed here) simplifies the choice of point for evaluation. If the polynomials to be multiplied are in $R_q = R/qR$ as above, the point at which they are evaluated need not be larger than nq^2 (if we are only multiplying two polynomials, and then decoding the result), though in practice it is more convenient to chose the smallest power of 2 larger than this number as it simplifies encoding and decoding.

Note that if we are multiplying polynomials in R_q, we first have to express them as polynomials in $\mathbb{Z}[X]$ (choosing their respective representatives with degree under n and coefficients between 0 and $q-1$). Encoding, multiplying and decoding these polynomials gives their product in $\mathbb{Z}[X]$, and to recover the product in R_q we must reduce this product in $\mathbb{Z}[X]$ by (X^n+1), and reduce each coefficient modulo q, and Fateman comments that there does not seem to be any particularly quick way of doing this.

This method seems to have no additional requirements on n and q beyond that they be positive integers.

Karatsuba multiplication for polynomials. Karatsuba multiplication for polynomials means splitting one multiplication of large polynomials into three multiplications of polynomials of half the size (which can recursively be multiplied using Karatsuba multiplication). Adapting Bernsteins description in Section 5 of [8] to the case of integer polynomials, let a and b be two polynomials of degree strictly less than $2n$ in $\mathbb{Z}[X]$, and rewrite them as $a_0 + a_1 Y, b_0 + b_1 Y \in \mathbb{Z}[X,Y]$ where $a_0, a_1, b_0, b_1 \in \mathbb{Z}[X]$ have degree less than n. This is done by mapping a and b into $\mathbb{Z}[X,Y]/(X^n - Y)$ and then lifting them to $\mathbb{Z}[X,Y]$, choosing representatives for them in such a way that Y replaces X^n where possible. We can now compute

$$(a_0 + a_1 Y)(b_0 + b_1 Y) = t + ((a_0 + a_1)(b_0 + b_1) - t - u)Y + uY^2,$$

where $t = a_0 b_0$ and $u = a_1 b_1$. Thus instead of a product of two polynomials of degree $< 2n$, we have three products of polynomials of degree $< n$, and a few additions and subtractions. Substituting X^n for Y will then give the product ab.

Karatsuba can be used recursively to compute the products $a_0 b_0$, $a_1 b_1$ and $(a_0 + a_1)(b_0 + b_1)$ until the polynomials are so small that it is more efficient to use some other more naive method for multiplication.

Toom-Cook multiplication. Toom-Cook multiplication is a generalisation of Karatsuba, where one polynomial multiplication is split into several multiplications, comparatively smaller than those in Karatsuba. For instance, instead of one multiplication of $4n$-degree polynomials, we can have seven multiplications of n-degree polynomials.

2.2.7 Gaussian distribution

For $s > 0$, the n-dimensional *Gaussian function* is the function $\rho_s : \mathbb{R}^n \to \mathbb{R}^+$ defined by $\rho_s(\mathbf{x}) = \exp(-\pi||\mathbf{x}||^2/s^2)$.

Definition 2.7. For $s > 0$, the n-dimensional *Gaussian distribution* is the distribution over $\mathbb{R}^n$ defined by the probability density function

$$D_s(\mathbf{x}) = \rho_s(\mathbf{x})/s.$$

Definition 2.8. For $s > 0$ and a lattice $\mathcal{L} \subset \mathbb{R}^n$, the *discrete Gaussian distribution* is defined as

$$D_{\mathcal{L},s}(\mathbf{x}) = \frac{\rho_s(\mathbf{x})}{\rho_s(\mathcal{L})} \quad \text{for } \mathbf{x} \in \mathcal{L},$$

where $\rho_s(\mathcal{L}) = \sum_{\mathbf{v}\in\mathcal{L}} \rho_s(\mathbf{v})$ (and $D_{\mathcal{L},s}(\mathbf{x}) = 0$ for $\mathbf{x} \notin \mathcal{L}$).

2.3 Hard lattice problems

The two main lattice problems are the *shortest vector problem* and the *closest vector problem*. The former asks for the shortest vector in a given lattice, or one of them if the shortest vector is not unique. The latter asks for a lattice vector closest to some given vector which is not in the lattice.

2.3.1 SVP and variants

There are several variants of the shortest vector problem. One variant asks about the shortest vector length, one asks for the unique shortest vector and another for a shortest basis for the lattice. Most of the variants (all of those given here) only ask for approximate solutions, since this is more useful in reductions to other problems like the *shortest integer solution problem* and the *learning with errors problem* (Sections 3.1 and 3.4 respectively).

In all the following definitions, let $\mathcal{L}$ be an n-dimensional lattice, and let $\mathcal{B}$ be a basis for $\mathcal{L}$.

Definition 2.9. (The Shortest Vector Problem, SVP) Given $\mathcal{B}$, find a vector $\mathbf{v} \in \mathcal{L}$ such that $||v|| = \lambda_1(\mathcal{L})$.

Definition 2.10. (The Approximate Shortest Vector Problem, SVP_γ) Given $\mathcal{B}$ and a function γ over n, find a vector $\mathbf{v} \in \mathcal{L}$ such that $||v|| \leq \gamma(n)\lambda_1(\mathcal{L})$.

Taking $\gamma(n) = 1$ gives the original problem.

Definition 2.11. (Decisional Approximate SVP, GapSVP_γ) Given $\mathcal{B}$ and a function γ over n, and knowing that either $\lambda_1(\mathcal{L}) \leq 1$ or $\lambda_1(\mathcal{L}) > \gamma(n)$, determine which is the case.

Definition 2.12. (Approximate Unique Shortest Vector Problem, uSVP_γ) Given $\mathcal{B}$ and a function γ over n, uSVP_γ, and knowing that $\lambda_2(\mathcal{L}) \geq \gamma\lambda_1(\mathcal{L})$, find a vector in $\mathcal{L}$ of length $\lambda_1(\mathcal{L})$.

This is a *promise problem*, meaning that the problem assumes something which is not necessarily true in general. In this case, it is assumed as part of uSVP_γ that $\lambda_2(\mathcal{L}) \geq \gamma\lambda_1(\mathcal{L})$, which does not hold in all lattices, so the problem is restricted to certain lattices. Note that a solution to uSVP_γ is unique up to a factor -1, unless $\gamma = 1$.

Definition 2.13. (Approximate Shortest Independent Vectors problem, SIVP_γ) Given $\mathcal{B}$ and a function γ over n, find a set of n linearly independent lattice vectors, all of length at most $\gamma(n)\lambda_n(\mathcal{L})$.

A solution to SIVP_γ is a (approximate) shortest basis for $\mathcal{L}$. In [25], Langlois and Stehlé use a slightly more general version of SIVP_γ. Note that replacing ϕ in the following definition gives back the usual SIVP_γ problem.

Definition 2.14. (Approximate General Independent Vectors problem, GIVP^ϕ_γ) Given $\mathcal{B}$, a function γ over n, and a function ϕ on $\mathcal{L}$, find a set of n linearly independent lattice vectors, all of length at most $\gamma(n)\phi(\mathcal{L})$.

2.3.2 CVP and variants

The closest vector problem has fewer variants, and is less commonly used for reductions. Again, let $\mathcal{L}$ be an n-dimensional lattice, and let $\mathcal{B}$ be a basis for $\mathcal{L}$.

Definition 2.15. (The Closest Vector Problem, CVP) Given $\mathcal{B}$ and a vector $\mathbf{w} \in \mathbb{R}^n$ that is not in $\mathcal{L}$, find a vector $\mathbf{v} \in \mathcal{L}$ that minimizes $||\mathbf{w} - \mathbf{v}||$.

According to Peikert [30], no cryptosystem based on CVP or its approximate variant has yet been proved secure. However, there is a more useful variant of CVP called the *bounded distance decoding problem*, where $\mathbf{w}$ is guaranteed to be rather close to some lattice point, and the solution is unique.

Definition 2.16. (Bounded Distance Decoding Problem, BDD_γ) Given $\mathcal{B}$, a function γ over n and and a vector $\mathbf{w} \in \mathbb{R}^n$ that is not in $\mathcal{L}$ with the guarantee that $\text{dist}(\mathbf{w}, \mathcal{L}) < d = \lambda_1(\mathcal{L})/(2\gamma(n))$, find the unique lattice vector $\mathbf{v}$ such that $||\mathbf{w} - \mathbf{v}|| < d$.

Note that like uSVP_γ, BDD_γ is a promise problem, but where uSVP_γ has certain restrictions on which lattices can be used BDD_γ has a solution in any lattice as long as some care is taken in the choice of the vector w.

FrodoKEM, a cryptosystem submitted to NIST, uses a variant of BDD where the adversary is assumed to have access to an oracle providing discrete Gaussian samples.

Definition 2.17. (Bounded Distance Decoding Problem with Discrete Gaussian Samples, $\text{BDDwDGS}_{d,r}$) Given $\mathcal{B}$, positive real values $d < \lambda_1(\mathcal{L})/2$ and $r > 0$, a vector $\mathbf{w} \in \mathbb{R}^n$ that is not in $\mathcal{L}$ with the guarantee that $\text{dist}(\mathbf{w}, \mathcal{L}) \leq d$, and access to an oracle that samples from $D_{\mathcal{L}^*,s}$ for any adaptively queried $s \geq r$ (where $\mathcal{L}^*$ is the dual of $\mathcal{L}$), find the unique lattice vector $\mathbf{v}$ closest to $\mathbf{w}$.

BDDwDGS is a variant of the *closest vector problem with preprocessing* (CVPP) which is essentially CVP except that the lattice is assumed to be fixed beforehand, and an attacker is allowed unlimited time to preprocess the lattice. Among other things, the attacker can compute samples from $D_{\mathcal{L}^*,r}$, which can be used to approximate the *periodic Gaussian function* $f : \mathbb{R}^n \to \mathbb{R}_+$ defined by

$$f(\mathbf{w}) = \frac{\rho_1(\mathcal{L}+\mathbf{w})}{\rho_1(\mathcal{L})} = \frac{\sum_{\mathbf{x}\in\mathcal{L}+\mathbf{w}} \exp(-\pi||\mathbf{x}||^2)}{\sum_{\mathbf{x}\in\mathcal{L}} \exp(-\pi||\mathbf{x}||^2)}.$$

This approximation, denoted f_W, can be made in the preprocessing stage, and then $f(\mathbf{w})$ can be approximated efficiently. Because $f_W(\mathbf{w})$ attains its maxima in the lattice points it can be used to find the nearest lattice point to $\mathbf{w}$ if the distance from $\mathbf{w}$ to the lattice is no more than about $O(\sqrt{\log n/n})\cdot\lambda_i(\mathcal{L})$. Thus, with preprocessing, or alternatively with an oracle that samples from $D_{\mathcal{L}^*,r}$ for some fixed r, we can solve BDD_γ for $\gamma = O(\sqrt{n/\log n})$. For details on this, and improvements on the bound, see [15].

However according to the specification [39] of FrodoKEM, known algorithms to solve this problem use samples from $D_{\mathcal{L}^*,r}$ for some fixed r, whereas the reduction from BDDwDGS to LWE uses the fact that, as in the definition of BDDwDGS above, $s \geq r$ can be adaptively queried.

2.3.3 Hardness of SVP and CVP

The closest vector problem CVP is known to be $\mathcal{NP}$-hard, and the shortest vector problem SVP is $\mathcal{NP}$-hard under randomised reduction, i.e., if the class of polynomial-time algorithms is enlarged to include algorithms which with high probability will terminate in polynomial time with a correct result. According to [21] (Section 6.5.1), CVP can often be reduced to SVP in a slightly higher dimension, so CVP is considered a little bit harder than SVP.

The approximate versions are $\mathcal{NP}$-hard under random reduction, but only for certain approximation factors, smaller than those used in cryptography. However, the approximate problems, including the approximate GapSVP, SIVP and BDD problems, are not easy to solve. Peikert writes in [30] that the known polynomial-time algorithms give nearly exponential ($2^{\Theta(n \log\log n/\log n)}$) approximation factors, and known algorithms that give approximation factors that are at most polynomial in n require superexponential ($2^{\Theta(n\log n)}$) time, or exponential ($2^{\Theta(n)}$) time and space.

2.4 Cryptography

2.4.1 Types of cryptosystems

Key Exchange (KE).

A *key exchange* is some method by which two parties create a secret key which they both have, but which an eavesdropper should not be able to work out from the (often public and unencrypted) exchanges between the parties agreeing on a key. The most common type of key exchange protocol follows

the Diffie-Hellman model, where there are two parties A and B, and a public parameter P (either already published, or sent openly from B to A when A asks for it). The key exchange is executed as follows:

- A chooses a secret a and computes M_A as a function of a and P, and sends M_A to B.
- B chooses a secret b and computes M_B as a function of b and P, and sends M_B to A.
- A derives a key K_A from P, a, M_B.
- B derives a key K_B from P, b, M_A.

If the key exchange is successful $K_A = K_B$, and if it is well-designed an eavesdropper should not be able to derive K_A or K_B from only P, M_A and M_B. In practice, we do not expect it to be *impossible* to derive the keys with only the public information, but just that in practice, it should take too many computations to be feasible in a reasonable amount of time. For instance, if a and b can only take a finite number different values, an attacker can try every possible value of a and find one that, together with P, gives M_A, and can then derive the key K_A from P, a, M_B. It is important to make sure sets of possible values for secret parameters are large enough that this type of attack (called a *brute-force attack*) is not computationally feasible in a reasonable amount of time.

Authenticated Key Exchange (AKE). One possible attack on a key exchange is the *man in the middle* attack. This is when an attacker M intercepts the key exchange by claiming to be B when communicating with A, and A when communicating with B, so that A and B think they have agreed on a shared key but have in fact both agreed on different shared keys with M. M can then read and relay their ensuing communication, or alter the communication between them.

An *authenticated key exchange* is a key exchange with some sort of identification of one or both parties. For instance, in the communication between a client and a server it is common that the server is identified by a certificate authority, but the client is often not identified. The certificate authority is a trusted third party, which issues a certificate to the server which it sends to the client as part of the key exchange so that the client can trust it is not communicating with an impostor. (Somehow the client must decide which certificate authorities to trust. In the context of the secure browsing protocol HTTPS, which is a common use for certificate authorities, it is the browser that makes this decision, and the certificate authorities' incentive to stay honest is the risk of no longer being supported by browsers if they are found to have provided false certificates.)

Public Key Encryption (PKE). A *public key encryption scheme* consists of three algorithms and a message space $\mathcal{M}$:

- KeyGen() $\rightarrow (pk, sk)$, key generation algorithm (probabilistic), outputs a public key pk and a secret key sk.
- Enc$(pk, m) \rightarrow c$, encryption algorithm (probabilistic or deterministic), takes message $m \in \mathcal{M}$ and pk as input, outputs ciphertext c. (The encryption algorithm can be deterministic and is then denoted Enc$(pk, m; r) \rightarrow c$, where the randomness r, chosen from the randomness space $\mathcal{R}$, is given as explicit input.)
- Dec$(sk, c) \rightarrow m'$ or $\perp$, decryption algorithm (deterministic), takes c and sk as input, outputs message $m' \in \mathcal{M}$ or an error symbol $\perp \notin \mathcal{M}$.

Since the key generation algorithm takes no input it must of course be probabilistic. The encryption algorithm may be either probabilistic or deterministic, but the decryption algorithm should recover the message and is designed to be deterministic.

Key Encapsulation Mechanism (KEM). A *key encapsulation mechanism* consists of three algorithms and a keyspace $\mathcal{K}$:

- KeyGen() $\rightarrow (pk, sk)$, key generation algorithm (probabilistic), outputs a public key pk and a secret key sk.
- Encaps$(pk) \rightarrow (K, c)$, encapsulation algorithm (probabilistic), takes pk as input, outputs encapsulation c and shared secret $K \in \mathcal{K}$.
- Decaps$(sk, c) \rightarrow K'$, decapsulation algorithm (deterministic), takes c and sk as input, outputs shared secret $K' \in \mathcal{K}$.

Again, the key generation algorithm must be probabilistic. The encapsulation algorithm must also be probabilistic since it only takes pk as input. The decapsulation algorithm is of course designed to be deterministic if everything goes well. However, occasionally something goes wrong (the ciphertext input into the decapsulation algorithm can be invalid, and some schemes allow the possibility of decapsulation errors where decapsulation can fail to recover the key despite valid input) and some KEMs hide this by outputting a "fake shared key" which is randomly or pseudorandomly generated in some way so that it is not easily distinguishable from a genuine key. This is called *implicit rejection*, as opposed to *explicit rejection* where the decapsulation algorithm outputs an error symbol $\perp \notin \mathcal{K}$ in case of failure.

In practice, a KEM can be built from a PKE, and then the encapsulation algorithm will pick a random message m in the message space of the PKE, and use the encryption algorithm to compute the ciphertext c, while the shared secret K will be computed from m. The decapsulation algorithm will then retrieve the message using the decryption algorithm of the PKE, and compute the shared secret K' from this message.

Definition 2.18.
- A PKE is *perfectly correct* if Dec$(sk,$ Enc$(pk, m)) = m$ with probability 1 for all $m \in \mathcal{M}$, where $(pk, sk) \leftarrow$ KeyGen().

- A KEM is *perfectly correct* if $\text{Decaps}(sk, c) = K$ with probability 1 if $c = \text{Encaps}(pk)$, where $(pk, sk) \leftarrow \text{KeyGen}()$.

If the PKE above is perfectly correct, we always have $m = m'$ if both parties are honest. Similarly if the KEM is perfectly correct, $K = K'$ if both parties are honest. It is convenient for a PKE or KEM to be perfectly correct, but sometimes it is not possible to make a scheme perfectly correct, or doing so impacts performance too much. This tends to be the case for lattice based PKEs (and KEMs), and these often have a small but nonzero probability of decryption (or decapsulation) error.

Data Encapsulation Mechanism (DEM). A *data encapsulation mechanism*, according to Shibuya and Shikata [36], is symmetric (meaning that both parties have access to the same key), and consists of a key space $\mathcal{K}$, a message space $\mathcal{M}$ and two algorithms:

- $\text{Encaps}(dk, m) \rightarrow c$ encapsulation algorithm (deterministic), takes $dk \in \mathcal{K}$ and message $m \in \mathcal{M}$ as input, outputs ciphertext c.
- $\text{Decaps}(dk, c) \rightarrow m$ decapsulation algorithm (deterministic), takes dk and c as input, outputs message $m \in \mathcal{M}$ or $\perp \notin \mathcal{M}$.

Typically it is more convenient to use a hybrid scheme that consists of a KEM (for exchanging a key) and a DEM (for actually exchanging messages) than to use a PKE for an entire communication, because symmetric schemes tend to be more efficient than asymmetric ones.

2.4.2 The Random Oracle Model (ROM)

To show theoretical security for an encryption scheme, we typically want to show that it is as hard as some mathematical problem which is known, or believed, to be hard to solve. This is done with a reduction proof. Let X and Y be two problems. A *reduction from* X *to* Y is an algorithm $\mathcal{R}$ that solves problem X by using a Y-solver $\mathcal{A}$ as a subroutine. $\mathcal{A}$ is treated as an oracle or black box, meaning that it solves Y, but we do not know how it does so, and therefore cannot alter it to turn it into an X-solver. However, if we can formulate problem X in terms of problem Y so that solving Y will give a solution to X, then we can use $\mathcal{A}$ to solve Y, and thereby X.

A reduction is *tight* if $\mathcal{R}$ has approximately the same running time and success probability as $\mathcal{A}$. A sufficiently tight reduction from X to Y proves that if there is a reasonably efficient algorithm that will solve Y, then there is a reasonably efficient algorithm that will solve X, that is, that Y is hard if X is hard. (If the reduction is so loose that $\mathcal{R}$ takes an unreasonable amount of time, or has a high likelihood of being unsuccessful, even if $\mathcal{A}$ is efficient, then it does not say anything meaningful about the hardness of problem Y because then Y could be easy to solve despite X being hard.)

A cryptographic scheme also tends to contain *hash functions*, functions that map data of arbitrary size to data of fixed size. The hash functions used in

cryptography should have output that looks random, in the sense that if two inputs are close to each other this should not be evident from their outputs, and though a hash function is not in general injective it should be *collision-resistant*, meaning that it should be hard to find two different inputs that will map to the same output (*collisions*). This also means that it should be difficult to find a preimage to an output from a hash function.

When proving a reduction, we must take these hash functions into account somehow. Ideally we want them to act as though they were really random functions, giving truly random output but (being functions) always giving the same output for any particular input. We can simulate such a function using a table where we store each input on which the function has been queried together with the output. This way, when the function is queried it first searches through the table to see if that input has been queried before and in that case returns the same output, and otherwise returns a random output and stores the new input and output in the table. Unfortunately, the table would get impractically large (and since it would contain perfectly random values it could not be compressed to a more manageable size) and looking up an entry in it would on average take exponential time in the size of the input.

Thus the hash functions cannot actually be random functions, and we cannot even assume for the sake of the proof that they are because if we want to rule out brute force attacks we cannot allow anything to take exponential time.

Definition 2.19. A *random oracle* is an oracle that given some input will respond with a truly random output, chosen uniformly from its output domain. Given the same input again, the oracle will give the same output as before.

We use the concept of random oracles to prove reductions in the *random oracle model* (ROM), meaning that we assume that there is a random oracle, whose inner workings we know nothing about but which in constant time returns the kind of output we would get from a random function. Proving a reduction reduction from X to Y in the ROM means proving the reduction under this assumption.

In the cryptosystem the random oracles of the security proof are replaced with practical hash functions, and confidence in the system then relies on the hope that the fact that the hash functions are not random oracles will not impact security. Thus the random oracle model does not allow us to rule out all attacks, but we can essentially rule out those that do not use the hash functions.

It is worth noting that in the random oracle model queries the adversary makes to the random oracle can be detected, because the oracle is assumed to be outside of the adversary (who is using $\mathcal{A}$ to solve Y) and the simulator (who is using $\mathcal{A}$ as a subroutine of $\mathcal{R}$ in order to solve X). The simulator can detect what preimages the adversary sends to the random oracle, and even program the answers that are sent back ([6] calls this *extractability/preimage awareness* and *adaptive programmability*, respectively). This means that it is permissable to use knowledge of the adversary's queries to the oracle to prove a reduction from X to Y in the ROM.

2.4.3 The Quantum Random Oracle Model (QROM)

To show that an encryption scheme is secure against quantum computers, we must of course show that the underlying mathematical problem seems to remain hard to solve even for quantum computers, but we should also take another look at the reduction. The *quantum random oracle model* aims to model the situation where the adversary has access to a quantum computer, by allowing the adversary to send quantum states to the random oracle and receive the evaluated quantum state in reply. This leads to some complications, since proofs using the random oracle model often make use the extractability and adaptive programmability of the model. In the quantum random oracle model these properties do not work quite as well, because the adversary may query the oracle on an exponential number of states in superposition, and it is not clear which is the actual query. Moreover, the adversary may get information about exponentially many states right at the beginning, making it difficult for the simulator to program the oracle adaptively.

As a result, reductions in the random oracle model do not automatically carry over to the quantum random oracle model.

2.4.4 Security notions

There are many different notions of security, of varying strength. One very weak notion of security is *one-wayness*, defined by Fujisaki and Okamoto in [18] as an adversary being unable to completely decrypt the encryption of a random plaintext. This is also called OW-CPA, One-Wayness under Chosen Plaintext Attacks. Another variant is OW-PCA, One-Wayness under Plaintext Checking Attacks, which is OW-CPA except that the adversary is assumed to have access to an oracle that takes a plaintext m and a ciphertext c and returns 1 if m is the decryption of c.

Definition 2.20. Let PKE=(KeyGen, Enc, Dec) be a public key encryption scheme with message space $\mathcal{M}$. The OW-CPA *advantage of an adversary A against* PKE $\mathbf{Adv}^{\text{OW-CPA}}_{\text{PKE}}(A)$ is defined as the probability that A, given access only to the public information about PKE, can completely decrypt $c \leftarrow \text{Enc}(pk, m)$ where $(pk, sk) \leftarrow \text{KeyGen}()$ and m is sampled at random from $\mathcal{M}$.

One can also talk of security in terms of indistinguishability under various types of attack. Security notions are often defined in terms of games, with a challenger and an adversary. In the indistinguishability games for PKEs, the adversary chooses two messages m_0 and m_1 from the message space, and the challenger encrypts one of these and returns the ciphertext c. If the adversary cannot tell with a greater success rate than a random guess of which message c is the encryption, the system is considered *secure in terms of indistinguishability*. The idea is that the adversary, or anyone who does not have the private key, should learn nothing from the ciphertext. The adversary always has access to public information, i.e., the algorithms and the public key, and may also have

access to certain oracles depending on the attack. The time allowance is limited: since any cryptosystem could be broken with brute force given sufficient time, the adversary may only use a polynomially bounded amount of time (meaning that anything that takes time - oracle queries, any kind of computations - must be polynomially bounded in number). The encryption oracle is public, so the adversary is allowed to use it.

Chosen Plaintext (CPA). An encryption scheme is IND-CPA (INDistinguishability under Chosen Plaintext Attack) secure if the adversary cannot distinguish, in polynomial time and with probability significantly greater than 1/2, between the ciphertexts of two messages from the message space.

Definition 2.21. Let PKE=(KeyGen, Enc, Dec) be a public key encryption scheme with message space $\mathcal{M}$. The IND-CPA *advantage of an adversary A against* PKE is defined as $\mathbf{Adv}^{\text{IND-CPA}}_{\text{PKE}}(A) := |p-1/2|$ where p is the probability that an adversary A, given access only to the public information about PKE, can correctly determine whether $b = 0$ or $b = 1$ where

- b is chosen uniformly at random from $\{0, 1\}$
- $c \leftarrow \text{Enc}(pk, m_b)$
- $(pk, sk) \leftarrow \text{KeyGen}()$
- m_0 and m_1 are chosen by A from $\mathcal{M}$

Note that there are only two possible values for b, and we are trying to measure how much better A can do than simply guessing b at random. This is why $\mathbf{Adv}^{\text{IND-CPA}}_{\text{PKE}}(A)$ is defined as $|p - 1/2|$ rather than simply p.

Chosen Ciphertext (CCA1). Here the adversary has access to a decryption oracle, but may only use it before choosing m_0 and m_1. An encryption scheme is IND-CCA1 (INDistinguishability under Chosen Ciphertext Attack) secure if the adversary cannot distinguish, in polynomial time and with probability significantly greater than 1/2, between the ciphertexts of two messages from the message space.

Adaptive Chosen Ciphertext (CCA2 or CCA). Here the adversary is allowed to call on the decryption oracle even after receiving the ciphertext from the challenger, with the important limitation that once the adversary has received the ciphertext c^* the decryption oracle will not work for this particular ciphertext. An encryption scheme is IND-CCA2 (INDistinguishability under Adaptive Chosen Ciphertext Attack) secure if the adversary cannot distinguish, in polynomial time and with probability significantly greater than 1/2, between the ciphertexts of two messages from the message space.

These definitions work for PKEs. In KEMs the message is chosen by the encapsulation algorithm and is simply used to generate a ciphertext and a shared

Algorithm 2 IND-CCA game for KEMs

1: Challenger runs the key generation algorithm to get the secret and the public keys.
2: Challenger chooses b randomly from $\{0, 1\}$.
3: Challenger runs the encapsulation algorithm on the public key to get ciphertext c and shared secret K_0.
4: Challenger chooses K_1 randomly from the key space.
5: Challenger publishes the public key, c and K_b
6: Adversary performs computations in polynomial time, including calls to a decapsulation oracle that will not work on c.
7: Adversary returns a guess $b' \in \{0, 1\}$.
8: If $b' = b$, the adversary wins.

secret. In [39] indistinguishability for KEMs is defined using the game shown in Algorithm 2.

The KEM is IND-CCA secure if the adversary's chance of winning is no better than a random guess.

Definition 2.22. Let KEM=(KeyGen, Encaps, Decaps) be a public key encryption scheme with key space $\mathcal{K}$. The IND-CCA *advantage of an adversary A against* KEM is defined as $\mathbf{Adv}^{\text{IND-CCA}}_{\text{KEM}}(A) := |p - 1/2|$ where p is the probability that an adversary A, given access to the public information about KEM and a decapsulation oracle, can correctly determine whether $b = 0$ or $b = 1$ where

- b is chosen uniformly at random from $\{0, 1\}$
- $(K_0, c) \leftarrow \text{Encaps}(pk)$
- $(pk, sk) \leftarrow \text{KeyGen}()$
- K_1 is sampled at random from $\mathcal{K}$.

For a PKE to be secure under any of these indistinguishability notions, the encryption algorithm has to be probabilistic. Otherwise the adversary need only run the encryption algorithm on both m_0 and m_1 and see which is encrypted to c (whereas with a probabilistic encryption algorithm the encryption of a message will be different every time). Of course, in reality, someone trying to break the cryptosystem will usually not know that the ciphertext is the encryption of one out of only two known messages, so as long as the message space is large enough that encrypting every element of it is not feasible in a reasonable amount of time a deterministic encryption algorithm is not necessarily insecure. This assumes that the adversary does not already have an idea what messages to expect, which may not be the case unless the message is chosen randomly (e.g. a key).

The encapsulation algorithm of a KEM, however, can never be deterministic, because it takes only the public key as input. However, many KEMs are built from PKEs, and then the encapsulation algorithm basically chooses a random

message m, generates a shared secret from that, and encrypts m with the encryption algorithm of the PKE to get the ciphertext. This encryption algorithm can be deterministic, as long as the choice of m is not. This way, a KEM can be for instance IND-CCA secure even if the underlying PKE has a deterministic encryption algorithm and is only OW-CPA secure.

2.4.5 The Fujisaki-Okamoto transform

In 1999 Fujisaki and Okamoto published an article [17] with a transform that made it easier to construct an IND-CCA2-secure PKE (a revised version [18] was published in 2013). Instead of constructing a PKE and then having to prove that it fulfilled this security notion, one could take an OW-CPA-secure PKE and use the Fujisaki-Okamoto transform to obtain a PKE which, in the classical random oracle model, would be IND-CCA2-secure. This was useful, since it is easier to construct a OW-CPA-secure scheme (and prove its security) than one that is IND-CCA2-secure.

Specifically, the FO transform taken a one-way secure (OW-CPA-secure) asymmetric encryption scheme $\mathcal{E}^{\text{asy}}_{pk}(\text{message; coins})$, a one-time secure[7] symmetric encryption scheme $\mathcal{E}^{\text{sy}}_{a}(\text{message})$ (where a is the private key used) and two hash functions G and H, and gives a hybrid encryption scheme where the message m is encrypted as the concatenation

$$\mathcal{E}^{\text{hy}}_{pk}(m;\sigma) = \mathcal{E}^{\text{asy}}_{pk}\big(\sigma; H(\sigma, c)\big)||\mathcal{E}^{\text{sy}}_{G(\sigma)}(m),$$

where $c = \mathcal{E}^{\text{sy}}_{G(\sigma)}(m)$ and σ is a random string[8]. With the additional assumption that the asymmetric scheme is $\omega(\log k)$-spread, meaning that any plaintext in its message space has at least $2^{\omega(\log k)}$ possible ciphertexts (for some sufficiently large k), $\mathcal{E}^{\text{hy}}_{pk}$ is IND-CCA2 secure under the random oracle model.

More recently, Targhi and Unruh gave a variant of the transform which yielded a PKE that what IND-CCA2-secure against a quantum adversary in the QROM. However, both the FO and the TU transforms assume that the input PKE has perfect correctness, which lattice-based PKEs tend not to have. (They can, as Hofheinz, Hövelmanns and Kiltz point out in [19], be made perfectly correct by adjusting the parameters, but not without also making them either less secure or less efficient.)

Remember that a reduction from problem X to problem Y is an algorithm $\mathcal{R}$ that solves problem X by using as a subroutine an adversary $\mathcal{A}$ against problem Y. In the context of the FO and TU transforms, X is the OW-CPA-secure PKE, and Y is the IND-CCA2-secure PKE obtained by applying the transform. If the reduction is not tight, problem Y may be easier to solve than problem X, and sometimes X may need to be made harder in order for the reduction to actually say anything meaningful about the hardness of Y. In practice, this may mean that the parameters must be chosen with extra margin for security,

[7] [18] defines one-time security as an adversary being unable to distinguish from each other the encryptions under a one-time private key of two plaintexts m_0 and m_1.

[8] In the original paper [17], m was used in place of c in $H(\sigma, c)$.

whereas if the reduction were tight smaller parameters could be used, making the cryptosystem more efficient.

Unfortunately, the security reduction of the FO transform is not tight which means security parameters must be adjusted accordingly. There is a variant which is tight, but it requires the input PKE be OW-PCA secure which is not common in lattice-based schemes.

In [19], Hofheinz, Hövelmanns and Kiltz give a collection of transformations more suitable for lattice-based schemes, in that they do not (all) require the input PKE to be OW-PCA secure, and moreover do not require the input PKE to be perfectly correct. Using one of these transformations, or two in combination, will transform a PKE with weaker security (usually OW-CPA or IND-CPA) to an IND-CCA-secure KEM. The security reductions for most of these transforms are tight in the classical random oracle model, and some hold in the quantum random oracle model (though in that model none of the reductions are tight). Combining the two which are secure in QROM and tight in ROM transforms an IND-CPA PKE and three hash functions into an (quantum) IND-CCA secure KEM, by way of an OW-PCA-secure PKE. This combination is used in many of the lattice-based schemes submitted to the recent NIST call for proposals.

All transformations in [19] take a PKE as input. The authors comment that it might be useful to have transforms that instead take a KEM as input as it might be more efficient.

2.4.6 Modular FO transformations

Many of the IND-CCA-secure KEMs submitted to NIST (conveniently listed in Table 1 of [22] by Jiang et al.) use some (variant of a) transform from [19], the most popular being $\mathrm{QFO}^{\perp}[\mathrm{PKE}, G, H, H'] = (\mathrm{Gen}, \mathrm{QEncaps}, \mathrm{QDecaps}^{\perp})$ and $\mathrm{QFO}^{\not\perp}[\mathrm{PKE}, G, H, H'] = (\mathrm{Gen}^{\not\perp}, \mathrm{QEncaps}, \mathrm{QDecaps}^{\not\perp})$, where PKE= (Gen, Enc, Dec) has message space $\mathcal{M} = \{0,1\}^n$ and randomness space $\mathcal{R}$, and

- $G : \mathcal{M} \to \mathcal{R}, H : \{0,1\}^* \to \{0,1\}^n$ and $H' : \{0,1\}^n \to \{0,1\}^n$ are hash functions;
- $\mathrm{Gen}^{\not\perp}$ returns $(pk, sk' := (sk, s))$ where $(pk', sk') \leftarrow \mathrm{Gen}$ and s is sampled uniformly from $\mathcal{M}$;
- $\mathrm{QEncaps}(pk)$ returns $(K := H(m, c), c := \mathrm{Enc}(pk, m; G(m)), d := H'(m))$, where m is sampled uniformly from $\mathcal{M}$;
- $\mathrm{QDecaps}^{\perp}(sk, c, d)$ returns, for $m' = \mathrm{Dec}(sk, c)$, the secret $K := H(m', c)$ if $c = \mathrm{Enc}(pk, m'; G(m'))$ and $d = H'(m')$, and $\perp$ otherwise;
- $\mathrm{QDecaps}^{\not\perp}(sk' = (sk, s), c, d)$ returns, for $m' = Dec(sk, c)$, $K := H(m', c)$ if $c = \mathrm{Enc}(pk, m'; G(m'))$ and $d = H'(m')$, and $K := H(s, c, d)$ otherwise.

These transforms are slightly different than those actually given in [19], $\mathrm{QFO}_m^{\perp}$ and $\mathrm{QFO}_m^{\not\perp}$, where the shared secret is simply $K := H(m)$, rather than $K := H(m, c)$.

If PKE is OW-CPA-secure, the KEMs obtained by $\mathrm{QFO}^{\perp}$ and $\mathrm{QFO}^{\not\perp}$ are IND-CCA-secure in the QROM, but the QROM reductions given in [19] (for $\mathrm{QFO}^{\perp}_m$ and $\mathrm{QFO}^{\not\perp}_m$) are highly non-tight: the success probability of the reduction $\mathcal{R}$ against the security of the KEM is $\epsilon \leq 8q_{RO}\sqrt{\delta q_{RO}^2 + q_{RO}\sqrt{\epsilon'}}$, where ϵ' is the success probability of an adversary $\mathcal{A}$ against the security of the PKE, δ is the probability of decryption error in the PKE, and q_{RO} is a bound on the number of queries made by $\mathcal{R}$ to quantum random oracles. (The running time of $\mathcal{A}$ is about that of $\mathcal{R}$.)

The ROM security reductions for $\mathrm{QFO}^{\perp}_m$ and $\mathrm{QFO}^{\not\perp}_m$ are tight if PKE is IND-CPA-secure, but not if it is only OW-CPA-secure.

Another transform from [19] which is fairly popular among the NIST submissions is $\mathrm{FO}^{\not\perp}[\mathrm{PKE}, G, H] = (\mathrm{Gen}^{\not\perp}, \mathrm{Encaps}, \mathrm{Decaps}^{\not\perp})$, where PKE, G, H and $\mathrm{Gen}^{\not\perp}$ are as above, and

- $\mathrm{Encaps}(pk)$ returns $(K := H(m, c), c := \mathrm{Enc}(pk, m; G(m)))$, where m is sampled uniformly from $\mathcal{M}$;
- $\mathrm{Decaps}^{\not\perp}(sk' = (sk, s), c)$ returns, for $m' = Dec(sk, c)$, $K := H(m', c)$ if $c = \mathrm{Enc}(pk, m'; G(m'))$, and $K := H(s, c)$ otherwise.

No reduction showing QROM security is given in [19] for $\mathrm{FO}^{\not\perp}$, and (as for $\mathrm{QFO}^{\perp}_m$ and $\mathrm{QFO}^{\not\perp}_m$) the ROM reduction is tight if PKE is IND-CPA-secure, but not if it is only OW-CPA-secure.

The only difference between $\mathrm{QFO}^{\not\perp}$ and $\mathrm{FO}^{\not\perp}$ is the extra hash $d = H'(m)$ (where H' is length preserving) that is found in the former but not the latter. This extra hash is also present in $\mathrm{QFO}^{\perp}_m$ and $\mathrm{QFO}^{\not\perp}_m$ and is used in the QROM reduction. Note that d must be sent, so in practice it must be included in the ciphertext.

All these transforms reencrypt the decrypted message m' to ensure c was in fact the encryption of m', because they are a combination of a transform T that produces an OW-PCA secure PKE and one of several transforms that then produces an IND-CCA-secure KEM. However, the reencryption is done in T and two of the partial transforms given in [19] that supposedly produce an IND-CCA-secure KEM (specifically, $\mathrm{U}^{\not\perp}_m$ and $\mathrm{U}^{\perp}_m$) do not actually require the input PKE to be OW-PCA-secure, suggesting that they could be used without T to obtain an IND-CCA-secure KEM from a deterministic PKE which is OW-CPA-secure (or OW-VA-secure, where the adversary has access to an oracle that checks whether c is a valid ciphertext). Omitting the reencryption would of course save time in the decapsulation, but Bernstein and Persichetti give counterexamples in the appendix of [10] to the security claims of these two transforms, using the fact that an OW-CPA-secure (or OW-VA-secure) PKE is not necessarily what they call "rigid", i.e., $\mathrm{Dec}(sk, c) = m$ if and only if $\mathrm{Enc}(pk, m) = c$ (where m is in the message space of a deterministic PKE= (Gen, Enc, Dec) and $(pk, sk) \leftarrow \mathrm{Gen}$). However, as Bernstein and Persichetti point out, the PKE obtained by applying T to an OW-CPA-secure PKE is rigid, so these counterexamples do not affect the security claims of the composite transforms like $\mathrm{FO}^{\not\perp}$, $\mathrm{QFO}^{\perp}$ and $\mathrm{QFO}^{\not\perp}$.

2.4.7 Tighter QROM security

Saito, Xagawa and Yamakawa give two transforms in [37] (similarly to [19], these are composites of modular transforms), both producing an IND-CCA-secure KEM, one from an OW-CPA-secure deterministic PKE and the other from an IND-CPA-secure PKE. The transforms themselves are similar to $\text{QFO}_m^{\not\perp}$ (though the one starting from an OW-CPA-secure deterministic PKE omits the extra hash d), but unlike those given in [19] the reductions require the underlying PKEs to be perfectly correct. Unfortunately this is rarely the case for lattice based cryptosystems.

In [22], Jiang et al. point out that 18 of the 25 IND-CCA-secure KEMs submitted to NIST are not perfectly correct, and that since correctness errors affect the security of the scheme, it is relevant to consider QROM security for transformations that allow correctness errors. They show that the KEMs produced by transforms $\text{FO}^{\not\perp}$ and $\text{FO}_m^{\not\perp}$ in [19] (where $\text{FO}_m^{\not\perp}$ is essentially the same as $\text{FO}^{\not\perp}$, except $K := H(m)$) are IND-CCA-secure in the QROM, meaning that we can omit the extra hash d to get a smaller ciphertext and still have IND-CCA-security in the QROM. Moreover, the reductions for $\text{FO}^{\not\perp}$ and $\text{FO}_m^{\not\perp}$ in [22] (Theorems 1 and 2) are tighter than those given in [19] (Theorem 4.4 combined with 4.5 or 4.6 respectively) for $\text{QFO}_m^{\perp}$ and $\text{QFO}_m^{\not\perp}$: the success probability of the reduction $\mathcal{R}$ against the security of the KEM produced by the transforms in [22] is $\epsilon \lessapprox q_{RO}\sqrt{\delta} + q_{RO}\sqrt{\epsilon'}$, where ϵ' is the success probability of an adversary $\mathcal{A}$ against the OW-CPA-security of the PKE, δ is the probability of decryption error in the PKE, and q_{RO} is a bound on the number of queries made by $\mathcal{R}$ to quantum random oracles. (Approximately, because there is also a term on the right hand side which is either $q_{RO}/|\mathcal{M}|$ for $\text{FO}^{\not\perp}$, or the success probability of an adversary against the pseudorandom function used in $\text{FO}_m^{\not\perp}$.)

One of the NIST submissions [39] mentions the proofs of [22], but since it was then only an eprint and had not been posted for long, the authors of [39] wrote that it was too early to know if the results were correct. The paper has now been published, however, and so far there seem to be no counterexamples. Moreover, in [20] Hövelmanns et al. modify the proof of one of the transformations given in [37] which is similar to $\text{FO}_m^{\not\perp}$ to allow for small non-zero decryption errors, and the reduction is about as tight as that in [22].

2.4.8 Different notions of correctness

Several NIST submissions (e.g. Frodo, Kyber and Saber [39, 42, 43]) define a KEM to be δ-correct if $\Pr[\text{Decaps}(sk, c) \neq K \mid K \leftarrow \text{Encaps}(pk)] \leq \delta$, where the probability is taken over $(pk, sk) \leftarrow \text{KeyGen}()$ and the random coins of Encaps. However, [19] uses a different notion of correctness.

Definition 2.23. A PKE with is *δ-correct* if

$$\mathbf{E}[\max_{m \in \mathcal{M}} \Pr[\text{Dec}(c, sk) \neq m \mid c \leftarrow \text{Enc}(m, pk)]] \leq \delta$$

where the expected value is taken over all $(pk, sk) \leftarrow \text{KeyGen}()$.

In [19] this is equivalently expressed as a game where it is apparent that with this definition of correctness a PKE is δ-correct if an adversary A with access to both the public and the secret keys (which are generated with KeyGen), actively trying to find a message m that will result in a decryption error will succeed with probability δ. Moreover, a PKE that uses a random oracle G is defined as $\delta(q_G)$-correct if $\delta(q_G)$ bounds the probability that an adversary A with access to G and to $(pk, sk) \leftarrow$KeyGen can find a message m such that $\mathrm{Dec}(sk, \mathrm{Enc}(pk, m)) \neq m$.

The other two articles discussed here concerning transforms that allow decryption errors, [22, 20], also define δ-correctness as in definition 2.23. As a result, any NIST submission using the notion of correctness seen in [39, 42, 43] but citing [19] or [22] in support of the IND-CCA security of their KEMs are using two different notions of correctness. Since [19] says that their definition of correctness "has been carefully crafted such that it is sufficient to prove our main theorems (i.e., the security of the Fujisaki-Okamoto transformation)", this would affect the theoretical security claims of such NIST submissions.

Moreover, at least two of the NIST submissions (Kyber and Saber), applying $\mathrm{FO}^{\not\perp}$ to an IND-CPA secure PKE to obtain an IND-CCA secure KEM, claim that by [19] the probability of decapsulation errors in their respective KEMs is the same as the probability of decryption errors in the respective underlying PKEs (see Section 5 of [43] and Section 4 of [42]). However, with the definition of correctness used in [19], this is not necessarily true because the transform T (which derandomises and reencrypts, and is a part of $\mathrm{FO}^{\not\perp}$ and other transforms in [19]) does not preserve correctness.

T takes as input a public key encryption scheme PKE = (KeyGen,Enc,Dec) with message space $\mathcal{M}$ and randomness space $\mathcal{R}$, and a hash function $G : \mathcal{M} \to \mathcal{R}$, and outputs a public key encryption scheme $\mathrm{PKE}_1 = (\mathrm{KeyGen}, \mathrm{Enc}_1, \mathrm{Dec}_1)$, where $\mathrm{Enc}_1(pk, m) = \mathrm{Enc}(pk, m; G(m))$, and for $m' = \mathrm{Dec}(sk, c)$,

$$\mathrm{Dec}_1(sk, c) = \begin{cases} \perp & \text{if } m' = \perp \text{ or } \mathrm{Enc}(pk, m'; G(m')) \neq c \\ m' & \text{otherwise.} \end{cases}$$

Theorem 3.1 of [19] proves that if PKE is δ-correct (in the sense used in that article) then $\mathrm{PKE}_1 = \mathrm{T}[\mathrm{PKE}, G]$ is δ_1-correct (in the ROM) with $\delta_1(q_G) = q_G \cdot \delta$. The proof depends on the fact that since we assume PKE is δ-correct each of the at most q_G distinct queries $G(m_1), ..., G(m_{q_G})$ has, in the ROM, probability at most δ of producing a decryption error. Thus, with at most q_G queries to G, PKE_1 is $(q_G \cdot \delta)$-correct.

In the QROM, Lemma 4.3 of [19] shows that if PKE is δ-correct then $\mathrm{PKE}_1 = \mathrm{T}[\mathrm{PKE}, G]$ is δ_1-correct with $\delta_1(q_G) \leq 8 \cdot (q_G + 1)^2 \cdot \delta$.

Neither Jiang et al. in [22] nor Hövelmanns et al. in [20] make any claims as to whether their versions of the $\mathrm{FO}^{\not\perp}$ transform preserves correctness. (Saito et al. in [37] assume the underlying PKE is perfectly correct, so naturally the question would not arise.)

3 Lattice based cryptography

3.1 SIS

Lattice-based cryptosystems do not tend to be based on SVP and CVP directly, but are instead based on certain problems more convenient than SVP and CVP for constructing cryptosystems, but whose hardness rests on that of some variant of SVP or CVP. One of these problems is the *short integer solution problem* (SIS), which was introduced by Ajtai in 1996 and has been used, among other things, for hash functions and digital signatures, but not for public-key encryption. More details on SIS can be found in e.g. [30], but an overview is given here.

In the following definition, we assume that the matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is *uniformly random*. This means that each element in the matrix is sampled independently from $\mathbb{Z}_q$ according to the uniform distribution.

Definition 3.1. (Short Integer Solution, $\text{SIS}_{n,q,\beta,m}$) Let n, q, m be positive integers and β a positive real. Given a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ with column vectors $\mathbf{a}_i$, find a nonzero vector $\mathbf{z} \in \mathbb{Z}^m$ such that $||\mathbf{z}|| \leq \beta$ and

$$\mathbf{Az} = \sum_{i=1}^{m} \mathbf{a}_i \cdot z_i = \mathbf{0} \mod q.$$

Equivalently, $\text{SIS}_{n,q,\beta,m}$ is the problem of finding a short vector of length at most β in the lattice $\Lambda_q^{\perp}(\mathbf{A})$, for a uniformly random $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$.

The SIS problem does not become harder for larger m, because the column $\mathbf{a}_i$ of $\mathbf{A}$ can be ignored simply by choosing $z_i = 0$, for any i (but of course not all at once, since the solution must be a nonzero vector).

Note that β must not be too large, because unless $\beta < q$, $\mathbf{z} = (q, 0, ..., 0) \in \mathbb{Z}^m$ is always a valid solution. On the other hand, β and m must be large enough that a solution is guaranteed to exist; it suffices that $\beta \geq \sqrt{\lceil n \log q \rceil}$ and $m \geq \lceil n \log q \rceil$, because then by the above we can assume $m = \lceil n \log q \rceil$ and so there are q^n vectors of the form $\{0, 1\}^m$. By the pigeonhole principle there must be two such vectors $\mathbf{x}, \mathbf{x}'$ such that $\mathbf{Ax} = \mathbf{Ax}' \in \mathbb{Z}_q^n$. Their difference has length at most β and is a solution to $\text{SIS}_{n,q,\beta,m}$.

Hermite normal form. If q is prime, a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ has full rank with high probability. Permuting the columns does not change the SIS problem, except that the solutions will also be permuted, so we can permute the columns of $\mathbf{A}$ so that the first n columns are linearly independent over $\mathbb{Z}_q$. Thus assuming that $\mathbf{A}$ has full rank, we can assume without further loss of generality that $\mathbf{A} = (\mathbf{A}_1 | \mathbf{A}_2)$, where $\mathbf{A}_1$ is square and invertible over $\mathbb{Z}_q$. We can then replace $\mathbf{A}$ with $\mathbf{A}_1^{-1}\mathbf{A} = (\mathbf{I}_n | \mathbf{A}_1^{-1}\mathbf{A}_2)$, which is in upper triangular Hermite normal form.[9] This has the same set of SIS solutions as

[9] More accurately, if we see the entries of this matrix simply as integers between 0 and q, then it is in Hermite normal form.

$\mathbf{A}$, and because $\mathbf{A}_2$ is uniformly random and independent of $\mathbf{A}_1$, $\mathbf{A}_1^{-1}\mathbf{A}_2$ is uniformly random. Therefore the SIS problem is at least as hard to solve given the matrix $(\mathbf{I}_n|\mathbf{A}_1^{-1}\mathbf{A}_2)$ as it is with the matrix $\mathbf{A}$.

This is convenient if $\mathbf{A}$ needs to be stored or sent, as it means that $\mathbf{A}_1^{-1}\mathbf{A}_2 \in \mathbb{Z}_q^{n\times(m-n)}$ can be used instead of the larger matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ ($\mathbf{I}_n$ being treated as implicit).

SIS does not require q to be prime, but even if it is not the likelihood of $\mathbf{A}$ having full rank remains high, at least as long as q is the product of reasonably large primes. For any selection of n columns of $\mathbf{A}$, the square matrix $\mathbf{A}'$ consisting of those columns is invertible if (and only if) $\det \mathbf{A}' \in \mathbb{Z}_q^\times$, which, if all possible values of the determinant are equally likely, it is with probability $\phi(q)/q$, where ϕ is Euler's totient function. For $\mathbf{A}$ to be full rank, it suffices that the matrix consisting of *any choice* of n columns of $\mathbf{A}$ (out of out of $m \geq \lceil n \log q \rceil$) is invertible.

If, in the worst case, a uniformly random $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ should not be full rank, presumably in implementations it would be discarded and a new matrix sampled.

3.1.1 Hardness of SIS

Ajtai proved that solving the average-case SIS problem is at least as hard as solving various approximate lattice problems in the worst case. This result has since been strengthened, in that it has been shown to hold for tighter bounds for the modulus q and the the approximation factor γ. Incorporating these tighter bounds in Theorem 4.1.2 of [30], we have the following theorem.

Theorem 3.1. *For any $m = \mathrm{poly}(n)$, any $\beta > 0$ and any $q \geq \beta \cdot n^\epsilon$, where $\epsilon > 0$, solving $SIS_{n,q,\beta,m}$ with non-negligible probability is at least as hard as solving $GapSVP_\gamma$ and $SIVP_\gamma$ for some $\gamma = \beta \cdot \tilde{O}(\sqrt{n})$, on arbitrary n-dimensional lattices, with overwhelming probability.*

For more details, see Section 3 (in particular Theorem 3.8) of [29].

Note that since solving GapSVP_γ and SIVP_γ on arbitrary lattices means solving them in the worst case, whereas SIS is a problem over a random lattice, so it is an average-case problem. The theorem therefore states that there is a worst-case to average-case reduction, i.e., if there is an efficient algorithm for solving the *average-case* problem SIS, then there is also an efficient algorithm for solving the problems GapSVP_γ and SIVP_γ *in the worst case*.

3.2 NTRU

As we have seen, lattice based schemes can be as hard to break as certain hard lattice problems on general lattices, but a downside is that the public key tends to be the entire matrix $\mathbf{A}$, which is large and inconvenient to send, or at least some seed from which $\mathbf{A}$ must then be pseudorandomly generated, which slows down the encryption. It is possible to get faster schemes and smaller key sizes by restricting the problems to some more structured group of lattices, especially

since, for some choices of restricted lattice, the problems can be presented using polynomial rings instead of matrices and vectors. Such a restricted version of SIS (see Section 3.3) was presented in 2002, inspired by the NTRU scheme.

NTRU is a public-key encryption scheme by Hoffstein, Pipher and Silverman, published in 1998, which uses polynomial rings. It can also be described using specially structured lattices, as Micciancio and Regev do in [28].

Let $\mathbf{T}$ be the $n \times n$ matrix

$$\mathbf{T} = \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix} = \left(\begin{array}{c|c} \mathbf{0}^T & 1 \\ \hline \mathbf{I} & \mathbf{0} \end{array}\right),$$

and define $\mathbf{T}^*\mathbf{v} = (\mathbf{v}, \mathbf{T}\mathbf{v}, \ldots, \mathbf{T}^{n-1}\mathbf{v})$, for a vector $\mathbf{v} \in \mathbb{Z}^n$. This is called a *cyclic matrix*.

The NTRU cryptosystem has parameters n, q, p and d, all integers, where the dimension n is prime, p should be small, and it is recommended that q be a power of 2. The private key is a vector $\begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix} \in \mathbb{Z}^{2n}$, where $\mathbf{f} \in \mathbf{e}_1 + \{p, 0, -p\}^n$ and $\mathbf{g} \in \{p, 0, -p\}^n$ are randomly chosen such that $\mathbf{f} - \mathbf{e}_1$ and $\mathbf{g}$ each have $d+1$ positive entries and d negative entries (and all remaining entries equal to 0). Additionally, $(\mathbf{T}^*\mathbf{f})$ must be invertible modulo q.

Having chosen these vectors $\mathbf{f}$ and $\mathbf{g}$, we have the q-ary lattice

$$\Lambda_q\Big(\begin{pmatrix} \mathbf{T}^*\mathbf{f} \\ \mathbf{T}^*\mathbf{g} \end{pmatrix}^T\Big) = \{\mathbf{y} \in \mathbb{Z}^{2n} : \mathbf{y} = \begin{pmatrix} \mathbf{T}^*\mathbf{f} \\ \mathbf{T}^*\mathbf{g} \end{pmatrix} \mathbf{s} \bmod q \text{ for some } \mathbf{s} \in \mathbb{Z}^n\}.$$

We say that this is a *cyclic lattice*, and it has a basis (in Hermite normal form)

$$\mathbf{H} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{T}^*\mathbf{h} & q \cdot \mathbf{I} \end{pmatrix} \qquad \text{where } \mathbf{h} = (\mathbf{T}^*\mathbf{f})^{-1}\mathbf{g} \pmod q,$$

which can be represented simply by the vector $\mathbf{h} \in \mathbb{Z}_q^n$. This is the public key.

A message is encrypted by first encoding it as a vector $\mathbf{m} \in \{1, 0, -1\}^n$ with $d+1$ positive and d negative entries. A random vector $\mathbf{r} \in \{1, 0, -1\}^n$ is chosen, also with $d+1$ positive and d negative entries. Concatenating these gives a short error vector $\begin{pmatrix} -\mathbf{r} \\ \mathbf{m} \end{pmatrix} \in \{1, 0, -1\}^{2n}$, which can be reduced modulo $\mathbf{H}$ to

$$\begin{pmatrix} -\mathbf{r} \\ \mathbf{m} \end{pmatrix} \bmod \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{T}^*\mathbf{h} & q \cdot \mathbf{I} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ (\mathbf{m} + (\mathbf{T}^*\mathbf{h})\mathbf{r}) \bmod q \end{pmatrix}.$$

The ciphertext is the vector $\mathbf{c} = \mathbf{m} + (\mathbf{T}^*\mathbf{h})\mathbf{r} \in \mathbb{Z}_q^n$.

To decrypt $\mathbf{c} \in \mathbb{Z}_q^n$, the ciphertext is multiplied by $(\mathbf{T}^*\mathbf{f})$. Because for any vectors $\mathbf{x}$ and $\mathbf{y}$ $(\mathbf{T}^*\mathbf{x})(\mathbf{T}^*\mathbf{y}) = \mathbf{T}^*(\mathbf{T}^*\mathbf{x})\mathbf{y}$, this multiplication yields the vector

$$(\mathbf{T}^*\mathbf{f})\mathbf{c} \bmod q = (\mathbf{T}^*\mathbf{f})\mathbf{m} + (\mathbf{T}^*\mathbf{g})\mathbf{r} \bmod q.$$

In the unlikely event that all the entries of $(\mathbf{T}^*\mathbf{f})$ and $(\mathbf{T}^*\mathbf{g})$ line up perfectly with regards to sign, the entries of $(\mathbf{T}^*\mathbf{f})\mathbf{m} + (\mathbf{T}^*\mathbf{g})\mathbf{r}$ are still bounded above by $4pd + 2p + 1$ and below by $-4pd - 1$. This means that assuming $d < (q/2 - 1)/(4p) - 1/2$, the exact value of the vector $(\mathbf{T}^*\mathbf{f})\mathbf{m} + (\mathbf{T}^*\mathbf{g})\mathbf{r}$, without reduction modulo q, can be recovered. (Even for larger values of d the vector can be recovered with high probability.)

Because of how $\mathbf{f}$ and $\mathbf{g}$ were chosen, $(\mathbf{T}^*\mathbf{f}) \equiv \mathbf{I} \pmod{p}$ and $(\mathbf{T}^*\mathbf{g}) \equiv \mathbf{0} \pmod{p}$, so reducing $(\mathbf{T}^*\mathbf{f})\mathbf{m} + (\mathbf{T}^*\mathbf{g})\mathbf{r}$ modulo p recovers the message $\mathbf{m}$.

There is no known proof of security for the NTRU cryptosystem, but it appears to be hard to recover $\mathbf{m}$ without knowing $\mathbf{f}$. (The probability that any other vector than $\mathbf{f}$ and its rotations will work as a decryption key is very low.)

According to Hoffstein, Pipher and Silverman [21] (Proposition 6.61), the vector $\begin{pmatrix}\mathbf{f}\\ \mathbf{g}\end{pmatrix} \in \mathbb{Z}^{2n}$ is a factor $O(1/\sqrt{n})$ shorter than the the Gaussian heuristic predicts the shortest vector of $\Lambda_q\Big(\begin{pmatrix}\mathbf{T}^*\mathbf{f}\\ \mathbf{T}^*\mathbf{g}\end{pmatrix}^T\Big)$ to be, so a shortest vector in the lattice is likely to be a rotation of $\begin{pmatrix}\mathbf{f}\\ \mathbf{g}\end{pmatrix} \in \mathbb{Z}^{2n}$. Thus, solving SVP_γ in the NTRU lattice for $\gamma \approx n^\epsilon$ where $\epsilon < 1/2$ will probably give a decryption key. This estimate assumes that $d \approx n/3$ and $q \approx 6d \approx 2n$, but more generally we still have

$$\frac{\Big\|\begin{pmatrix}\mathbf{f}\\ \mathbf{g}\end{pmatrix}\Big\|}{\sigma\Big(\Lambda_q\Big(\begin{pmatrix}\mathbf{T}^*\mathbf{f}\\ \mathbf{T}^*\mathbf{g}\end{pmatrix}^T\Big)\Big)} \approx \frac{\sqrt{4d}}{\sqrt{nq/\pi e}} = \sqrt{\frac{4\pi e d}{nq}}.$$

(In the formula for the Gaussian heuristic, keep in mind that $\Lambda_q\Big(\begin{pmatrix}\mathbf{T}^*\mathbf{f}\\ \mathbf{T}^*\mathbf{g}\end{pmatrix}^T\Big)$ has dimension $2n$.)

3.2.1 NTRU with polynomial rings

Another way to describe NTRU is using polynomial rings, and this section gives an idea of how this is done.

Take integers N, p, q such that N is prime and $\gcd(N, q) = \gcd(p, q) = 1$. Let R be the polynomial ring $R = \mathbb{Z}[X]/(X^N - 1)$, and let $R_p = R/pR$ and $R_q = R/qR$. Finally, for positive integers d_1, d_2 let

$$\mathcal{T}(d_1, d_2) = \Big\{a(X) \in R : \begin{array}{l} a(X) \text{ has } d_1 \text{ coefficients equal to } 1 \\ a(X) \text{ has } d_2 \text{ coefficients equal to } -1 \\ a(X) \text{ has all other coefficients equal to } 0 \end{array}\Big\}.$$

The NTRU cryptosystem has public parameters (N, p, q, d). The public key $h(X)$ is constructed by randomly choosing $f(X) \in \mathcal{T}(d+1, d)$ and $g(X) \in \mathcal{T}(d, d)$ such that $F_q(X) = f(X)^{-1}$ in R_q and $F_p(X) = f(X)^{-1}$ in R_p exist.

Then the public key is

$$h(X) = F_q(X) \cdot g(X) \in R_q.$$

For $m(X) \in R$ with coefficients between $-p/2$ and $p/2$ and a random $r(X) \in \mathcal{T}(d, d)$, the ciphertext is

$$c(X) = p \cdot h(X) \cdot r(X) + m(X) \in R_q.$$

To decrypt the message, one then computes $a(X) = f(X) \cdot c(X) \in R_q$, and $b(X) = F_p(X) \cdot a'(X) \in R_p$, where $a'(X)$ is the unique polynomial in R with coefficients in $(-q/2, q/2]$ such that $a'(X) \mod q = a(X)$. Then assuming the parameters were chosen correctly, $b(X)$ and $m(X)$ are equal modulo p.

3.3 SIS over rings

Schemes using SIS have large keys, with the public key typically consisting of the $n \times m$ matrix $\mathbf{A}$ (or the more compact $n \times (m-n)$-matrix $\mathbf{A}_1^{-1}\mathbf{A}_2$ where $\mathbf{A} = (\mathbf{A}_1|\mathbf{A}_2)$), where $n \geq 100$ and $m \geq \lceil n \log q \rceil$. Thus keys take some time to generate and are inconvenient to send, and though vector operations are fast, multiplying large matrices ends up being relatively time consuming. Inspired by NTRU, Micciancio introduced a variant of SIS over more structured lattices which came to be known as *ring-SIS*. This is SIS over a special class of lattices called *ideal lattices*, which correspond to ideals in polynomial rings. The cyclic lattices used in NTRU are a special case of ideal lattices.

In general, RSIS is described using not lattices but polynomial rings, but we will later see the connection to lattices.

Let R be the polynomial ring $R := \mathbb{Z}[X]/(f(X))$ for some polynomial $f(X) \in \mathbb{Z}[X]$ of degree n. For some positive integer q, let $R_q := R/qR = \mathbb{Z}_q[X]/(f(X))$.

To be able to talk about short vectors, we need to define a norm on R. One possible way to do this is to simply associate $z \in R$ with the vector of its coefficients, but then the length of a vector would depend on the choice of representatives of R. For security analysis, it is better to use the *canonical embedding*, which maps $z \in R$ to the vector $(z(\alpha_1), z(\alpha_2), ..., z(\alpha_n)) \in \mathbb{C}^n$, where the α_i are the complex roots of $f(X)$.

We can now define the short integer solution problem over rings.

Definition 3.2. (Short Integer Solution Problem over Rings, $\text{RSIS}_{q,\beta,m}$) Given a uniformly random vector $\mathbf{a} \in R_q^m$, find a nonzero vector $\mathbf{z} \in R^m$ of norm $||\mathbf{z}|| \leq \beta$, for some given $\beta > 0$, such that

$$\mathbf{a}^T\mathbf{z} = 0 \in R_q.$$

(Peikert writes in Section 4.3.1 [30] that in order for a solution to exist, it suffices that $m \approx \log q$, rather than $m \approx n \log q$ as for SIS.)

An early suggestion was to use $f(X) = X^n - 1$, making the corresponding lattice a cyclic lattice as in NTRU. However, it was found that SIS is easy to solve in a cyclic lattice, because multiplying a cyclic matrix with a constant

vector will always yield a constant vector. In this case, it will yield a constant vector in a finite space, where there are only q constant vectors, meaning that by the pigeon hole principle some constant vectors must map, under multiplication with a cyclic matrix, to the same constant vector, and therefore the difference between two such vectors is a nonzero vector that is mapped to zero. Thus we can restrict the problem considerably by only looking at the constant vectors. (It is of course conceivable that all nonzero constant vectors that map to zero have too large a norm to be solutions to the SIS problem, but this seems too unlikely to make it worth the risk.)

This problem, however, does not arise if $f(X) = X^{2^k} + 1$, so this is a more secure and still convenient choice for RSIS. More generally, Micciancio and Regev describe in [28] (though in the context of hash functions), the structured lattice corresponding to a certain choice of $f(X)$. Essentially, an instance of RSIS over $R = \mathbb{Z}[X]/(f(X))$ for $f(X) = X^n + f_n X^{n-1} + ... + f_1$ with a uniformly random $\mathbf{a} \in R_q^m$ corresponds to an instance of SIS for the matrix

$$\mathbf{A} = (\mathbf{A}_1, ..., \mathbf{A}_m),$$

i.e., to finding short vectors in $\Lambda_q^\perp(\mathbf{A})$, where each square matrix $\mathbf{A}_i$ is of the form $\mathbf{F}^*\mathbf{a}_i$ for vectors $\mathbf{a}_1, ..., \mathbf{a}_m \in \mathbb{Z}_q^n$ (chosen independently and uniformly at random) and

$$\mathbf{F} = \begin{pmatrix} 0 & 0 & \cdots & 0 & -f_1 \\ 1 & 0 & \cdots & 0 & -f_2 \\ 0 & 1 & \cdots & 0 & -f_3 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -f_n \end{pmatrix}.$$

If $f(X) = X^n - 1$, $\mathbf{F}$ is equal to the matrix $\mathbf{T}$ used in NTRU, whereas if $f(X) = X^{2^k} + 1$, $\mathbf{F}$ is the $2^k \times 2^k$ matrix

$$\mathbf{F} = \begin{pmatrix} 0 & 0 & \cdots & 0 & -1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$$

so the matrices $\mathbf{A}_i = \mathbf{F}^*\mathbf{a}_i$ are similar to the cyclic matrices of NTRU, except that the elements above the diagonal have changed signs. This seemingly small change makes a big difference: finding short vectors in the lattice $\Lambda_q^\perp(\mathbf{A})$ for $\mathbf{A} = (\mathbf{F}^*\mathbf{a}_1, ..., \mathbf{F}^*\mathbf{a}_{m/n})$ is as hard as solving lattice problems like approximate SVP and SIVP in the worst case over ideal lattices, as long as

- for any two unit vector $\mathbf{u}$ and $\mathbf{v}$, $||(\mathbf{F}^*\mathbf{u})\mathbf{v}||$ is small, typically $O(\sqrt{n})$, and
- $f(X) \in \mathbb{Z}[X]$ is irreducible over the integers.

3.3.1 Hardness of RSIS

RSIS is more efficient than SIS for several reasons.

- If n is the dimension of R over $\mathbb{Z}$, then there is an additive group isomorphism which approximately preserves shortness between R and $\mathbb{Z}^n$, and between R_q and $\mathbb{Z}_q^n$. In this way one element of R corresponds to n (non-independent) integers in $\mathbb{Z}$.
- The domain $\mathbb{Z}_q^n$ used in SIS is treated as a $\mathbb{Z}$-module, while the R_q of RSIS is treated as an R-module. The latter has a richer structure which makes it more efficient.
- RSIS is more compact than SIS, in that the number m of elements a_i needed to guarantee that a short solution exists is only $m \approx \log q$, as opposed to $m \approx n \log q$ for SIS.

On the other hand, the richer structure of R_q that makes RSIS more efficient than SIS has its drawbacks. Because RSIS uses ideal lattices rather than general ones, the hardness proofs of SIS no longer apply. RSIS can still be shown to be as hard as certain lattice problems in the worst case, but these are lattice problems in ideal lattices. In such lattices, GapSVP_γ for small $\gamma = \text{poly}(n)$ is easy, and the problems SVP_γ and SIVP_γ are (almost) equivalent. However, SVP_γ and SIVP_γ still appear to be very hard on the ideal lattices typically used in cryptography, and for relevant choices of γ; usually γ is taken to be polynomial in n. The best known algorithms (including quantum algorithms) for solving SVP_γ in typical ideal lattices and for $\gamma = \text{poly}(n)$ take exponential time, but ideal lattices have not been studied as carefully as general lattices.

3.3.2 SIS over module lattices

In [25], Langlois and Stehlé introduced a variant of SIS over module lattices, with a security reduction to show that this problem is at least as hard as SIVP on module lattices (lattices corresponding to finitely generated modules). In module-SIS as in RSIS we have a ring $R = \mathbb{Z}[X]/(f(X))$ of dimension n, but we work with elements in the module R_q^d of rank d. Thus the following problem corresponds to SIS over a module lattice of dimension nd.

Definition 3.3. (The Shortest Integer Solution Problem for Modules, $\text{MSIS}_{q,m,\beta}$) Given uniformly random $\mathbf{a}_1, ...\mathbf{a}_m \in R_q^d$, find $\mathbf{z} \in R^m$ such that $0 < ||\mathbf{z}|| \leq \beta$ and

$$\sum_{i=1}^{m} \mathbf{a}_i \cdot z_i = 0 \mod q.$$

Again, it is convenient for security analysis to consider the norm in terms of the canonical embedding.

The reduction from SIVP to SIS allows for converse reductions, meaning that while being able to solve SIS in general means being able to solve SIVP in the worst case, the converse also holds. However according to [25] no such result

is known for the reduction from SIVP on ideal lattices to RSIS, so it is possible that SIVP on ideal lattices is easier than RSIS. The reduction from SIVP on module lattices to MSIS (though in fact the reduction in [25] uses GIVP rather than SIVP) does allow converse reductions, meaning that MSIS and SIVP on module lattices are possibly harder than RSIS, which is possibly harder than SIVP on ideal lattices.

3.4 LWE

Another lattice problem whose hardness rests on that of variants of SVP and CVP is the *learning with errors problem*, which, unlike SIS, can be used for public-key encryption. Again, [30] has more details.

There are two versions of the original learning with errors problem: the search problem, where the task is to find a secret vector $\mathbf{s}$, and the decision problem, where the task is to determine whether there even is an $\mathbf{s}$ to find, or whether the given samples are from the uniform distribution. These two problems are polynomially equivalent.

Let χ be a probability distribution on $\mathbb{Z}_q$. By $e \leftarrow \chi$ we mean that e is sampled according to χ.

Definition 3.4. (Search-LWE$_{n,q,\chi,m}$) Given m samples $(\mathbf{a}_i, b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where $\mathbf{s} \in \mathbb{Z}_q^n$ and for $1 \leq i \leq m$, $\mathbf{a}_i \in \mathbb{Z}_q^n$ is uniformly random and $e_i \leftarrow \chi$, find $\mathbf{s}$.

Definition 3.5. (Decision-LWE$_{n,q,\chi,m}$) Given m samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where for $1 \leq i \leq m$, $\mathbf{a}_i \in \mathbb{Z}_q^n$ is uniformly random, determine whether the $b_i \in \mathbb{Z}_q$ are also uniformly random, or whether there is $\mathbf{s} \in \mathbb{Z}_q^n$ such that for every i, $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i$ (where $e_i \leftarrow \chi$).

In practice it is more convenient to sample $\mathbf{s}$ from the error distribution (modulo q) rather than uniformly. This is called *normal form* and is at least as hard as uniform secret (up to a small difference in the number of samples). Though theoretical hardness proofs tend to assume χ is a discrete Gaussian distribution, it is also more convenient not to use this distribution for the errors in practice, since it is inefficient to sample from. Many applications ([39, 40, 41, 44, 45]) use some other distribution which is easier to sample from, sometimes but not always designed to approximate a discrete Gaussian, with the argument that the best known attacks do not depend on the exact distribution of the errors.

Like SIS, the problems can be formulated using linear algebra instead of talking about a collection of samples. Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be the matrix with columns $\mathbf{a}_i$, let $\mathbf{b} \in \mathbb{Z}_q^m$ the vector with entries b_i, and $\mathbf{e} \leftarrow \chi^m$. Then the search-LWE problem is to find $\mathbf{s} \in \mathbb{Z}_q^n$ given

$$\mathbf{b} = \mathbf{A}^T \mathbf{s} + \mathbf{e}.$$

Search-LWE can be seen as average-case BDD on the lattice

$$\mathcal{L}(\mathbf{A}) := \{\mathbf{A}^T \mathbf{s} : \mathbf{s} \in \mathbb{Z}_q^n\} + q\mathbb{Z}^m.$$

In search-LWE, the vector $\mathbf{b}$ will be fairly close to exactly one vector in this lattice, whereas in the uniform case of decision-LWE, $\mathbf{b}$ will in all likelihood be far from all vectors of $\mathcal{L}(\mathbf{A})$.

3.4.1 Hardness of LWE

When introducing the LWE problem, Regev also gave a security reduction showing that LWE is at least as hard as quantumly solving GapSVP and SIVP on arbitrary lattices. This result has been strengthened slightly, and is given in [30] as follows.

Theorem 3.2. *For any $m = \text{poly}(n)$, any modulus $q \leq 2^{\text{poly}(n)}$, and any (discretized) Gaussian error distribution χ of parameter $\alpha q \geq 2\sqrt{n}$ where $0 < \alpha < 1$, solving the decision-LWE$_{n,q,\chi,m}$ problem is at least as hard as* quantumly *solving GapSVP$_\gamma$ and SIVP$_\gamma$ on arbitrary n-dimensional lattices, for some $\gamma = \tilde{O}(n/\alpha)$.*

The proof is by first showing that search-LWE is at least as hard as GapSVP and SIVP via a quantum reduction, and then a classical reduction shows that decision-LWE is equivalent to search-LWE (up to polynomial blow-up of m). The first part is accomplished by using an oracle for search-LWE and a source of discrete Gaussian samples to solve BDD, and then using an oracle for BDD to quantumly generate discrete Gaussian samples with narrower parameter. Iterating will yield sufficiently narrow discrete Gaussian samples to give solutions to GapSVP and SIVP.

There is also a classical reduction by Peikert, proving that LWE with error rate α is at least as hard as GapSVP$_\gamma$ on arbitrary lattices, for $\gamma = \tilde{O}(n/\alpha)$. However, this reduction only works for GapSVP, and moreover it requires $q \geq 2^{n/2}$. (It has since been shown by Brakerski *et al.* that as long as q is bounded below by some small polynomial, hardness for a given error rate α depends mainly on $n \log q$ rather than n and q themselves.)

Another useful quality of LWE is that it is robust, in the sense that even if an attacker has some bounded information about the secret, the problem remains as hard as if the secret was still perfectly secret, albeit for a lower dimension n and error rate α.

3.4.2 LWE over rings

Like for SIS, there is a ring version of the LWE problem, introduced by Lyubashevsky, Peikert and Regev in [23]. Let $R = \mathbb{Z}[X]/(f(X))$ for some $f(X) \in \mathbb{Z}[X]$ of degree n, $R_q = R/qR$ and let χ be an error distribution over R.

Definition 3.6. Given m samples $(a_i, b_i) \in R_q \times R_q$, where the $a_i \in R_q$ are uniformly random, the decisional LWE problem over rings (RLWE$_{q,\chi,m}$) is to determine whether the $b_i \in R_q$ are also uniformly random, or whether $b_i = s \cdot a_i + e_i \bmod q$ for all i, where $s \in R_q$ and $e_i \leftarrow \chi$.

RLWE corresponds to the LWE problem over ideal lattices, and the advantage is the efficiency and compactness compared to LWE. Multiplication in RLWE is quick, and more importantly fewer multiplications are necessary. A cryptosystem using LWE will typically have a large square matrix $\mathbf{A}$ as a public key, meaning that encryption requires multiplying this matrix with a vector. A system using RLWE will just have a polynomial $\mathbf{a}$ as a public key, and encryption requires a single polynomial multiplication. Thus RLWE has significantly smaller keys and faster computations, and also faster key generation.

In the original work the problem was defined over a fractional ideal dual to R, which is more convenient for proofs, but this gets quite technical and is not repeated here. Lyubashevsky, Peikert and Regev showed in [23] that RLWE is as hard as SIVP on ideal lattices, using this other definition of the problem, and Langlois and Stehlé (using the same definition) adapted this using GIVP over ideal lattices instead of SIVP. The theorem here most closely resembles Langlois and Stehlé's Theorem 4.5 in [25].

In the following theorem, $\mathrm{RLWE}_{q,\Upsilon_\alpha}$ is a version of the RLWE problem defined over a fractional ideal dual to R, where $R = \mathbb{Z}[X]/(f(X))$ for the ν-th cyclotomic polynomial $f(X) \in \mathbb{Z}[X]$ of degree $\phi(\nu) = n$ where ϕ is Euler's totient function, and the errors are sampled from an elliptical Gaussian distribution Υ_α. The *smoothing parameter* η_ϵ for a lattice $\mathcal{L}$ and $\epsilon > 0$, η_ϵ is the smallest s such that $\rho_{1/s}(\mathcal{L}^* \backslash \{0\}) \leq \epsilon)$. For more details see [23, 25]

Theorem 3.3. *Let $\epsilon(n) = n^{-\omega(1)}$, $\alpha \in (0,1)$, and prime $q \geq 2$ such that $\nu|(q-1)$, $\alpha q > \omega(\sqrt{\log n})$ and $q \leq \mathrm{poly}(n)$. Then there exists a quantum reduction from solving $GIVP_\gamma^{\eta_\epsilon}$ on ideal lattices (in the worst case, with high probability) with $\gamma = \sqrt{n} \cdot \omega(\sqrt{\log n})/\alpha$ to solving $RLWE_{q,\Upsilon_\alpha}$ in polynomial time with non-negligible probability.*

The quantum reduction is to the search version of RLWE, with a different error distribution than the decision problem, and this has fewer requirements on q (it requires $q \geq 2$ such that the factorisation of q is known and $\alpha q > \omega(\sqrt{\log n})$). With the additional requirements on q, there is a classical polynomial time search-to-decision reduction.

Though the theorem only assumes that the ring R is cyclotomic, in practice many cryptosystems use power-of-two cyclotomics, i.e., they take $f(X) = X^n + 1$ where n is a power of two. This restriction is not strictly necessary but the arguments usually given in favour of this choice are the ease of working with this particular type of ring and the relative efficiency of the multiplication algorithm NTT (see Section 2.2.6) over these rings compared to others. (Though this is not necessarily the best choice of multiplication algorithm for polynomials of the size typically used.)

In [24], Lyubashevsky, Peikert and Regev offer techniques for using arbitrary cyclotomic rings for LWE, with no loss in the underlying worst-case hardness guarantees, and very little loss in efficiency, but of the contributions to the NIST call for proposals almost all that use some variant of RLWE use power-of-two cyclotomic rings. A drawback of this is that it severely limits the possible values

for n as the only power-of-two values in a reasonable interval is $n = 512$ and $n = 1024$, which makes it hard to tightly meet security goals.

As for the error distribution, most cryptosystems will not use a Gaussian distribution, but will instead use some distribution which approximates a discrete Gaussian distribution but is more efficient to sample from.

3.4.3 LWE over module lattices

As previously mentioned, RLWE gets quite inflexible if the polynomial $f(X)$ is taken to be a power of two cyclotomic polynomial, but despite this such polynomials are popular. In 2011, Brakerski, Gentry and Vaikuntanathan introduced a problem that allows for more flexibility while still using power of two cyclotomics (though it is not restricted only to such polynomials). This is the *learning with errors problem over module lattices*.

Let $R = \mathbb{Z}[X]/(f(X))$ for some $f(X) \in \mathbb{Z}[X]$ of degree n, $R_q = R/qR$, χ be an error distribution over R, and d a positive integer.

Definition 3.7. (Decision-MLWE) Given m samples $(\mathbf{a}_i, b_i) \in R_q^d \times R_q$, where the $\mathbf{a}_i \in R_q^d$ are uniformly random, determine whether the $b_i \in R_q$ are also uniformly random, or whether $b_i = \mathbf{a}_i^T \mathbf{s} + e_i \bmod q$ for all i, where $\mathbf{s} \in R_q^d$ and $e_i \leftarrow \chi$.

The underlying lattice has dimension nd, where d is usually small. This makes it possible to scale the problem to a greater extent than is possible with RLWE. For instance, the recommended parameter choices for the MLWE-based KEM Kyber (a submission to the NIST call for proposals) is $n = 256$ and $d = 3$, which gives a lattice of dimension $nd = 768$ (the key that is encapsulated is 256 bits long, and to minimise the risk of decapsulation errors n should therefore not be smaller than 256).

Langlois and Stehlé showed in [25] that MLWE was as hard as GIVP over module lattices. Like the proofs for RLWE this requires a different formulation of the problem, analogous to that of RLWE above. With that definition of the MLWE problem, we have the following theorem.

Theorem 3.4. *(Theorem 4.7, [25]) Let $\epsilon(nd) = (nd)^{-\omega(1)}$, $\alpha \in (0,1)$, and prime $q \geq 2$ such that $\nu|(q-1)$, $\alpha q > 2\sqrt{d} \cdot \omega(\sqrt{\log n})$ and $q \leq \text{poly}(nd)$. Then there exists a quantum reduction from solving $GIVP_\gamma^{\eta_\epsilon}$ on module lattices (in the worst case, with high probability) with $\gamma = \sqrt{8nd^2} \cdot \omega(\sqrt{\log n})/\alpha$ to solving $MLWE_{q,\Upsilon_\alpha}$ in polynomial time with non-negligible probability.*

This is a simplified version of the theorem in [25]; as for RLWE, the quantum reduction is to the search version of MLWE for a different error distribution, and again this reduction requires only of q that $q \geq 2$ such that the factorisation of q is known and $\alpha q > 2\sqrt{d} \cdot \omega(\sqrt{\log n})$). The classical polynomial time search-to-decision reduction requires the additional conditions on q.

It is known that GapSVP_γ, though believed to be a hard problem in general for γ that is polynomial in the security parameter, is easy for ideal lattices.

Langlois and Stehlé comment that this problem should not be easy in the worst case for module lattices (of rank > 1), since this would make it possible to efficiently solve (decisional) RLWE with just two samples, by using these to construct a module lattice. If the shortest vector in this module lattice were close to the bound in Hermite's theorem, then this would indicate that the RLWE samples were uniformly random, but if the lattice contained an unexpectedly short vector they were not.

Langlois and Stehlé also give converse reductions (for power-of-two n, though they say that they expect the result to hold for cyclotomic rings in general) showing that not only can GIVP over module lattices be solved using a solver for MLWE, but the converse also holds. Such a result is not known to hold for RLWE and GIVP over ideal lattices, so GIVP over ideal lattices may in fact be easier than RLWE.

Albrecht and Deo give a reduction in [2] from MLWE to RLWE for different parameters. However, for decision versions of MLWE and RLWE, the reduction does not preserve non-negligible advantage unless q is superpolynomial (i.e., grows faster than a polynomial of n as n grows). For search versions of the problem the reduction works better, and for power of two cyclotomic rings they show that there is an efficient reduction from (search) MLWE with modulus q, rank d and error rate α to (search) RLWE with modulus q^d and error rate $\alpha \cdot n^2\sqrt{d}$.

3.5 Variants of LWE

There are many variations of the LWE problem, all of which consist (in the decision version of the problem) of distinguishing uniformly random samples from samples that are some kind of "noisy products". As the decision problems are more commonly used in cryptography than the search problems, only the former are given below.

When LWE was suggested in 2005 there was already a similar problem called the *learning parity with noise problem*, which is equivalent to the problem of decoding linear codes.

Denote as Ber_τ the Bernoulli distribution on $\mathbb{Z}_2$ with parameter τ (where $0 < \tau < 1/2$).

Definition 3.8. (Decision-LPN) Given m samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_2^n \times \mathbb{Z}_2$, where for $1 \leq i \leq m$, $\mathbf{a}_i \in \mathbb{Z}_2^n$ is uniformly random, determine whether the $b_i \in \mathbb{Z}_2$ are also uniformly random, or whether there is $\mathbf{s} \in \mathbb{Z}_2^n$ such that for every i, $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i$ (where $e_i \leftarrow \text{Ber}_\tau$).

Thus LPN is essentially LWE in the special case where $q = 2$. Operations using LPN are very efficient, since multiplication is simply the logical operation AND and addition is XOR. The quantum reduction from GapSVP_γ and SIVP_γ does not hold for LPN, since the reduction requires $q \geq 2\sqrt{n}/\alpha$ (where $0 < \alpha < 1$), so in particular $q > 2$. Despite this the problem still seems to be hard, at least for a classical attacker. The best known classical algorithms require $2^{\Theta(n/\log n)}$ time and samples (given only a linear number of samples, exponential time is

required), but in a recent article [33] from April 2017 Ristè et al. show that a quantum attacker would require only a logarithmic number of samples and linear time (as functions of the problem size n) to solve LPN. Therefore it does not seem a useful problem for cryptosystems that need to provide post quantum security.

3.5.1 LWR

A more recent variant of LWE, introduced in 2012 by Banerjee, Peikert and Rosen in [11] and studied further by Alwen, Krenn, Pietrzak and Wichs in [4], is the *learning with rounding problem*. Here the noisy products are formed using a rounding function rather than by adding an error to the product, meaning that the noise is deterministic.

For integers $p < q$, define a function $\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$ by $x \mapsto \lfloor (p/q) \cdot x \rceil$.

Definition 3.9. (Decision-LWR) Given m samples $(\mathbf{a}_i, \lfloor b_i \rceil_p)$, where for $1 \leq i \leq m$, $\mathbf{a}_i \in \mathbb{Z}_q^n$ is uniformly random, determine whether the $b_i \in \mathbb{Z}_q$ are also uniformly random, or whether there is $\mathbf{s} \in \mathbb{Z}_q^n$ such that for every i, $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle$.

The hardness of LWR rests on that of LWE: if LWE with sufficiently small error size is hard, then LWR for sufficiently large q/p is hard, because then $\lfloor \langle \mathbf{a}, \mathbf{s} \rangle \rceil_p \approx \lfloor \langle \mathbf{a}, \mathbf{s} \rangle + e \rceil_p$. More specifically, as Alwen et al. showed in [4][10], LWR with parameters n, m, q, p is hard if LWE with parameters n', m, q, β is hard, and

$$n \geq \frac{\log q}{\log 2\gamma}(1 + n' + \lambda) \quad \text{and} \quad q \geq 2\gamma(nm\beta p),$$

for some $\gamma \geq 1$, where q is prime and β is the error size, and λ is a security parameter of which all other parameters are functions. If the attack on LWE succeeds with advantage ϵ, that on LWR succeeds with advantage $m(2n\epsilon + 3 \cdot 2^{-\lambda})$.

The above still holds if q is not prime, as long as the largest prime factor of q divides q only once, and is greater than or equal to $2\gamma(nm\beta p)$.

Though the flexibility of γ allows for some choice in the parameters, at least one of the modulus and the dimension of the LWR instantiation must be very large for this result to hold, especially if q is not prime.

Bogdanov et al. also studied LWR in [9], where they showed that for integers p, q, n, m and B such that $q > 2pB$,

$$\Pr_{\mathbf{A},\mathbf{s},\mathbf{e}}[\mathcal{A}(\mathbf{A}, \lfloor \mathbf{As} + \mathbf{e} \rceil_p) = \mathbf{s}] \geq \frac{\Pr_{\mathbf{A},\mathbf{s}}[\mathcal{A}(\mathbf{A}, \lfloor \mathbf{As} \rceil_p) = \mathbf{s}]^2}{(1 + 2pB/q)^m},$$

where $\mathcal{A}$ is any algorithm, $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is uniformly random, $\mathbf{s} \in (\mathbb{Z}_q^n)^\times$ is chosen from any distribution and the noise $\mathbf{e}$ is chosen independently over the m coordinates from a distribution such that $e_i \in \{-B, ..., B\}$, $\Pr[e_i \geq 0] \geq 1/2$ and $\Pr[e_i \leq 0] \geq 1/2$ for each coordinate e_i of $\mathbf{e}$.

[10] Alwen et al. used a slightly different rounding function, $\lfloor \cdot \rfloor_p : \mathbb{Z}_q \to \mathbb{Z}_p$ by $x \mapsto \lfloor (p/q) \cdot x \rfloor$.

This gives a lower bound for the probability that an algorithm can find the secret vector $\mathbf{s}$ in an instance of the LWR problem with a given $\mathbf{A}$, in terms of the probability that the same algorithm can find $\mathbf{s}$ if given rounded samples from an instance of the LWE problem with the same parameters.

This is still not a very tight result, and since m is typically at least as large as n, which is typically in the hundreds, and p cannot be very much smaller than q while keeping the risk of decryption errors reasonably low, the denominator of the right hand side will be impractically large.[11]

Alperin-Sheriff and Apon give two reductions in [1]. For security parameter λ, $n \in \mathbb{N}$ and $B > 0$, let ψ be a distribution over $\mathbb{Z}$, and let $p, q, m =$poly(λ) such that $q \geq 4e \cdot Bmp\lambda$. Then Alperin-Sheriff and Apon show that if there exists a probabilistic polynomial time algorithm $\mathcal{A}$ succeeding with advantage $\epsilon \geq \lambda^{-c}$ (for some constant $c \geq 1$) in solving (decisional) LWR with parameters n, m, q, p, then there exists a probabilistic polynomial time algorithm $\mathcal{A}'$

- solving (decisional) LWE with parameters n, m, q, ψ with advantage $\epsilon' \geq (\lambda mB)^{-c}/4$, if every prime factor of q is greater than B and ψ is B-bounded;
- solving (decisional) LWE with parameters $n - c, m, q, \psi$ with advantage $\epsilon' \geq (\lambda m)^{-c}/4$, if ψ is D_α and $D_{(a^2+\omega(\log\lambda))}$ is B-bounded.

This still means a considerable increase of q. Indeed, all these reductions allow q to be polynomial in the security parameter only by bounding the number of samples m (except the first reduction given by Banerjee, Peikert and Rosen in [11] along with the definition of LWR, which allows for unbounded samples but requires superpolynomial q). When constructing a KEM, however, this does not really present a problem, as m is typically not much larger than the dimension n.

Montgomery suggests a way [26] to get unbounded samples while keeping q polynomial, but this builds on a different version of the problem, called NLWLR (Nearby Learning with Lattice Rounding), which consists of distinguishing "NLWLR-samples" from uniform. NLWLR-samples are obtained by sampling vectors $\mathbf{s}, \mathbf{u}_i, \delta_i$ and a matrix $\mathbf{B}$, from various distributions, and solving $\mathbf{A}_i\mathbf{s} = \mathbf{B}\mathbf{u}_i + \delta_i$ for $\mathbf{A}_i$ using Gaussian elimination. The NLWLR-samples are $(\mathbf{A}_i, \mathbf{z}_i = \mathbf{u}_i \bmod 2)$. The idea is that we are rounding to a lattice rather than to simply the nearest multiple of some given p, but, as Montgomery points out, this approach has drawbacks: the samples $\mathbf{A}_i$ are not uniform, which is impractical, and rounding to a lattice is not particularly efficient.

The main advantage of using LWR rather than LWE is the efficiency gain of not having to sample errors. Depending on whether there is something else that is randomly sampled, this may mean that a PKE using LWR will have a

[11] Kyber [41], which like the sample $(\mathbf{A}, \lfloor \mathbf{As}+\mathbf{e} \rceil_p)$ on the left hand side above uses both error sampling and rounding, though on module lattices, has $p = 2^{11} \approx q/4$, and a probability of decapsulation failure of $\leq 2^{-142}$. This gives us some idea, so for a conservative lower bound on the denominator we let $2pB \approx q/4$ and $m \approx 500$, which gives $(1 + 2pB/q)^m \approx 3 \cdot 10^{48} \approx 2^{161}$.

deterministic encryption algorithm, which can be an advantage or a disadvantage depending on what one wants to use it for. In particular, a PKE with an entirely determinstic encryption algorithm is not IND-CPA secure because encrypting a message several times (with the same public key) will always yield the same ciphertext. On the other hand, this property is useful when using a PKE to construct a KEM, since transformations that do this tend to require re-encryption in the encapsulation algorithm of the KEM to ensure the received ciphertext was honestly generated.

3.5.2 MLWR

As LWE can be restricted to ideal or module lattices, so can the deterministic variant LWR. The *module learning with rounding problem* is very similar to the module learning with errors problem, except the noise is formed using the rounding function $\lfloor \cdot \rceil_p$. Applying this function to a polynomial simply means applying it to each coefficient in the polynomial. As before, let $R = \mathbb{Z}[X]/(f(X))$ for some $f(X) \in \mathbb{Z}[X]$ of degree n, $R_q = R/qR$, and d a positive integer.

Definition 3.10. (Decision-MLWR) Given m samples $(\mathbf{a}_i, \lfloor b_i \rceil_p) \in R_q^d \times R_q$, where the $\mathbf{a}_i \in R_q^d$ are uniformly random, determine whether the $b_i \in R_q$ are also uniformly random, or whether $b_i = \mathbf{a}_i^T \mathbf{s} \bmod q$ for all i, where $\mathbf{s} \in R_q^d$ is uniformly random.

Bogdanov et al., who have looked at the hardness of the LWR problem, also give in the same article [9] a similar result for the ring version of the problem, i.e., for MLWR with $d = 1$. They show that for integers p, q, n, m and B such that $q > 2pB$, any function g over R_q and any algorithm $\mathcal{A}$,

$$\Pr_{\mathbf{a},s,\mathbf{e}}[\mathcal{A}(\mathbf{a}, \lfloor \mathbf{a}s + \mathbf{e} \rceil_p) = g(s)] \geq \frac{\Pr_{\mathbf{a},s}[\mathcal{A}(\mathbf{a}, \lfloor \mathbf{a}s \rceil_p) = g(s)]^2}{(1 + 2pB/q)^{nm}},$$

where $\mathbf{a} \in R_q^m$ is uniformly random, s is chosen from any distribution over the units of R_q and the noise $\mathbf{e}$ is chosen independently over the m coordinates from a distribution such that $e \in \{-B, ..., B\}$, $\Pr[e \geq 0] \geq 1/2$ and $\Pr[e \leq 0] \geq 1/2$ for each coefficient e in each coordinate of $\mathbf{e}$.

This inequality gives a lower bound for the probability that an algorithm can find, for any g over R_q, some information $g(s)$ about the secret s belonging to an instance of the MLWR problem with a given $\mathbf{A}$, in terms of the probability that the same algorithm can find the same information $g(s)$ about the same secret s if given rounded samples from an instance of the MLWE problem with the same parameters as the MLWR instance.

Unfortunately, like the version for general lattices, this bound is too loose to say much about the theoretical security of RLWR. Even though instantiations of RLWE may use $m = 1$ (see e.g. [40]), the denomitator of the right hand side becomes very large.[12]

[12] With the parameters of NIST-submission NewHope which uses RLWE ($q = 12289, B =$

Alperin-Sheriff and Apon's reduction [1] for LWR also holds in the module setting, and that seems rather tighter with regards to the advantage, though it requires a larger modulus than the reduction of Bogdanov et al.

3.5.3 Other variants of MLWE

I-MLWE. In [13] Chunsheng suggested a variant of RLWE over the ring of integers modulo some large number rather than a polynomial ring. This ring was formed by taking some instance of RLWE with ring $R' = \mathbb{Z}[X]/(f(X))$ (for some $f(X) \in \mathbb{Z}[X]$ of degree n) and modulus q, and then replacing X with q to get a new ring $R = \mathbb{Z}/N\mathbb{Z}$ where N is the polynomial f evaluated at $X = q$. The error distribution χ remains a discrete Gaussian distribution.

One of the submissions to the NIST call for proposals called ThreeBears [44] uses the module version of this problem, and calls it the *integer module learning with errors problem*. It is not defined using an instance of RLWE, but the principle is the same.

Let $R = \mathbb{Z}/N\mathbb{Z}$ for some $N \in \mathbb{Z}$, let d, m be integers, and let χ be an error distribution over R.

Definition 3.11. (Decision-I-MLWE) Given m samples $(\mathbf{a}_i, b_i) \in R^d \times R$, where for $1 \leq i \leq m$, $\mathbf{a}_i \in R^d$ is uniformly random, determine whether the $b_i \in R$ are also uniformly random, or whether there is $\mathbf{s} \in R^d$ such that for every i, $b_i = \mathbf{a}_i^T \mathbf{s} + e_i$ (where $e_i \leftarrow \chi$).

The modulus N is very large (in ThreeBears $N = 2^{3120} - 2^{1560} - 1$) so the system deals with very large numbers, but operations are still quicker than the polynomial multiplications of RLWE and MLWE. As for security, I-MLWE is a very new problem and not much studied.

MP-LWE. In [32] Roşca, Sakzad, Stehlé, and Steinfeld suggested a variant of RLWE called *middle-point LWE*. This uses so called middle-point multiplication to multiply two polynomials of some maximum degree, and keeping only the middle n terms.

More formally, let m, n, q be integers and let $\mathbb{Z}_q^{<n}[X]$ denote the set of polynomials in $\mathbb{Z}_q[X]$ of degree less than n. For $a \in \mathbb{Z}_q^{<n}[X]$ and $s \in \mathbb{Z}_q^{<2n-1}[X]$ define

$$a \odot_n s = \lfloor (a \cdot s \bmod X^{2n-1})/X^{n-1} \rfloor \in \mathbb{Z}_q^{<n}[X].$$

Thus, $a \odot_n s$ means that we multiply a and s, reduce the maximum degree of the terms in the product by modding by X^{2n-1}, then divide by X^{n-1} and round to zero all resulting terms that have X to a negative power.

Definition 3.12. (Decision-MP-LWE) Given m samples $(a_i, b_i) \in \mathbb{Z}_q^{<n}[X] \times \mathbb{Z}_q^{<n}[X]$, where for $1 \leq i \leq m$, $a_i \in \mathbb{Z}_q^{<n}[X]$ is uniformly random, determine whether the $b_i \in \mathbb{Z}_q^{<}[X]$ are also uniformly random, or whether there is $s \in$

$8, m = 1$, and $n = 512$ or $n = 1024$), and choosing $p = 256$ (which is probably far too small for a low error rate) the factor $(1 + 2pB/q)^{nm}$ is about $10^{64} \approx 2^{212}$ for $n = 512$, and about $10^{128} \approx 2^{425}$ for $n = 1024$.

$\mathbb{Z}_q^{<2n-1}[X]$ such that for every i, $b_i = a_i \odot_n s + e_i$ (where $e_i \leftarrow \chi$ for χ some error distribution over $\mathbb{Z}_q^{<n}[X]$).

This is a bit more technical than most other variations of LWE, and the implementation is not very efficient. One of the submissions the the NIST call for proposals, Titanium [45], uses MP-LWE, and is about on level with the submission using plain LWE (FrodoKEM [39]) when it comes to both runtime and keysizes.

The point of using MP-LWE is that it comes with a security reduction which shows that as long as RLWE over $R = \mathbb{Z}[X]/(f(X))$ is hard for some $f \in \mathcal{F}$, MP-LWE is hard, where $\mathcal{F}$ is a polynomial family of exponential size in the security parameter (it contains polynomials of the form $X^m + \sum_{i \leq k(m)} f_i X^i$ for some $k(m)$). Therefore, MP-LWE is a good basis for a cryptosystem if we think that polynomials f such that RLWE is hard over $R = \mathbb{Z}[X]/(f(X))$ exist and are common enough that $\mathcal{F}$ will usually contain such an f, but that we cannot with reasonable certainty identify for which f RLWE is hard (since if we could do so, it would be more efficient simply to use RLWE for such an f).

However, unless there is room for improvement when it comes to efficiency, it may be better to simply use plain LWE in situations when we are not ready to trust RLWE or MLWE.

4 Examples of cryptosystems

Many of the schemes submitted to NIST on their recent call for proposals are lattice based, which gives an idea of how schemes will work that are based on the various lattice problems mentioned so far. This section gives some idea of three of these, but the reader is warned that less of the notation and terminology is explained from now on.

The three cryptosystems discussed in this section are FrodoKEM, which is based on plain LWE, NewHope, which uses RLWE, and Kyber, which uses MLWE. In each case the (main) product is an IND-CCA secure KEM, which is obtained by applying a transform to an IND-CPA secure PKE. There are a number of options for transforms (see Sections 2.4.5-2.4.7) but some care must be taken in the case of lattice schemes since many transforms assume the underlying PKE is perfectly correct (i.e. that the decryption algorithm always recovers the correct message if the ciphertext is valid). Lattice schemes do not tend to be perfectly correct, though they can be made so at the cost of either security or efficiency.

Each scheme comes with a few suggested sets of parameters, giving different security levels. The NIST call for proposals [38] defines five levels or categories of security. Quoted from the call for proposals, these are:

1. "Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 128-bit key (e.g. AES128)

2. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for collision search on a 256-bit hash function (e.g. SHA256/SHA3-256)

3. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 192-bit key (e.g. AES192)

4. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for collision search on a 384-bit hash function (e.g. SHA384/SHA3-384)

5. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 256-bit key (e.g. AES 256)"

Different metrics may be used to measure the computational resources, and in order for a scheme to fulfill any of the above requirements any attack must require comparable or greater resources than the above bounds, with respect to all metrics that may be relevant. This includes quantum metrics, and NIST recommends restricting quantum attacks to some fixed maximum circuit depth, MAXDEPTH, which may range from 2^{40} to 2^{96}.

4.1 FrodoKEM

FrodoKEM, being based on plain LWE, is the simplest of the schemes to explain. It offers IND-CCA security at two levels: level 1 and level 3, comparable to key search on block ciphers with a 128-bit and 192-bit key, respectively, with respect to relevant classical and quantum metrics. It is significantly less efficient than more structured variants but according to the paper submitted to NIST [39] it is still practical for the vast majority of devices, networks and applications, and since there are few requirements on the parameters it is possible to meet security targets rather tightly.

Commenting on FrodoKEM, Bernstein argues that one of the security claims (regarding quantum security) is incorrect because the relevant theorem does not hold unless the parameters are increased so much as to render the scheme impractical, and that another is formulated so vaguely that it says nothing about the scheme. However, in a reply to this comment Peikert clarifies that in the FrodoKEM submission, the theoretical security proofs refer to the asymptotic security of a parametrisable scheme, and that the parameter choices for the specific instantiations are supported by cryptanalysis (see also Section 1.2.2 of [39]).

4.1.1 The algorithms

FrodoKEM is obtained by applying a transform to the IND-CPA secure PKE FrodoPKE, which consists of the algorithms FrodoPKE.KeyGen, FrodoPKE.Enc

and FrodoPKE.Dec, Algorithms 3, 4 and 5. These contain some integer parameters, $n, \bar{n}, \bar{m}$, $\text{len}_{\mathbf{A}}$ and $\text{len}_{\mathbf{E}}$, and there are a number of separate algorithms that convert bit strings to matrices and vice versa, or sample matrices from bit string seeds. These processes are described in words in the pseudo code below, glossing over the details, but it is perhaps worth noting that the matrix $\mathbf{A}$, which has entries in $\mathbb{Z}_q$ is generated using a different algorithm than the error matrices, which have integer entries. In the sampling of error matrices, different indices are used as domain separators so that different matrices can be generated from the same seed, like $\mathbf{S}$ and $\mathbf{E}$ in FrodoPKE.KeyGen.

Algorithm 3 FrodoPKE.KeyGen(): key generation

Output: Public key pk
Output: Secret key sk
1: Choose a uniformly random seed $\text{seed}_{\mathbf{A}} \in \{0,1\}^{\text{len}_{\mathbf{A}}}$
2: Generate pseudorandom matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ deterministically from $\text{seed}_{\mathbf{A}}$, using AES128 or cSHAKE128
3: Choose a uniformly random seed $\text{seed}_{\mathbf{E}} \in \{0,1\}^{\text{len}_{\mathbf{E}}}$
4: Generate pseudorandom matrices $\mathbf{S}, \mathbf{E} \in \mathbb{Z}^{n \times \bar{n}}$ deterministically from $\text{seed}_{\mathbf{E}}$, using cSHAKE
5: Compute $\mathbf{B} = \mathbf{AS} + \mathbf{E}$
6: **return** public key $pk = (\text{seed}_{\mathbf{A}}, \mathbf{B})$ and secret key $sk = \mathbf{S}$

Algorithm 4 FrodoPKE.Enc(pk, μ): encryption

Input: Public key pk
Input: Message μ (bit string)
Output: Ciphertext c
1: Generate $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ from $\text{seed}_{\mathbf{A}}$ as in KeyGen
2: Choose a uniformly random seed $\text{seed}_{\mathbf{E}} \in \{0,1\}^{\text{len}_{\mathbf{E}}}$
3: Generate pseudorandom matrices $\mathbf{S}', \mathbf{E}' \in \mathbb{Z}^{\bar{m} \times n}$ from $\text{seed}_{\mathbf{E}}$
4: Generate pseudorandom matrix $\mathbf{E}'' \in \mathbb{Z}^{\bar{m} \times \bar{n}}$ from $\text{seed}_{\mathbf{E}}$
5: Compute $\mathbf{B}' = \mathbf{S}'\mathbf{A} + \mathbf{E}'$ and $\mathbf{V} = \mathbf{S}'\mathbf{B} + \mathbf{E}''$
6: **return** ciphertext $c = (\mathbf{C}_1, \mathbf{C}_2)$ where $\mathbf{C}_1 = \mathbf{B}'$ and $\mathbf{C}_2$ is the sum of $\mathbf{V}$ and the message μ, encoded as a matrix

Note that $\text{seed}_{\mathbf{E}}$ in FrodoPKE.Enc is different from $\text{seed}_{\mathbf{E}}$ in FrodoPKE.KeyGen (but just in case they should happen to be the same, the algorithm sampling error matrices still has a different value of the domain separator for each of $\mathbf{E}, \mathbf{S}, \mathbf{E}', \mathbf{E}'', \mathbf{S}'$).

The algorithms for FrodoKEM are formed from FrodoPKE using the transform $\text{QFO}_m^{\not\perp}$ from [19] (see Section 2.4.6) with some adjustments. The transform as given in [19] uses random coins $G(m)$ for encryption, and has the shared key $K = H(m)$ and $d = H'(m)$, for message m and hash functions G, H and H'. The transform used in [39] differs in the following ways:

Algorithm 5 FrodoPKE.Dec(): decryption

Input: Secret key sk
Input: Ciphertext c
Output: Message μ'
1: Compute $\mathbf{M} = \mathbf{C}_2 - \mathbf{C}_1\mathbf{S}$
2: **return** message μ', $\mathbf{M}$ expressed as a bit string

- One hash function generates (r, k, d), where r are the random coins ($G(m)$ in $\text{QFO}^{\not\perp}_m$) and k is used in generating K.
- The hash function generating (r, k, d) takes the public key as input as well as the message.
- The computation of the shared secret K takes c, k and d as input, rather than just the message.
- The key generation algorithm for FrodoKEM also includes pk as part of sk, though the description of the transform does not. In practice, since the decapsulation algorithm reencrypts the message, it will of course need to have access to the public key.

The resulting transform is denoted $\text{QFO}^{\not\perp\prime}_m$.

The FrodoKEM algorithms also converts matrices that are to be sent (e.g. $\mathbf{B}$ in the public key) or used as input in hash functions (e.g. $\mathbf{C}_1$ and $\mathbf{C}_2$) to bit strings.

The suggested parameters are

- $q = 2^D$ for $D = 15$ and $D = 16$, respectively;
- integers $n, \bar{n}, \bar{m}$ such that $n \equiv 0 \pmod 8$;
- $\text{len}_{\mathbf{A}} = 128$;
- $\text{len}_{\mathbf{E}} = 128$ and $\text{len}_{\mathbf{E}} = 192$, respectively;
- $B = 2$ and $B = 3$, respectively ($B \leq D$ is the number of bits encoded in each matrix entry).

There is also an error distribution used in sampling the error matrices, which should ideally be a discrete Gaussian, but since this is inefficient to sample from the scheme uses another distribution that approximates a discrete Gaussian.

Generating $\mathbf{A}$ is costly: it is about 40% of the cost of encapsulation and decapsulation. The matrix $\mathbf{A}$ could be fixed, but this brings a risk of backdoors and all-for-the-price-of-one attacks. The authors suggest it could instead be reused a small number of times without greatly increasing risks.

4.1.2 Decapsulation error

A critical part of the decapsulation algorithm is of course recovering μ'. Let $\mathbf{M}$ denote the matrix that is μ encoded as a matrix. Then

$$\mathbf{M} = \mathbf{C} - \mathbf{V} = \mathbf{C} - (\mathbf{S}'\mathbf{B} + \mathbf{E}'').$$

Decaps has access to $\mathbf{A}, \mathbf{S}, \mathbf{B}, \mathbf{B}'$ and $\mathbf{C}$, but not to $\mathbf{S}'$, so it approximates $\mathbf{S}'\mathbf{B} + \mathbf{E}''$ as $\mathbf{B}'\mathbf{S}$. Since

$$\mathbf{S}'\mathbf{B} + \mathbf{E}'' - \mathbf{B}'\mathbf{S} = \mathbf{S}'\mathbf{E} - \mathbf{E}'\mathbf{S} + \mathbf{E}'',$$

this only works if $\mathbf{E}''' = \mathbf{S}'\mathbf{E} - \mathbf{E}'\mathbf{S} + \mathbf{E}''$ is so small that rounding $\mathbf{B}'\mathbf{S}$ will yield $\mathbf{S}'\mathbf{B} + \mathbf{E}''$. This rounding occurrs in decoding $\mathbf{C} - \mathbf{B}'\mathbf{S}$ as a bit string. The decoding algorithm applies the function $\mathrm{dc}(\cdot)$ to each entry of the matrix, interprets the resulting integers as B-bit strings and concatenates these into one larger string (B is some integer such that $B \leq D$, given as part of the parameter set), where

$$\mathrm{dc}(c) = \lfloor c \cdot 2^B/q \rceil \bmod 2^B.$$

The encoding algorithm which encodes a bit string as a matrix with entries in $\mathbb{Z}_q$ splits the bit string into B-bit substrings, interprets each substring as an integer, and applies the function $\mathrm{ec}(\cdot)$ to each integer in turn to get the entries for a matrix, where

$$\mathrm{ec}(k) = k \cdot q/2^B.$$

Thus $\mathrm{dc}(\mathrm{ec}(k)) = k$ as long as $0 \leq k < 2^B$, so the decapsulation algorithm will recover μ by decoding $\mathbf{C} - \mathbf{B}'\mathbf{S}$ as long as all entries of $\mathbf{E}'''$ are strictly larger than $-q/2^{B+1}$ and at most $q/2^{B+1}$. (In the suggested parameter sets, $q/2^{B+1} = 2^{12}$.) Fortunately, this is usually the case. The level 1 security parameter set has failure probability $2^{-148.8}$, and the level 3 security parameter set has failure probability $2^{-199.6}$.

4.1.3 Security

The paper [39] for FrodoKEM offers a number of security reductions:

- FrodoKEM is an IND-CCA-secure KEM assuming FrodoPKE is an IND-CPA PKE and the hash functions used (cSHAKE) are random oracles.[13]
- FrodoPKE is an IND-CPA PKE assuming that the corresponding normal form LWE problem is hard.
- Justification for substituting exact rounded Gaussian errors with the distribution used. Bounds for exact security loss.
- Replacing $\mathbf{A}$ sampled from a truly uniform distribution with pseudorandom $\mathbf{A}$ generated from random seed.

[13] This is claimed to be supported by [19], but the specification of Frodo uses a different definition of correctness than [19]; see Section 2.4.8.

- Normal form LWE is hard assuming uniform-secret LWE is hard for same parameters (small loss of samples).
- Average-case uniform-secret LWE is hard assuming that the worst-case BDDwDGS problem is hard for related parameters.

Recall that the BDDwDGS (BDD with discrete Gaussian samples) problem is a variant of the BDD problem on the lattice $\mathcal{L}$, where the adversary has access to an oracle that samples from the discrete Gaussian distribution $D_{\mathcal{L}^*,s}$ for any adaptively queried $s \geq r$, where $r > 0$ is given and $\mathcal{L}^*$ is the dual of $\mathcal{L}$.

The authors of FrodoKEM note that for a given distance bound, known BDDwDGS algorithms use samples that all have the same parameter s, but the reduction to LWE uses the ability to vary s.

BDDwDGS is used instead of GapSVP or SIVP since the standard deviation used in FrodoKEM is too small for the reduction to GapSVP and SIVP to hold. However, even so the theoretical security does not say much for the suggested parameters in FrodoKEM, because the reductions are too loose. They prove asymptotic security, but the security of specific instantiations is estimated using cryptanalysis, and this is how the parameter choices are motivated.

4.2 NewHope

NewHope is based on RLWE over cyclotomic rings $R = \mathbb{Z}[X]/(X^n + 1)$ for n a power of 2, and offers IND-CCA security at two levels: level 1 and level 5. It is less flexible than FrodoKEM, and the algorithms are optimized for $n = 512$ or $n = 1024$, and $q = 12289$ and $k = 8$, where q is the modulus and k a parameter for noise distribution. Instead of discrete Gaussian errors, NewHope uses the centered binomial distribution of parameter $k = 8$, which has standard deviation $\sqrt{k/2} = 2$.

For multiplying polynomials NewHope uses NTT, and it can not easily be used with any other method for multiplication since some of the messages are sent in the NTT domain to reduce the number of necessary transforms. (This means that no other definition or parametrisation of NTT can be used with NewHope, at least not without then transforming messages to fit this definition and parametrisation.) Using NTT affects the choice of parameters, since it requires q to be a prime congruent to 1 mod $2n$. The smallest such prime, for either value of n, is $q = 12289$. (According to [40], the security level grows with n and the noise-to-modulus ratio, so q should be as small as possible. A small q is also makes NTT more efficient.)

4.2.1 The algorithms

Like FrodoKEM, NewHope starts from an IND-CPA secure PKE, shown in Algorithms 6, 7 and 8, and applies the exact same transform $\text{QFO}^{\not\perp}_m{}'$ as FrodoKEM to obtain an IND-CCA secure KEM.

Algorithm 6 NewHope-CPA-PKE.Gen(): key generation

Output: Public key pk
Output: Secret key sk

1: Choose a uniformly random seed$\in \{0, ..., 255\}^{32}$
2: Compute $z_1, z_2 \in \{0, ..., 255\}^{32}$ as (z_1, z_2) =SHAKE256(64,seed)
3: Generate pseudorandom polynomial $\hat{\mathbf{a}} \in R_q$ deterministically from z_1, using SHAKE128
4: Generate pseudorandom polynomials $\mathbf{e}, \mathbf{s} \in R_q$ deterministically from z_2, and bit-reverse them.
5: Compute $\hat{\mathbf{e}}$ =NTT($\mathbf{e}$) and $\hat{\mathbf{s}}$ =NTT($\mathbf{s}$)
6: Compute $\hat{\mathbf{b}} = \hat{\mathbf{a}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
7: **return** (pk, sk) where pk is $\hat{\mathbf{b}}$ encoded as a byte string, concatenated with z_1, and sk is $\mathbf{s}$ encoded as a byte string

As in FrodoKEM, different domain separators ensure that $\mathbf{e} \neq \mathbf{s}$ though they are generated in the same way from the same seed. Generation of $\hat{\mathbf{a}}$ is done using a different algorithm.

Bit-reversal, for a sequence of 2^l integers, is a permutation of the sequence, and is done by writing the integers in binary representation, padded to have length l, and reversing the order. For a polynomial $\mathbf{s}$, bit-reversal is defined as the polynomial where the exponents have been bit-reversed.

Inputs to NTT would normally need to be bit-reversed if implementations use in-place NTT algorithms, and this is therefore included in the algorithms for key generation and encryption. However, since the polynomials that go into NTT are pseudorandom they can be considered already bit-reversed, and as an optimization NewHope "allow[s] implementations to skip these bit-reversals for forward transformation". The operation $\circ$ is coordinate-wise multiplication modulo q. Because of the NTT-transform, $\mathsf{NTT}^{-1}(\hat{\mathbf{a}} \circ \hat{\mathbf{s}}) = \mathbf{a} \cdot \mathbf{s}$. Since the $\mathbf{a}$ is not needed, $\hat{\mathbf{a}}$ is generated in the NTT domain.

(There are different ways, not repeated here, to turn polynomials into byte strings and vice versa. Most of these are more or less the same, sometimes with built-in concatenation with some other byte string, but some differ in the length of the resultant byte string. See the NewHope specification [40] for details.)

Like most lattice based cryptosystems, NewHope is not perfectly correct, but the probability of decryption error is 2^{-213} for the smaller parameter set, and 2^{-216} for the larger.

4.2.2 Security

Like FrodoKEM, NewHope comes with a number of security reductions, more details on which can be found in the NewHope paper [40]. These reductions are:

- Using the centered binomial distribution instead of a discrete Gaussian one gives negligible advantage to the adversary.

Algorithm 7 NewHope-CPA-PKE.Encrypt($pk, \mu, coin$): encryption

Input: Public key pk
Input: Message μ
Input: Random coins $coin$
Output: Ciphertext c

1: Recover $\hat{\mathbf{b}}$ and z_1 from pk
2: Generate $\hat{\mathbf{a}} \in R_q$ from z_1, as in Gen
3: Generate pseudorandom polynomials $\mathbf{e}', \mathbf{s}' \in R_q$ deterministically from $coin$, and bit-reverse them
4: Generate pseudorandom polynomial $\mathbf{e}'' \in R_q$ deterministically from $coin$
5: Compute $\hat{\mathbf{t}} = \mathsf{NTT}(\mathbf{s}')$
6: Compute $\hat{\mathbf{u}} = \hat{\mathbf{a}} \circ \hat{\mathbf{t}} + \mathsf{NTT}(\mathbf{e}')$
7: Encode message μ as a polynomial $\mathbf{v} \in R_q$
8: Compute $\mathbf{v}' = \mathsf{NTT}^{-1}(\hat{\mathbf{b}} \circ \hat{\mathbf{t}}) + \mathbf{e}'' + \mathbf{v}$
9: Compress $\mathbf{v}'$ into a byte string h
10: Encode $\hat{\mathbf{u}}$ as a byte string, concatenate with h, and denote this c
11: **return** ciphertext c

Algorithm 8 NewHope-CPA-PKE.Decrypt(sk, c): decryption

Input: Secret key sk
Input: Ciphertext c
Output: Message μ'

1: Recover $(\hat{\mathbf{u}}, h)$ from c
2: Recover polynomial $\hat{\mathbf{s}}$ from byte string sk
3: Recover polynomial $\mathbf{v}'$ from byte string h
4: Recover message μ' by decoding $\mathbf{v}' - \mathsf{NTT}^{-1}(\hat{\mathbf{u}} \circ \hat{\mathbf{s}})$
5: **return** message μ'

- Using a pseudorandomly generated $\hat{\mathbf{a}}$ instead of a uniformly random $\hat{\mathbf{a}}$ gives no advantage to the adversary, assuming SHAKE128 is a random oracle.
- NewHope-CCA-KEM is an IND-CCA-secure KEM assuming NewHope-CPA-PKE is an IND-CPA PKE and the hash functions used (SHAKE256) are random oracles.[14]
- NewHope-CPA-PKE is an IND-CPA PKE assuming that the corresponding decision RLWE problem is hard.
- Decision RLWE is hard assuming approximate SVP is hard (in the worst case) on ideal lattices in R, for appropriate parameters.

NewHope rests on the assumption that SVP is still hard even on more structured lattices, which, at least so far, it seems to be. The best known (quantum) polynomial-time algorithm for approximate-SVP gives subexponential approximation factors $2^{\tilde{O}(\sqrt{n})}$ which is better than has so far been achieved in more general lattices, but this is still larger than the approximation factors used in cryptography.

4.3 Kyber

Kyber, or properly CRYSTALS-Kyber [41], is based on MLWE, but the noisy products are also rounded as in MLWR. This is mainly done to decrease the message sizes, and though it adds noise and therefore increases security it is not considered in the security analysis. Like NewHope and many others, it uses a cyclotomic ring $R = \mathbb{Z}[X]/(X^n + 1)$ for n a power of 2, but instead of using simply polynomials of R_q it uses short vectors of polynomials, of length k. This way, the dimension of the underlying lattice is not n, but nk, allowing for better scaling. Kyber uses $n = 256$, and k between 3 and 5, giving lattice dimensions 512, 768 and 1024, respectively, for instantiations of the scheme with IND-CCA security at level 1, 3 and 5.

Like NewHope, Kyber uses NTT for multiplying polynomials, and therefore n should be a power of 2, and q the smallest prime congruent to 1 modulo $2n$. The choice of n as 256 specifically has to do with the length of the keys to be encrypted. These are 256 bits, and choosing a smaller n would mean encoding more than one bit into each polynomial coefficient, which would require lower noise levels. A larger n would make the ability to scale less interesting.

This smaller n than in NewHope has the added advantage of allowing for a smaller q, which makes NTT more efficient: Kyber uses $q = 7681$ as opposed to NewHope's $q = 12289$.

[14] This is claimed to be supported by [19]. The specification [40] for NewHope does not define correctness, but if it uses the same definition as Frodo, among others, then this is not the same as the definition of correctness used in [19]; see Section 2.4.8.

Kyber also has parameters η, for noise magnitude, and d_u, d_v, d_t, for rounding. Noise is sampled from a centered binomial distribution B_η, which samples

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

and outputs

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

All three instantiations Kyber512, Kyber768 and Kyber1024 share the parameters $(n, q, d_u, d_v, d_t) = (256, 7681, 11, 3, 11)$, while k and η vary, with $(k, \eta) \in \{(2,5), (3,4), (4,3)\}$. The middle instantiation, Kyber768, with $(k, \eta) = (3, 4)$, is recommended.

4.3.1 Algorithms

Let $R = \mathbb{Z}[X]/(X^n + 1)$ and $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ and let $\mathcal{B}$ denote the set $\{0, ..., 255\}$. Kyber uses a number of functions in its algorithms.

- $\mathsf{BytesToBits}$ takes a byte array $(b_1, ..., b_l)$ of length l and outputs a bit array $(\beta_1, ..., \beta_{8l})$ of length $8l$, by computing $\beta_i = ((b_{\lceil i/8 \rceil}/2^{(i \mod 8)}) \mod 2)$
- NTT is as in Section 2.2.6, with $\omega = 3844$ and $\gamma = 62$.
- br_{256} reverses the bits of an 8-bit integer. That is, write an integer $i \in \{0, ..., 255\}$ in base 2, padding it with zeros if necessary so that it has length 8 exactly. Reverse the order of the bits. This is the base 2 representation of $\mathsf{br}_{256}(i)$.
- For $x \in \mathbb{Z}_q$ and $d < \lceil \log_2(q) \rceil$, $\mathsf{Compress}_q(x, d) = \lceil 2^d/q \cdot x \rfloor \mod^+ 2^d \in \{0, ..., 2^d - 1\}$.
- $\mathsf{Decompress}_q(x, d) = \lceil q/2^d \cdot x \rfloor \in \mathbb{Z}_q$.
- Symmetric primitives: Kyber uses a pseudorandom function $\mathsf{PRF} : \mathcal{B}^{32} \times \mathcal{B} \to \mathcal{B}^*$, an extendable output function $\mathsf{XOF} : \mathcal{B}^* \to \mathcal{B}^*$ and two hash functions $H : \mathcal{B}^* \to \mathcal{B}^{32}$ and $G : \mathcal{B}^* \to \mathcal{B}^{32} \times \mathcal{B}^{32}$.

If $x \in R$ or R_q, $\mathsf{Compress}_q(x, d)$ and $\mathsf{Decompress}_q(x, d)$ are applied to each coefficient individually. It is these functions which do the rounding in Kyber.

$\mathsf{Decompress}_q(x, d)$ is almost the inverse of $\mathsf{Compress}_q(x, d)$, in that, for $x \in \mathbb{Z}_q$,

$$|\mathsf{Decompress}_q(\mathsf{Compress}_q(x, d), d) - x \mod^\pm q| \leq \lceil q/2^{d+1} \rfloor.$$

Note that Kyber does not use the canonical embedding for the norm of elements in R, but instead associates an element in R with the vector of its coefficients and takes the l_2 or l_{inf} norm.

The NTT transform of a polynomial can be computed conveniently in place, if we assume polynomials in the NTT domain to be in bit-reversed order, meaning that coefficient $\hat{g}_i$ of $\hat{\mathbf{g}} = \mathsf{NTT}(\mathbf{g})$ is stored at position $\mathsf{br}_{256}(i)$.

The main algorithms of Kyber also use a few algorithms for sampling or encoding. Detailed descriptions of these can be found in [41].

- Parse takes a byte stream $b_0, b_1, ... \in \mathcal{B}^*$ and outputs a polynomial $\hat{a} \in R_q$, which is assumed to be in the NTT domain. If the input byte stream is statistically close to a uniformly random byte array, the output polynomial is statistically close to a uniformly random element of R_q. Since the NTT maps uniformly random polynomials to uniformly random polynomials, we can assume that the output is in the NTT domain.

- CBD_η samples from the centered binomial distribution B_η. It outputs an element of R_q whose coefficients are sampled from B_η using the input byte array of length 64η (assuming $n = 256$).

- Encode_l encodes a polynomial in R_q with coefficients in $\{0, ..., 2^l - 1\}$ as a byte array of length $32l$. Applied to a vector of polynomials means applying it to each polynomial individually and concatenating the output polynomials.

- Decode_l is the inverse of Encode_l, and decodes a byte array of length $32l$ as a polynomial in R_q with coefficients in $\{0, ..., 2^l - 1\}$.

Key generation, encryption and decryption for the IND-CPA secure PKE Kyber.CPAPKE are given in algorithms 9, 10 and 11, which are algorithms 4, 5 and 6 in the Kyber specification [41].

Algorithm 9 Kyber.CPAPKE.KeyGen(): key generation

Output: Public key $pk \in \mathcal{B}^{d_t \cdot k \cdot n/8+32}$
Output: Secret key $sk \in \mathcal{B}^{13 \cdot k \cdot n/8}$

$d \leftarrow \mathcal{B}^{32}$
$(\rho, \sigma) := G(d)$
$N := 0$
for i from 0 to $k-1$ **do** ▷ Generate $\hat{\mathbf{A}} \in R_q^{k \times k}$ in NTT domain
  for j from 0 to $k-1$ **do**
    $\hat{\mathbf{A}}[i][j] := \mathsf{Parse}(\mathsf{XOF}(\rho||j||i))$
for i from 0 to $k-1$ **do** ▷ Sample $\mathbf{s} \in R_q^k$ from B_η
  $\mathbf{s}[i] := \mathsf{CBD}_\eta(\mathsf{PRF}(\sigma, N))$
  $N := N + 1$
for i from 0 to $k-1$ **do** ▷ Sample $\mathbf{e} \in R_q^k$ from B_η
  $\mathbf{e}[i] := \mathsf{CBD}_\eta(\mathsf{PRF}(\sigma, N))$
  $N := N + 1$
$\hat{\mathbf{s}} := \mathsf{NTT}(\mathbf{s})$
$\mathbf{t} := \mathsf{NTT}^{-1}(\hat{\mathbf{A}} \circ \hat{\mathbf{s}}) + \mathbf{e}$
$pk := (\mathsf{Encode}_{d_t}(\mathsf{Compress}_q(\mathbf{t}, d_t))||\rho)$ ▷ $pk := \mathbf{As} + \mathbf{e}$
$sk := \mathsf{Encode}_{13}(\hat{\mathbf{s}} \mod^+ q)$ ▷ $sk := \mathbf{s}$
return (pk, sk)

Algorithm 10 Kyber.CPAPKE.Enc(pk, m, r): encryption

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

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

Algorithm 11 Kyber.CPAPKE.Dec(sk, c): decryption

Input: Secret key $sk \in \mathcal{B}^{13 \cdot k \cdot n/8}$
Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8+d_v \cdot n/8}$
Output: Message $m \in \mathcal{B}^{32}$

$\mathbf{u} := \mathsf{Decompress}_q(\mathsf{Decode}_{d_u}(c), d_u)$
$v := \mathsf{Decompress}_q(\mathsf{Decode}_{d_v}(c + d_u \cdot k \cdot n/8), d_v)$
$\hat{\mathbf{s}} := \mathsf{Decode}_{13}(sk)$
$m := \mathsf{Encode}_1(\mathsf{Compress}_q(v - \mathsf{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \mathsf{NTT}(\mathbf{u})), 1))$ ▷ $m := \mathsf{Compress}_q(v - \mathbf{s}^T\mathbf{u}, 1)$
return m

Like FrodoKEM and NewHope, Kyber uses a transformation to obtain an IND-CCA secure KEM Kyber.CCAKEM, but the transformation Kyber uses is slightly different from FrodoKEM and NewHope, which both use a variant of $\text{QFO}_m^{\not\perp}$ from [19]. In Kyber, the extra hash of the message (used in [19] to prove that the $\text{QFO}_m^{\not\perp}$-transformation of an OW-CPA secure PKE will be IND-CCA secure in QROM) is not included, and instead a variant of $\text{FO}^{\not\perp}$ is used. The authors do not seem to give any reasons for using this transform, which in [19] is not proven to give QROM security, rather than the other, but a contributing reason may be the proof of Jiang et al. in [22] that $\text{FO}^{\not\perp}$ does in fact give QROM security (see Section 2.4.7).

Key generation, encapsulation and decapsulation for Kyber.CCAKEM are given in algorithms 12, 13 and 14, which are algorithms 7, 8 and 9 in [41].

Algorithm 12 Kyber.CCAKEM.KeyGen(): key generation

Output: Public key $pk \in \mathcal{B}^{d_t \cdot k \cdot n/8+32}$
Output: Secret key $sk \in \mathcal{B}^{(13+d_t) \cdot k \cdot n/8+96}$
1: $z \leftarrow \mathcal{B}^{32}$
2: $(pk, sk') :=$ Kyber.CPAPKE.KeyGen()
3: $sk := (sk'||pk||H(pk)||z)$
4: **return** (pk, sk)

Algorithm 13 Kyber.CCAKEM.Enc(pk): encapsulation

Input: Public key $pk \in \mathcal{B}^{d_t \cdot k \cdot n/8+32}$
Output: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8+d_v \cdot n/8}$
Output: Shared key $K \in \mathcal{B}^{32}$
1: $m \leftarrow \mathcal{B}^{32}$
2: $m \leftarrow H(m)$ ▷ Do not send output of system RNG
3: $(\bar{K}, r) := G(m||H(pk))$
4: $c :=$ Kyber.CPAPKE.Enc$(pk, m; r)$
5: $K := H(\bar{K}||H(c))$
6: **return** (c, K)

4.3.2 Security

In the original Kyber specification of November 2017, the security of the scheme is expressed in terms of $\mathbf{Adv}^{mlwe}_{m,k,\eta}(A)$, which is defined as

$$\mathbf{Adv}^{mlwe}_{m,k,\eta}(A) = \Big| \Pr[b' = 1 | \mathbf{A} \leftarrow R_q^{m \times k}; (\mathbf{s}, \mathbf{e}) \leftarrow B_\eta^k \times B_\eta^m; \mathbf{b} = \mathbf{As} + \mathbf{e}; b' \leftarrow A(\mathbf{A}, \mathbf{b})] - \Pr[b' = 1 | \mathbf{A} \leftarrow R_q^{m \times k}; \mathbf{b} \leftarrow R_q^{m \times k}; b' \leftarrow A(\mathbf{A}, \mathbf{b})] \Big|,$$

for any algorithm A, and they note that, since the public key and the ciphertext are pseudorandom, for any adversary A there exist adversaries B and C such

Algorithm 14 Kyber.CCAKEM.Dec(c, sk): decapsulation

Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$
Input: Secret key $sk \in \mathcal{B}^{(13+d_t) \cdot k \cdot n/8 + 96}$
Output: Shared key $K \in \mathcal{B}^{32}$

1: $pk := sk + 13 \cdot k \cdot n/8$
2: $h := sk + (13 + d_t) \cdot k \cdot n/8 + 32 \in \mathcal{B}^{32}$
3: $z := sk + (13 + d_t) \cdot k \cdot n/8 + 64$
4: $m' :=$ Kyber.CPAPKE.Dec($\mathbf{s}, (\mathbf{u}, v)$)
5: $(\bar{K}', r') := G(m'||h)$
6: $c' :=$ Kyber.CPAPKE.Enc($pk, m'; r'$)
7: **if** $c = c'$ **then**
8: $\quad K := H(\bar{K}'||H(c))$
9: **else**
10: $\quad K := H(z||H(c))$
11: **return** K

that

$$\mathbf{Adv}^{\text{IND-CPA}}_{\text{Kyber.CPAPKE}}(A) \leq 2\mathbf{Adv}^{mlwe}_{k+1,k,\eta}(B) + \mathbf{Adv}^{prf}_{\mathsf{PRF}}(C)$$

(where $\mathbf{Adv}^{prf}_{\mathsf{PRF}}(C)$ is not defined, but presumably signifies the success probability of the algorithm C attacking the security of the pseudorandom function PRF, for some appropriate security notion).

However, in a later paper [42] this claim has been modified since the compressing of $\mathbf{t}$ in line 15 of Algorithm 9 means that pk and c are no longer indistinguishable from uniform. Instead, the authors show, for the adjusted scheme Kyber.CPAPKE' where $\mathbf{t}$ is not compressed, that for any algorithm A there exists an adversary B such that

$$\mathbf{Adv}^{\text{IND-CPA}}_{\text{Kyber.CPAPKE'}}(A) \leq 2\mathbf{Adv}^{mlwe}_{k+1,k,\eta}(B).$$

The authors suggest that one way to make sure pk and c are indistinguishable from uniform in the actual scheme would be to add some small error after compression, but this would be cumbersome and would add to the decryption error. Instead, they choose to keep the compression of $\mathbf{t}$ without adding an extra error, arguing that this choice should not affect security because the value involving the decompressed $\mathbf{t}$ to which the message is added (in line 16 of algorithm 10) is then compressed much more than $\mathbf{t}$ was originally, and also because the proposed CCA-transformation does not actually require the output of the underlying PKE to be pseudorandom.

In the original specification [41], security bounds in ROM and QROM are then given, implying reductions from MLWE. However, because the reduction from MLWE in [42] was for the adjusted Kyber.CPAPKE' rather than Kyber.CPAPKE, the bounds in [42] do not use $\mathbf{Adv}^{mlwe}_{k+1,k,\eta}(A)$, but rather $\mathbf{Adv}^{\text{IND-CPA}}_{\text{Kyber.CPAPKE}}(A)$ and $\mathbf{Adv}^{pr}_{\text{Kyber.CPAPKE}}(B)$.

Theorem 4.1. *(Theorem 3, [42]) For any classical adversary A that makes at most q_{RO} queries to random oracles H and G, and q_D queries to the decryption oracle, there exists an adversary B such that*

$$\mathbf{Adv}^{IND\text{-}CCA}_{Kyber.CCAKEM}(A) \leq 3\mathbf{Adv}^{IND\text{-}CPA}_{Kyber.CPAPKE}(B) + q_{RO} \cdot \delta + \frac{3q_{RO}}{2^{256}}.$$

(Here, and in the following theorem, δ is the probability of decryption failure for Kyber.CPAPKE[15], and 2^{256} is the size of the message space.) This bound is precisely that obtained in [19] for the transform $\text{FO}^{\not\perp}$, which is essentially the transform used in Kyber.

For QROM security, the authors argue that Kyber.CPAPKE', without the compression of $\mathbf{t}$, fulfills the additional property of "sparse pseudorandomness" defined in [37], a slightly stronger security than IND-CPA, and that "pseudorandomness security" is given by

$$\mathbf{Adv}^{pr}_{\text{Kyber.CPAPKE'}}(A) \leq 2 \cdot \mathbf{Adv}^{mlwe}_{k+1,k,\eta}(B).$$

Arguing that one can assume that Kyber.CPAPKE has this same property, the authors obtain a bound for the QROM reduction. This bound is a combination of the bound for the transformation SXY in [37] and the bound (in QROM) for the transformation T in [19] (SXY requires a perfectly correct PKE as input, but otherwise the combination of T and SXY is essentially $\text{FO}^{\not\perp}$ of [19]).

Theorem 4.2. *(Theorem 4, [42]) For any quantum adversary A that makes at most q_{RO} queries to quantum random oracles H and G, and q_D (classical) queries to the decryption oracle, there exists a quantum adversary B such that*

$$\mathbf{Adv}^{IND\text{-}CCA}_{Kyber.CCAKEM}(A) \leq 8q_{RO}^2 \cdot \delta + 4q_{RO} \cdot \sqrt{\mathbf{Adv}^{pr}_{Kyber.CPAPKE}(B)}.$$

This reduction is non-tight, what with the square root and the factor $4q_{RO}$, and so only indicates asymptotic security in the QROM, but there is a suggestion that if we were to assume that the deterministic version DKyber.CPAPKE of Kyber.CPAPKE (where the random coins r are derived deterministically from the message) is is also sparse pseudorandom in the QROM, then [37] would give a tight security bound in QROM. This would presumably done by applying just the transformation SXY from [37] (T not being needed since DKyber.CPAPKE would be deterministic already), and adding some term to the bound to account for the fact that DKyber.CPAPKE would not be perfectly correct.

Extra hashes. Several of the hashes in Kyber (hashing $H(pk)$ into the pre-key $\hat{K}$ and the random coins r, and hashing $H(c)$ into the final key K) are not necessary for the security reduction, but the authors of Kyber argue that they

[15] Be aware that this probability is defined differently in Kyber than in [19], which is cited as source for this bound and as partial source for the bound in Theorem 4.2. See Section 2.4.8.

add robustness, and because the shared key K does not depend only on input from one party, as it would if it were simply hashed from m, it is safe to use in authenticated key exchange.

The reason for using $H(pk)$ and $H(c)$ here instead of just pk and c is to make Kyber more convenient to use with a non-incremental hash API[16]. The hashes $H(pk)$ and $H(c)$ are only 32 bytes long, whereas pk and c are rather longer. Moreover, using $H(pk)$ as input rather than pk speeds up the call to G in decapsulation slightly, at the cost of 32 extra bytes in the secret key.

The hash of m in line 2 of algorithm 13 does not seem to be motivated except for the comment in the algorithm that the output of the system random generator should not be sent. In the somewhat simplified algorithms of [42], this step seems to have been removed and the encapsulation simply samples $m \leftarrow \mathcal{B}^{32}$ and uses it to compute $(\bar{K}, r) := G(m||H(pk))$, skipping $m \leftarrow H(m)$.

4.3.3 Attacks

Attacks against the MLWE problem. The best known attacks against the MLWE problem do not use the structure of the module lattice, so the security analysis for Kyber considers the MLWE problem as if it were LWE. They mention recent works exploiting the structure of ideal lattices to solve SVP in such lattices, but despite the reduction of Albrecht and Deo in [2] from MLWE to RLWE (which has significant slowdown and moreover requires superpolynomial modulus to preserve non-negligible advantage for the decision-problems) these attacks do not seem to be an immediate threat to MLWE.

Some types of attacks against LWE can be ruled out because the attacker only has access to a limited number ($(k+1)n$) of LWE samples, leaving two BKZ attacks known as primal and dual attacks.

Recall that the BKZ (or BKZ-LLL) algorithm is the LLL-algorithm (algorithm 1) except that instead of the swap step it has a block reduction where a block of b vectors is reduced to a KZ-reduced basis for the sublattice they span. This reduction can be done using an SVP oracle in dimension b, but the authors of Kyber say that it is difficult to evaluate the (polynomial) number of calls to the oracle required for each block reduction, and they choose instead to count only the cost of one call to the SVP oracle in dimension b, for each block reduction. (This is called evaluating the *core SVP hardness*, and is a strategy used in security analysis by other lattice-based KEMs [39, 40].) They also choose conservative estimates for the cost of one such call.

The primal attack on an LWE problem in dimension n with m samples (and the usual notation) consists of solving unique-SVP in the lattice

$$\mathcal{L} = \{\mathbf{x} \in \mathbb{Z}^{m+kn+1} : (\mathbf{A}| - \mathbf{I}_m| - \mathbf{b})\mathbf{x} = \mathbf{0} \mod q\},$$

which has a unique shortest vector $\mathbf{v} = (\mathbf{s}, \mathbf{e}, 1)$ of norm $\lambda \approx \varsigma\sqrt{kn+m}$.[17] The number of samples may be chosen between 0 and $(k+1)n$. Optimising m and

[16] At least in the sense used here, a hash function is *incremental* if it can produce a hash $h = H(m_1||m_2)$ without first copying m_1 and m_2 into a single string $m = m_1||m_2$.

[17] ς does not seem to be defined, but may be some value specific to the LWE instance.

	b	m	Core-SVP (classical)	Core-SVP (quantum)
Kyber512				
Primal attack:	390	455	114	103
Dual attack:	385	485	112	102
Kyber768				
Primal attack:	615	695	179	163
Dual attack:	610	690	178	161
Kyber1024				
Primal attack:	845	835	244	221
Dual attack:	825	850	241	218

Table 1: Classical and quantum core-SVP hardness of the MLWE problem underlying Kyber, treated as an LWE problem. The block dimension of BKZ is denoted d, and m denotes the number of samples. Cost is given in $\log_2$ of operations.

b under the condition that this shortest vector can be found gives the attack costs shown in table 1 (table 4 of [41]).

The dual attack consists of finding a short vector in the lattice

$$\mathcal{L}' = \{(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^m \times \mathbb{Z}^{kn} : \mathbf{A}^T\mathbf{x} = \mathbf{y} \mod q\}.$$

Finding a vector of length l in this lattice gives an advantage $\epsilon = 4\exp(-2\pi^2(l\varsigma/q)^2)$ against decision-LWE. For this to be useful to an attacker, ϵ needs to be at least $1/2$, so to amplify the success probability of the attack, about $1/\epsilon^2$ such short vectors are needed. Optimising m and b gives the attack costs in table 1.

Attacks exploiting decryption failures. As we see in theorems 4.1 and 4.2 above, the probability of decapsulation failure affects the attacker's advantage, and in practice this is because decapsulation failure can reveal information about the secret. Since the random coins r are hashed from the message m and the public key pk an attacker could use different values for the message m to try and find random coins that lead to decapsulation failure (though this computation would only work for one specific public key pk, so finding such an m once will not mean that the attacker can attack any other users with this m). This would mean a brute force search, which a quantum attacker can perform more quickly using Grover's algorithm, though the speedup is limited by the fact that while a quantum attacker can send quantum queries to G and H, it can still only send classical queries to the decapsulation oracle. Therefore it cannot determine offline whether a specific choice of r will lead to a decapsulation failure, and the best bet is to use Grover's algorithm to simply search for values of r that will give $\mathbf{e}_1, \mathbf{r}$ with above average norm. Even for a quantum attacker such a search is expensive and while it would increase the probability of decapsulation error it would certainly not guarantee it. The Kyber specification suggests that if Grover's algorithm saves a square-root factor in the search for m that gives r such that $\mathbf{e}_1, \mathbf{r}$ have above average norm, then the time to find a single decryption error would still be $> 2^{128}$.

The specification also argues that a single decapsulation error would give away very little information about the secret, saying that "It seems extremely unlikely that even 10 decapsulation failures in Kyber would allow an attacker to recover any meaningful information about the secret key $\mathbf{s}$." The authors conclude that decapsulation failures do not present a threat to Kyber.

However, D'Anvers, Vercauteren and Verbauwhede have recently (November 2018) published an eprint [14] with an attack exploiting decapsulation failures (and boosting the likelihood of finding them), where they show that for schemes that, like Kyber, have very low probability of decapsulation failure, the variance of the secret decreases drastically if the attacker can find a few failing ciphertexts. By the estimate from the Kyber specification, a (primal) attack on Kyber768 would cost 2^{163} operations, whereas the cost of D'Anvers, Vercauteren and Verbauwhede's attack is 2^{141} (with 24 decapsulation failures, and 2^{130} queries to the decapsulation oracle). Similarly, the cost of an attack on Kyber1024 decreases from 2^{221} to 2^{169}, with 95 decapsulation failures and 2^{157} queries to the decapsulation oracle.

These attacks are still quite expensive, and require more decapsulation queries than the adversary is assumed to be able to make in the Call for Proposals, where the limit is 2^{64} decapsulation queries.

Side-channel attacks. The Kyber specification considers several possible side-channel attacks, that is, attacks where some kind of information is gathered from the scheme beyond simply its outputs. An attacker might collect information about runtime and power use, and from this might be able to retrieve secret information, or at least narrow down the options.

- Timing attacks collect information should not be a problem as Kyber has no secret-dependent branches or table lookups. A little care must be taken multiplication, but most non-constant-time multipliers do not show timing variation for inputs of the size occurring in Kyber. Also, the modular reductions must not be implemented via conditional statements, but they are not in Kyber.

- The authors expect that Kyber will be vulnerable to differential or electromagnetic radiation attacks unless it is implemented with dedicated protection against such attacks, but say that this is true for most scheme that uses non-ephemeral keys.

- The threat from template attacks[18] seems a little uncertain; going by previous work, the authors imply that Kyber's constant runtime gives it some protection, but that further research is required in this area, to determine to what extent template attacks are a threat to constant-time implementations of lattice-based schemes.

[18] Template attacks are two-phase attacks where the attacker builds templates or statistical models for a device when the parameters and data are known, and then in the second phase matches these templates with the information gathered from the device when not all the parameters are known.

Multi-target attacks. No formal claim about bounds for multi-target security is made, but the authors point out that the extra hashes (i.e., hashing pk into $\bar{K}$ and r) protects against an attacker trying to break one key out of many, or to find some message m that gives especially large random coins r (which would be possible if r depended only on m) and using this m against many users in the hope of producing decryption failures. Moreover, the fact that the matrix $\mathbf{A}$ is not a system parameter but is generated afresh each time protects against all-for-the-price-of-one attacks.

Attacks against symmetric primitives. Kyber uses SHAKE256, SHAKE-128, SHA3-256 and SHA3-512 to instantiate the functions modelled as random oracles. Breaking any of these would of course compromise any instantiation of Kyber using them.

4.4 Comparison between FrodoKEM, NewHope and Kyber

FrodoKEM, NewHope and Kyber are similar in some ways and differ in others. As previously mentioned, they are all IND-CCA secure KEMs obtained by applying a transform to an IND-CPA secure PKE (Frodo and NewHope use identical transforms, while that of Kyber differs slightly). All three schemes are based on LWE, though over different lattices, which makes some difference. FrodoKEM, with LWE over general lattices, enjoys the best theoretical security proof, but the best known attacks that solve SVP_γ for polynomial approximation factor γ take exponential time even for the more structured lattices used in NewHope and Kyber. FrodoKEM is more scalable than the other two, with no particular constraints on the dimension except that it be a multiple of 8. Meanwhile, NewHope and Kyber, both using NTT for polynomial multiplication, are less flexible. NewHope requires a dimension that is a power of two, while Kyber, because of the module structure, is somewhere in between with dimension that is a multiple of 256.

NTT. Both Kyber and NewHope use NTT for polynomial multiplication, which is fast. There are other contenders, but according to the Kyber specification [41] some of these (Karatsuba and Toom) require extra memory, while NTT can be computed in place. Both Kyber and NewHope have built NTT into the scheme by sometimes sending messages in the NTT domain. NewHope does this more than Kyber, sending messages and keys in the NTT domain whenever this saves a transform NTT or NTT^{-1}, meaning that if another multiplication algorithm is used in implementation, NTT must then be applied to everything that should be sent in the NTT domain. Kyber, on the other hand, compresses everything that is sent except the seed used for generating the matrix $\hat{\mathbf{A}}$, which prevents them from sending messages in the NTT domain. However, NTT is built into the scheme through the generation of $\hat{\mathbf{A}}$, and the secret key, which is not compressed, is also stored in the NTT domain.

Against all authority. All three schemes recommend generating the matrix or polynomial $\mathbf{A}/\mathbf{a}$ afresh every time, to prevent all-for-the-price-of-one attacks where an attacker (with considerable effort) finds a good basis for the lattice corresponding to $\mathbf{A}/\mathbf{a}$ and can use this to attack all users. This is most expensive for FrodoKEM, where generating $\mathbf{A}$ from a seed is about 40% of the cost of encapsulation and decapsulation. All three schemes suggest that if generating a new $\mathbf{A}/\mathbf{a}$ every time is too expensive, one could cache them for a short time.

Gaussian noise. Despite the theoretical proofs assuming that the noise is sampled from a discrete Gaussian distribution, all three schemes use other distributions for efficiency reasons. FrodoKEM samples from $\{-s, ..., -1, 0, 1, ..., s\}$ (for a positive integer s) using a discrete probability density function, while NewHope and Kyber both sample from a centered binomial distribution. In all three cases, the distribution approximates a discrete Gaussian (this is, or can be, shown using Rényi divergence). Moreover, Kyber argues that the best known attack depend on the standard deviation of the distribution, rather than the distribution itself.

Allowing decapsulation failures. Although decapsulation failures can be used to attack the scheme, all three schemes choose to allow decapsulation failures that happen with very low probability (the highest among all schemes and parameter sets is 2^{-142}). To guarantee the schemes to be perfectly correct would mean sacrificing security by increasing the modulus or decreasing the errors, or sacrificing efficiency by increasing the dimension to make up for the loss in security. Kyber argues that negligible probability of decapsulation failure is less of a threat than e.g. improvements of attacks targeting schemes with low noise.

The attack of D'Anvers, Vercauteren and Verbauwhede in [14] using failure boosting to find decapsulation errors and thus find out information about the secret, can lower the attack cost for a quantum attacker. This attack should be relevant for all three schemes, though NewHope is not included in the paper. However, it does not seem to have produced any decapsulation failures for FrodoKEM-976 which according to [39] has a error probability of about 2^{-200}. (NewHope has even lower error probability, and so may have been inconvenient to include in the study for this reason.)

Implicit rejection. All three schemes have chosen a transform that produces a KEM with implicit rejection, and according to [41] this makes implementations safe to use even if higher level protocols do not check the return value of decapsulation.

4.4.1 Sizes and speeds

Table 2 shows the sizes (in bytes) of the keys and ciphertexts of FrodoKEM, NewHope and Kyber. Note that the secret keys contain the public keys, as these are used in the reencryption part of decapsulation for all three schemes, so this contributes to the size of the secret keys.

Scheme	Security level	pk	sk	c
FrodoKEM-640	1	9616	19 872	9736
FrodoKEM-976	3	15 632	31 272	15 768
NH-512-CCA-KEM	1	928	1888	1120
NH-1024-CCA-KEM	5	1824	3680	2208
Kyber512	1	736	1632	800
Kyber768	3	1088	2400	1152
Kyber1024	5	1440	3168	1504

Table 2: Sizes (in bytes) of the public key, secret key, ciphertext and shared key of the different instantiations of FrodoKEM, NewHope and Kyber.

Scheme	Security level	KeyGen	Encaps	Decaps
FrodoKEM-640-AES	1	1287	1810	1811
FrodoKEM-976-AES	3	2715	3572	3588
NH-512-CCA-KEM	1	117	181	206
NH-1024-CCA-KEM	5	245	377	437
Kyber512	1	142	205	246
Kyber768	3	243	333	394
Kyber1024	5	368	481	559

Table 3: Cycle counts (in thousands of cycles) of key generation, encapsulation and decapsulation for FrodoKEM, NewHope and Kyber (reference implementations).

Table 3 shows the cycle counts (in thousands of cycles) for key generation, encapsulation and decapsulation for the reference implementations of FrodoKEM, NewHope and Kyber. These are not quite comparable since different processors were used: FrodoKEM used a 3.4GHz Intel Core i7-6700 (Skylake), while NewHope and Kyber both used an Intel Core i7-4770K (Haswell).

Note that the version of FrodoKEM shown here uses AES128 to generate the matrix $\mathbf{A}$, and this depends on hardware instructions. The alternative is to generate $\mathbf{A}$ using cSHAKE128, which means a significant slowdown.

For the AES-version of FrodoKEM, AVX2 instructions does not noticeably improve performance, but cycle counts (again in thousands of cycles) for NewHope and Kyber optimised using AVX and AVX2[19] instructions respectively are shown in table 4.

It is apparent that the more structured problems RLWE and MLWE make a big difference for performance. This is at least partly because schemes building on the latter can be more compact, so that the main component of the public key in NewHope is just a polynomial, and that in Kyber is a short vector of polynomials (of lower degree), whereas FrodoKEM has an $n \times n$ (where n is

[19] AVX stands for Advanced Vector Extensions, and these operate on eigth 32-bit single-precision or four 64-bit double-precision floating-point values in parallell. AVX2 is an expansion of AVX.

Scheme	Security level	KeyGen	Encaps	Decaps
NH-512-CCA-KEM	1	68	110	114
NH-1024-CCA-KEM	5	130	210	221
Kyber512	1	55	76	74
Kyber768	3	85	113	109
Kyber1024	5	121	158	155

Table 4: Cycle counts (in thousands of cycles) of key generation, encapsulation and decapsulation for NewHope and Kyber (optimised).

640 or 976) matrix with integer entries. As a result, the generation of $\mathbf{A}$ in FrodoKEM takes much longer than the generation of $\mathbf{a}$ in NewHope and $\mathbf{A}$ in Kyber. Moreover, operations in the polynomial rings of NewHope and Kyber are very fast.

5 Kyber using MLWR

In the Kyber specification [41], it is suggested that Kyber could be adapted to rely on MLWR instead of MLWE, by removing the error terms $\mathbf{e}, \mathbf{e}_1$ and e_2, and possibly compressing a little, but the authors say that they choose not to do so as generating noise is not particularly costly.

However, it seems interesting to make the attempt. Algorithms 15, 16 and 17 show the algorithms for a version of Kyber.CPAPKE based on MLWR instead of MLWE.

Algorithm 15 LWRKyber.PKE.KeyGen(): key generation

Output: Public key $pk \in \mathcal{B}^{d_t \cdot k \cdot n/8+32}$
Output: Secret key $sk \in \mathcal{B}^{13 \cdot k \cdot n/8}$
1: $d \leftarrow \mathcal{B}^{32}$
2: $(\rho, \sigma) := G(d)$
3: **for** i from 0 to $k-1$ **do** ▷ Generate $\hat{\mathbf{A}} \in R_q^{k \times k}$ in NTT domain
4: **for** j from 0 to $k-1$ **do**
5: $\hat{\mathbf{A}}[i][j] := \mathsf{Parse}(\mathsf{XOF}(\rho||j||i))$
6: **for** i from 0 to $k-1$ **do** ▷ Sample $\mathbf{s} \in R_q^k$ from B_η
7: $\mathbf{s}[i] := \mathsf{CBD}_\eta(\mathsf{PRF}(\sigma, i))$
8: $\hat{\mathbf{s}} := \mathsf{NTT}(\mathbf{s})$
9: $\mathbf{t} := \mathsf{NTT}^{-1}(\hat{\mathbf{A}} \circ \hat{\mathbf{s}})$
10: $pk := (\mathsf{Encode}_{d_t}(\mathsf{Compress}_q(\mathbf{t}, d_t))||\rho)$ ▷ $pk := \lfloor \mathbf{As} \rceil_{2^{d_t}}$
11: $sk := \mathsf{Encode}_{13}(\hat{\mathbf{s}} \mod^+ q)$ ▷ $sk := \mathbf{s}$
12: **return** (pk, sk)

Algorithm 16 LWRKyber.PKE.Enc(pk, m, r): encryption

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

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

Algorithm 17 LWRKyber.PKE.Dec(sk, c): decryption

Input: Secret key $sk \in \mathcal{B}^{13 \cdot k \cdot n/8}$
Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8+d_v \cdot n/8}$
Output: Message $m \in \mathcal{B}^{32}$

$\mathbf{u} := \mathsf{Decompress}_q(\mathsf{Decode}_{d_u}(c), d_u)$
$v := \mathsf{Decompress}_q(\mathsf{Decode}_{d_v}(c + d_u \cdot k \cdot n/8), d_v)$
$\hat{\mathbf{s}} := \mathsf{Decode}_{13}(sk)$
$m := \mathsf{Encode}_1(\mathsf{Compress}_q(v - \mathsf{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \mathsf{NTT}(\mathbf{u})), 1))$ ▷
$m := \mathsf{Compress}_q(v - \mathbf{s}^T\mathbf{u}, 1)$
return m

5.1 Transform

For LWRKyber.KEM, I choose to keep the same transform used in Kyber. This gives the algorithms 18, 19 and 20. This transform has been shown to produce an IND-CCA secure KEM (in the QROM) by Jiang et al. in [22].

Algorithm 18 LWRKyber.KEM.KeyGen(): key generation

Output: Public key $pk \in \mathcal{B}^{d_t \cdot k \cdot n/8+32}$
Output: Secret key $sk \in \mathcal{B}^{(13+d_t) \cdot k \cdot n/8+96}$
1: $z \leftarrow \mathcal{B}^{32}$
2: $(pk, sk') :=$ LWRKyber.PKE.KeyGen()
3: $sk := (sk'||pk||H(pk)||z)$
4: **return** (pk, sk)

Algorithm 19 LWRKyber.KEM.Enc(pk): encapsulation

Input: Public key $pk \in \mathcal{B}^{d_t \cdot k \cdot n/8+32}$
Output: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8+d_v \cdot n/8}$
Output: Shared key $K \in \mathcal{B}^{32}$
1: $m \leftarrow \mathcal{B}^{32}$
2: $(\bar{K}, r) := G(m||H(pk))$
3: $c :=$ Kyber.CPAPKE.Enc$(pk, m; r)$
4: $K := H(\bar{K}||H(c))$
5: **return** (c, K)

5.2 Error probability

If the parameters are unchanged from Kyber, the probability of decapsulation error will be at most that of Kyber. According to Theorem 1 of [42], the probability of decryption error in Kyber.CPAPKE is

$$\delta = \Pr[||\mathbf{e}^T\mathbf{r} + e_2 + c_v - \mathbf{s}^T\mathbf{e}_1 - \mathbf{c}_t^T - \mathbf{s}^T\mathbf{c}_u||_\infty \geq \lfloor q/4 \rceil],$$

where $\mathbf{c}_t \leftarrow \psi_{d_t}^k, \mathbf{c}_u \leftarrow \psi_{d_u}^k, c_v \leftarrow \psi_{d_v}$ and ψ_d^k is defined as the distribution of $(\mathbf{y} - \mathsf{Decompress}_q(\mathsf{Compress}_q(\mathbf{y}, d), d) \mod^{\pm} q)$ where $\mathbf{y} \leftarrow R^k$.

For LWRKyber, this gives the error probability

$$\delta' = \Pr[||c_v - \mathbf{c}_t^T - \mathbf{s}^T\mathbf{c}_u||_\infty \geq \lfloor q/4 \rceil] \leq \delta.$$

Note that [42] claims that according to [19], the probability of decapsulation error in Kyber.CCAKEM is the same as the probability of decryption error in Kyber.CPAPKE (if G is a random oracle). However, it is not clear that this claim is supported by [19], which uses a different notion of correctness. Instead of (as in [42]) saying that a scheme is δ-correct if the probability of, essentially, stumbling upon a decryption error does not exceed δ, [19] assumes

Algorithm 20 LWRKyber.KEM.Dec(c, sk): decapsulation

Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$
Input: Secret key $sk \in \mathcal{B}^{(13+d_t) \cdot k \cdot n/8 + 96}$
Output: Shared key $K \in \mathcal{B}^{32}$

$pk := sk + 13 \cdot k \cdot n/8$
$h := sk + (13 + d_t) \cdot k \cdot n/8 + 32 \in \mathcal{B}^{32}$
$z := sk + (13 + d_t) \cdot k \cdot n/8 + 64$
$m' :=$ Kyber.CPAPKE.Dec($\mathbf{s}, (\mathbf{u}, v)$)
$(\bar{K}', r') := G(m'||h)$
$c' :=$ Kyber.CPAPKE.Enc($pk, m'; r'$)
if $c = c'$ **then**
    $K := H(\bar{K}'||H(c))$
else
    $K := H(z||H(c))$
return K

that there is an algorithm actively searching for decryption errors. With this notion of correctness, the transform T (which is part of the transform $\text{FO}^{\not\perp}$ used in Kyber) increases the probability of decryption error by a factor of at most q_G in the ROM and at most $8(q_G + 1)^2$ in the QROM, where q_G is the number of queries to the oracle G (which is used to generate the random coins, as the hash function of the same name in Kyber) made by an adversary against the correctness of the output PKE of T (see Section 2.4.8).

Using the "stumbling upon errors" notion of correctness, T does not increase the probability of decryption errors in the ROM (or the QROM), though as seen in [14] an adversary actively looking for decryption errors is a real threat.

5.3 Design rationale

Multiplication. In this version of LWRKyber we still use NTT for the multiplication. It would however be possible to use something else, e.g. Karatsuba, and one way to do this is of course to generate $\hat{\mathbf{A}}$ in the NTT domain and apply NTT^{-1} to it before using some other form of multiplication, but this would mean applying NTT^{-1} to each of the k^2 entries of $\hat{\mathbf{A}}$. If a different multiplication seems preferable to NTT, it seems better to build it into the scheme itself, and assuming that both key generation, encryption and decryption use this other form of multiplication, and that $\hat{\mathbf{A}}$ is not generated in the NTT domain. (In practice, it would be generated in exactly the same way, we would just not assume the result to be in the NTT domain.)

One argument for using a different multiplication algorithm is that Kyber inevitably has to apply NTT^{-1} more than for instance NewHope, which sends the public key and the vector $\mathbf{u}$ in the NTT domain. Kyber cannot do this as it would prevent using $\mathsf{Compress}_q$, and so it is certainly not an option in LWRKyber where security relies entirely on $\mathsf{Compress}_q$.

Centered binomial distribution. The algorithms still use the centered binomial distribution to generate $\mathbf{s}$ and $\mathbf{r}$. In theory, it would be possible to use something else, but as the size of $\mathbf{s}$ and $\mathbf{r}$ affect the probability of decryption errors they should be small, so they should not, for instance, be sampled uniformly as $\mathbf{A}$ is. Therefore we retain the centered binomial distribution.

Parameters. Since the shared key we arrive at is 256 bits, n should not be smaller than 256, otherwise multiple key bits must be encrypted into one polynomial coefficient, increasing the risk of decryption failure. On the other hand, to get the most out of the scalability given by the module structure, n should not be much larger than 256.

If using NTT for multiplication, it seems convenient to keep Kyber's parameter choices of n as a power of 2 and q a prime such that $2n|(q-1)$, specifically $n = 256, q = 7681$.

As for the other parameters, Kyber has k and η. These could stay about as they are in Kyber, but to be certain one would need to examine the effects on security and correctness. The parameters (d_u, d_v, d_t) decide how many bits are dropped in rounding, and for Kyber $(d_u, d_v, d_t) = (11, 3, 11)$. Since in LWRKyber we do not add errors but rely solely on rounding for security, we might consider a lower value for d_u and d_t, increasing the number of bits dropped in the rounding. The LWR-based NIST contribution Saber [43] uses $d_u = d_t = 10$, so this should be a feasible choice. If, however, decreasing d_u and d_t is not possible without the probability of decryption failure increasing significantly, one option to achieve the desired security might be to increase k slightly.

Extra hashes. Most of the extra hashes that appear in Kyber though they are not strictly necessary according to the security reduction seem meaningful for security and ease of implementation, and in the LWR version of Kyber these remain. The only one that is removed is the hash of the message m done immediately after sampling m in line 2 of algorithm 13, as this seems loosely motivated and indeed had been removed in the later paper [42].

The nonce. Since the nonce N, present in the KeyGen and Enc algorithms of Kyber.CPAPKE would have been used only in one for-loop in each of these algorithms in LWRKyber, it has been replaced with i (where the for-loop is over i) in LWRKyber. This should make very little actual difference, but seems neater.

5.4 Security

The IND-CPA security of LWRKyber.PKE is somewhat delicate. In [42] (Theorem 2), it is shown that if there exists an efficient attack on a version of Kyber.CPAPKE without rounding, there is an efficient attack on the MLWE problem, and the authors then assume that essentially the same security still holds when rounding is introduced into the scheme. This strategy does not work for LWRKyber.PKE, because a version of LWRKyber.PKE without rounding

would certainly not be secure: the public key would be $\mathbf{As}$, where $\mathbf{A}$ is also public, so $\mathbf{s}$ would not be secret for long.

Saber.PKE [43] is shown by Theorems 3 and 4 of that article to be as hard as MLWR. That PKE is formed from a key exchange protocol, but the end result is a PKE with algorithms that closely resemble those of LWRKyber.PKE. It therefore seems reasonable to suppose that a similar result would hold for LWRKyber.PKE.

Expressing the IND-CCA security of LWRKyber.KEM in terms of the IND-CPA security of LWRKyber.PKE, on the other hand, is easy enough. The transform used is that called $\text{FO}^{\not\perp}$ is [19], and Theorems 3.2 and 3.4 of that article give a bound for the security in the ROM. If there is an IND-CCA adversary B against LWRKyber.KEM making at most q_G queries to the random oracle G and at most q_H to H (and at most q_D queries to the decapsulation oracle), then there is an IND-CPA adversary A against LWRKyber.PKE with about the same running time as B such that

$$\mathbf{Adv}^{\text{IND-CCA}}_{\text{LWRKyber.KEM}}(B) \leq q_G \cdot \delta + \frac{2q_G + q_H + 1}{2^{256}} + 3\mathbf{Adv}^{\text{IND-CPA}}_{\text{LWRKyber.PKE}}(A),$$

where δ is the probability of decryption failure for LWRKyber.PKE and 2^{256} is the size of the message space.

Theorem 1 of [22] gives a security bound in the QROM.[20] That theorem, gives a bound for IND-CCA security in terms of OW-CPA security rather than IND-CPA, but using Lemma 2.3 in [19] this can easily be adapted to IND-CPA instead. This shows that if there is an IND-CCA adversary B against LWRKyber.KEM making at most q_G and q_H (quantum) queries to the random oracles G and H respectively (and at most q_D classical queries to the decapsulation oracle), then there is an IND-CPA adversary A against LWRKyber.PKE with about the same running time as B such that

$$\mathbf{Adv}^{\text{IND-CCA}}_{\text{LWRKyber.KEM}}(B) \leq \frac{2q_H}{\sqrt{2^{256}}} + 4q_G\sqrt{\delta} + 2(q_G+q_H)\sqrt{\mathbf{Adv}^{\text{IND-CPA}}_{\text{LWRKyber.PKE}}(A) + \frac{1}{2^{256}}},$$

where δ is the probability of decryption failure for LWRKyber.PKE and 2^{256} is the size of the message space. This bound could potentially be improved upon using the results of Ambainis, Hamburg and Unruh in [3].

5.5 Attacks

Just as Kyber analyses MLWE as an LWE problem, Saber analyses the hardness of MLWR as an LWE, arguing that there are no known attack that make use of

[20] Kyber uses [37] rather than [22] to prove QROM security, but the proofs in [37] (besides assuming the input PKE to be perfectly correct which Kyber compensates for) requires the PKE to fulfill a slightly stronger security notion than IND-CPA, and the argument for why Kyber.CPAPKE does so uses the rounding-free version of that PKE. A rounding-free version of LWRKyber.PKE would certainly not be secure, so it seems more appropriate to use the results of [22] for LWRKyber.PKE.

the module or LWR structure. LWRKyber has limited samples just like Kyber, so the possible attacks on the underlying MLWR problem as LWE would be the BKZ attacks, and these should have approximately the same costs. Saber has similar attack costs. (See [41] Section 5.1, [43] Section 6.1.)

LWRKyber will in general have nonzero probability of decapsulation failure, and the attack in [14] would be a threat. This attack costs about the same for Saber and Kyber, and the cost for attacking LWRKyber would presumably be similar. (See [43] Table 1.)

When it comes to side-channel attacks and multi-target attacks, LWRKyber is similar enough to Kyber that the same analysis holds. See [41] Section 4.5.

5.6 Making bigger changes

An alternative to the suggested scheme LWRKyber above would be one that more closely resembles the LWR-based Saber [43], which was also submitted to NIST[21]. The parameters of Saber have been chosen to balance performance, security and low probability of decryption error, and they suggest that choosing q and p as primes[22], while facilitating the use of NTT, introduces bias. Instead, they choose q and p as powers of two (specifically, $q = 2^{13}$ and $p = 2^{10}$) to ensure that the rounding preserves pseudorandomness. Because of this, they do not use NTT for multiplication, but instead four-way Toom-Cook multiplication, a generalisation of Karatsuba. The authors comment that this is asymptotically slower than NTT and that the possibility of reducing NTT transforms by e.g. generating $\hat{\mathbf{A}}$ in the NTT domain is also a benefit, but in practice Saber seems quite efficient. In the reference implementation it is faster than Kyber (about 70-80% of the number of cycles), though with AVX2 optimisations Kyber overtakes Saber.

Letting q be a power of two might not be a good idea, though, since several of the reductions from LWE to LWR require, if not prime q, then at least properties that are not true for power-of-two q, such as a lower bound for all prime factors (or at least the largest prime factor) of q. Therefore it seems preferable to keep q a prime unless this turns out to be a clear disadvantage to security in other ways, such as the reduction from MLWR to LWRKyber.PKE or the probability of decapsulation errors and information exposed by these. The Toom-Cook multiplication of Saber could still be brought into LWRKyber without changing the modulus, so an interesting compromise would be an LWR version of Kyber similar to LWRKyber as shown in algorithms 15 to 20 but with the following changes:

- $\hat{\mathbf{A}}$ is not generated in the NTT domain (this would make no actual difference in the generation of $\hat{\mathbf{A}}$ except calling it $\mathbf{A}$).
- All NTT transforms are removed.

[21] I could not find the original specification at the time of writing, as the NIST website is unavailable during the US government shutdown. However, an eprint from 2018 describes the scheme.

[22] Note that Kyber, and LWRKyber as suggested above, has prime q but power-of-two p.

- Toom-Cook multiplication is used where the current algorithms have coefficient-wise multiplication (denoted $\circ$).

Using Toom-Cook muliplication instead of NTT, it would also be possible to use a ring $R = \mathbb{Z}[X]/(f(X))$ for some other polynomial $f(X)$ than $X^n + 1$ with power-of-two n. Because security proofs for RLWE and MLWE (though not the one for MLWR in [9]) tend to require $f(X)$ to be cyclotomic we suggest that cyclotomic $f(X)$ be used, and that it should have degree at least 256 (the length of the shared key) to lessen the risk of decapsulation failures.

5.7 Conclusion

While the more structured ring- and module versions of the LWE problem do not have the same hardness proofs as the general version of the problem, and while some problems (such as GapSVP_γ) are easier to solve on ideal lattices, so far no attacks have emerged that present a threat in the parameters used in cryptosystems such as NewHope and Kyber, and the advantage of these over schemes such as Frodo based on more general lattices when it comes to speed and key sizes is so great that more structured schemes seem very promising for post-quantum secure KEMs. That said, a scheme like Frodo is a good candidate for situations where security is very important and speed and small key sizes less so.

As for LWR, studies so far have not produced useful security proofs but neither are there any known attacks that take advantage of the rounding, and as long as none emerge LWR and its structured variants RLWR and MLWR has the advantage of not needing to sample errors according to a Gaussian distribution (or a distribution that approximates one). However, if as in Saber the reduction from MLWR to a MLWR-based PKE depends on parameters that do not satisfy the requirements of the many security proofs, this is troubling. (Specifically, the reduction in Saber has power-of-two modulus, and while this is allowed in some of the security proofs for MLWR it rules out others.) Both these areas would benefit from further study, but MLWR-based schemes still seem promising.

References

[1] Alperin-Sheriff J., Apon D. (2016) Dimension-Preserving Reductions from LWE to LWR. Cryptology ePrint Archive, Report 2016/589. `https://eprint.iacr.org/2016/589`

[2] Albrecht M.R., Deo A. (2017) Large Modulus Ring-LWE $\geq$ Module-LWE. In: Takagi T., Peyrin T. (eds) Advances in Cryptology – ASIACRYPT 2017. ASIACRYPT 2017. Lecture Notes in Computer Science, vol 10624. Springer, Cham

[3] Ambainis A., Hamburg M., Unruh D. (2018) Quantum security proofs using semi-classical oracles. Cryptology ePrint Archive, Report 2018/904. https://eprint.iacr.org/2018/904

[4] Alwen J., Krenn S., Pietrzak K., Wichs D. (2013) Learning with rounding, revisited. In Advances in Cryptology CRYPTO 2013, pages 57-74. Springer, 2013.

[5] Babai, L. (1986) On Lovász' lattice reduction and the nearest lattice point problem. In: Combinatorica 6:1

[6] D. Boneh, Ö. Dagdelen, M. Fischlin, A. Lehmann, C. Schaffner and M. Zhandry (2011) Random Oracles in a Quantum World. In: Lee D.H., Wang X. (eds) Advances in Cryptology – ASIACRYPT 2011. ASIACRYPT 2011. Lecture Notes in Computer Science, vol 7073. Springer, Berlin, Heidelberg

[7] Bellare M., Desai A., Pointcheval D., Rogaway P. (1998) Relations among notions of security for public-key encryption schemes. In Krawczyk H. (ed) Advances in Cryptology — CRYPTO '98. CRYPTO 1998. Lecture Notes in Computer Science, vol 1462. Springer, Berlin, Heidelberg

[8] D. J. Bernstein (2001) Multidigit multiplication for mathematicians. http://cr.yp.to/papers/m3.pdf

[9] Bogdanov A., Guo S., Masny D., Richelson S., Rosen A. (2016) On the Hardness of Learning with Rounding over Small Modulus. In Kushilevitz E., Malkin T. (eds) Theory of Cryptography. TCC 2016. Lecture Notes in Computer Science, vol 9562. Springer, Berlin, Heidelberg

[10] Bernstein D., Persichetti E. (2018) Towards KEM Unification. Cryptology ePrint Archive, Report 2018/526. https://eprint.iacr.org/2018/526

[11] Banerjee A., Peikert C., Rosen A. (2012) Pseudorandom Functions and Lattices. In Pointcheval D., Johansson T. (eds) Advances in Cryptology – EUROCRYPT 2012. EUROCRYPT 2012. Lecture Notes in Computer Science, vol 7237. Springer, Berlin, Heidelberg

[12] Bellare M., Rogaway P. (1993) Random oracles are practical: A paradigm for designing efficient protocols. In: Proc. of ACM Conference on Computers and Communication Security, pages 62-73, 1993.

[13] Chunsheng G. (2017) Integer Version of Ring-LWE and its Applications. Cryptology ePrint Archive, Report 2017/641. https://eprint.iacr.org/2017/641

[14] J.-P. D'Anvers, F. Vercauteren, I. Verbauwhede (2018) On the impact of decryption failures on the security of LWE/LWR based schemes. Cryptology ePrint Archive, Report 2018/1089. https://eprint.iacr.org/2018/1089

[15] Dadush, D., Regev, O., Stephens-Davidowitz, N. (2014) On the Closest Vector Problem with a Distance Guarantee. Proceedings of the Annual IEEE Conference on Computational Complexity. 10.1109/CCC.2014.18.

[16] R. Fateman (2010) Can You Save Time in Multiplying Polynomials By Encoding Them as Integers? `https://people.eecs.berkeley.edu/~fateman/papers/polysbyGMP.pdf`

[17] Fujisaki E., Okamoto T. (1999) Secure integration of asymmetric and symmetric encryption schemes. In Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (Aug 1999)

[18] Fujisaki, E., Okamoto, T. (2013) Secure integration of asymmetric and symmetric encryption schemes. J. Cryptol. 26(1), 80–101

[19] Hofheinz D., Hövelmanns K., Kiltz E. (2017) A Modular Analysis of the Fujisaki-Okamoto Transformation. In: Kalai Y., Reyzin L. (eds) Theory of Cryptography. TCC 2017. Lecture Notes in Computer Science, vol 10677. Springer, Cham

[20] Hövelmanns K., Kiltz E., Schäge S., Unruh D. (2018) Generic Authenticated Key Exchange in the Quantum Random Oracle Model. Cryptology ePrint Archive, Report 2018/928. `https://eprint.iacr.org/2018/928`

[21] Hoffstein, Pipher, Silverman *An Introduction to Mathematical Cryptography*, Springer, 2008.

[22] Jiang H., Zhang Z., Chen L., Wang H., Ma Z. (2018) IND-CCA-Secure Key Encapsulation Mechanism in the Quantum Random Oracle Model, Revisited. In: Shacham H., Boldyreva A. (eds) Advances in Cryptology – CRYPTO 2018. CRYPTO 2018. Lecture Notes in Computer Science, vol 10993. Springer, Cham

[23] Lyubashevsky V., Peikert C., Regev O. (2010) On Ideal Lattices and Learning with Errors over Rings. In: Gilbert H. (eds) Advances in Cryptology – EUROCRYPT 2010. EUROCRYPT 2010. Lecture Notes in Computer Science, vol 6110. Springer, Berlin, Heidelberg

[24] V. Lyubashevsky, C. Peikert, O. Regev (2013) A Toolkit for Ring-LWE Cryptography. In: Johansson T., Nguyen P.Q. (eds) Advances in Cryptology – EUROCRYPT 2013. EUROCRYPT 2013. Lecture Notes in Computer Science, vol 7881. Springer, Berlin, Heidelberg

[25] Langlois A., Stehlé D. (2012) Worst-Case to Average-Case Reductions for Module Lattices. Cryptology ePrint Archive, Report 2012/090. `https://eprint.iacr.org/2012/090`

[26] Montgomery H. (2018) A Nonstandard Variant of Learning with Rounding with Polynomial Modulus and Unbounded Samples. In: Lange T., Steinwandt R. (eds) Post-Quantum Cryptography. PQCrypto 2018. Lecture Notes in Computer Science, vol 10786. Springer, Cham

[27] Mosca M. (2015) Cybersecurity in an era with quantum computers: will we be ready?. Cryptology ePrint Archive, Report 2015/1075. https://eprint.iacr.org/2015/1075

[28] D. Micciancio, O. Regev (2008) Lattice-based Cryptography. https://cims.nyu.edu/~regev/papers/pqc.pdf

[29] Micciancio D., Peikert C. (2013) Hardness of SIS and LWE with Small Parameters. In: Canetti R., Garay J.A. (eds) Advances in Cryptology – CRYPTO 2013. CRYPTO 2013. Lecture Notes in Computer Science, vol 8042. Springer, Berlin, Heidelberg

[30] Peikert C. (2015) A Decade of Lattice Based Cryptography. Cryptology ePrint Archive, Report 2015/939 https://eprint.iacr.org/2015/939

[31] Pietrzak K. (2012) Cryptography from learning parity with noise. In SOFSEM 2012: Theory and Practice of Computer Science, pages 99- 114. Springer, 2012.

[32] Roşca M., Sakzad A., Stehlé D., Steinfeld R. (2017) Middle-product learning with error. Cryptology ePrint Archive, Report 2017/628. https://eprint.iacr.org/2017/628

[33] D. Ristè, M. P. da Silva, C. A. Ryan, A. W. Cross, A. D. Córcoles, J. A. Smolin, J. M. Gambetta, J. M. Chow, B. R. Johnson (2017) Demonstration of quantum advantage in machine learning. In npj Quantum Information 3, Article number: 16

[34] Schnorr, C.P., Euchner, M. (1994) Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical Programming 66: 181. Springer-Verlag.

[35] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput., 26(5):1484–1509, October 1997.

[36] Shibuya Y., Shikata J. (2011) Constructing Secure Hybrid Encryption from Key Encapsulation Mechanism with Authenticity. In: Chen L. (eds) Cryptography and Coding. IMACC 2011. Lecture Notes in Computer Science, vol 7089. Springer, Berlin, Heidelberg

[37] Saito T., Xagawa K., Yamakawa T. (2018) Tightly-Secure Key-Encapsulation Mechanism in the Quantum Random Oracle Model. In: Nielsen J., Rijmen V. (eds) Advances in Cryptology – EUROCRYPT 2018. EUROCRYPT 2018. Lecture Notes in Computer Science, vol 10822. Springer, Cham

[38] NIST (2006) Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf

[39] M. Naehrig, E. Alkim, J. Bos, L. Ducas, K. Easterbrook, B. LaMacchia, P. Longa, I. Mironov, V. Nikolaenko, C. Peikert, A. Raghunathan, D. Stebila (2017) Specification of FrodoKEM submitted to NIST.

[40] T. Poppelmann, E. Alkim, R. Avanzi, J. Bos, L. Ducas, A. de la Piedra, P. Schwabe, D. Stebila (2017) Specification of NewHope submitted to NIST.

[41] P. Schwabe, R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, G. Seiler, D. Stehlé (2017) Specification of CRYSTALS-KYBER submitted to NIST. `https://pq-crystals.org/kyber/data/kyber-specification.pdf`

[42] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, D. Stehlé (2018) CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. 2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018. `https://pq-crystals.org/kyber/data/kyber-20180716.pdf`

[43] J.-P. D'Anvers, A. Karmakar, S. Sinha Roy, and F. Vercauteren (2018) Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. Cryptology ePrint Archive, Report 2018/230. `https://eprint.iacr.org/2018/230`

[44] M. Hamburg (2017) Specification of ThreeBears submitted to NIST.

[45] Steinfeld R., Sakzad A., Kuo Zhao R. (2017) Specification of Titanium submitted to NIST.