On Building a Post-Quantum Secure Lattice-Based Key Encapsulation Tool

Georges André Paul Schmoetten

1862518

June 2018 - September 2018

School of Computer Science

University of Birmingham

Supervisor: Dr. David Galindo

This manuscript is submitted in partial fulfilment of the requirements for the degree of MSc Computer Science at the University of Birmingham.

Abstract

This project resulted in the successful creation of a post-quantum secure, lattice-based key encapsulation tool based on the New Hope and CRYSTALS Kyber cryptosystems, implemented in Python 3.7. The tool provides access to the key generation, encapsulation and decapsulation functions of all valid parameter configurations of the following three KEMs: New Hope IND-CPA, New Hope IND-CCA and Kyber IND-CCA. The project further served to significantly increase our understanding of cryptography in general and post-quantum, specifically lattice-based, cryptography in particular. Additionally, it gave us a solid introduction to the mathematical background of lattice-based cryptography and vastly increased our proficiency in the Python programming language.

The general aim of the project was to create a post-quantum secure tool allowing parties to establish shared secrets via insecure channels and in the process gain expertise in the area of post-quantum cryptography, previously unknown to us, as well as to expand both our computational and mathematical toolkit. Having met all these criteria, the project is considered a success.

Acknowledgements

I would like to thank my supervisor, Dr. David Galindo, for his guidance and advice throughout this project.

Contents

1 Introduction

The aim of this project was to create a tool for key encapsulation using two lattice-based cryptographic protocols from among those submitted to the NIST call for proposals for post-quantum standardisation; the two systems were to be chosen such that their security was based on the hardness of different lattice-based problems, to impart robustness and increased resistance to future developments in cryptanalysis to the tool. To the author's best knowledge, no such tools exist as of this writing. The motivation behind it was to provide a convenient tool for two parties to use, in conjunction with any insecure communication channel, in order to obtain a shared 256-bit secret for symmetric encryption, while minimizing the number of operations and transmissions required.

The project was divided into two parts: the first part was devoted to developing an understanding of post-quantum cryptography in general, and lattice-based cryptography in particular. This included establishing a working knowledge of the mathematical structure and properties of lattices, as well as gaining familiarity with the various computationally hard problems based on lattices and the different learning problems derived from them. It further included an inspection of the relevant NIST submissions, in order to make an informed decision on the matter of which two of the submitted systems to pick for implementation.

The second part of the project was focused on the actual implementation of the two key encapsulation mechanisms and the creation of a user interface to allow convenient access to all the required functionalities: these include choice of KEM and security level, key generation, encapsulation using a recipient's public key and decapsulation using a recipient's private key and a suitable ciphertext. The implementation was to be done in Python, a language the author had only limited experience with, hence this part of the project also included a dedicated period of re-introduction to Python.

We begin this paper by providing the cryptographic and the mathematical background required for the reader to appreciate the key encapsulation tool built in this project and the motivation behind its creation. Having established these fundamentals, we then present a review of the major scientific works that have led to the inception of the cryptographic systems the tool is based on. Finally, we discuss the design and implementation processes of our key encapsulation tool, and present the tool itself. We then conclude this paper with a review of the project's main challenges and ideas for extending work.

2 Cryptographic Background

2.1 The Evolution of Cryptography

Classical cryptography, the practice of altering messages in order to conceal their true meaning from anyone but the intended recipient, is a discipline over two millennia old that has been employed in many civilised cultures. Egyptian tombs show first signs of its advent; in ancient Greece, the Spartan military made use of transposition ciphers to protect sensitive information and similarly, in ancient Rome, Julius Ceasar himself is said to have instigated the use of substitution ciphers to communicate with his generals. [29] This use of so-called symmetric or private-key cryptography as a way to encode and decode messages following a certain algorithm prevails to this day. Yet in the 20^{th} century, with the inception of modern computers and the Internet, the uses of cryptography grew beyond the secret-keeping of messages and the discipline of cryptography itself grew into a proper science.

One of the major changes brought about by the development of modern computers was that cryptography was no longer restricted to being performed by humans using pen and paper. Cryptographic algorithms could henceforth become vastly more complex, since computers could perform the underlying calculations much faster than humans, making even intricate algorithms practically useful. However, the security of encryption algorithms was in equal measure threatened by the technological advances as they triggered equivalent evolutions in the field of cryptanalysis, which concerns itself with the breaking of cryptographic constructs. The hardness of the mathematical problems at the root of the intractability of cryptographic primitives therefore grew exponentially.

A second ground-breaking revolution in the field of cryptography and growth of the discipline was triggered by the advent of the Internet. So far, cryptography had mainly been a tool for acquainted parties to exchange communications without any third parties being able to read them. However, the Internet allowed parties from all over the world to connect with each other with hitherto unknown ease; frequently, the parties entering into contact with each other would never even have met beforehand. This gave rise to a plethora of new challenges: authenticating digital documents whose origins may be in doubt, verifying the identities of different parties entering into an online transaction, keeping people from going back on their online agreements, and many more. The development of technology thus led to the creation of new tools, like digital signatures and authentication procedures, as well as new facets and branches of cryptography, like public key cryptography.

2.2 Private-Key vs. Public-Key Cryptography

2.2.1 Private-Key Cryptography

Private-key cryptography, also known as symmetric cryptography, is used by two parties, Alice and Bob, who wish to encrypt their communications, using a shared secret as the key, in order to prevent a potential eavesdropper, Eve, from reading the messages. A symmetric cryptographic scheme provides two operations, encryption and decryption, that both make use of the same key; hence the name "symmetric". The nature of the key depends on the nature of the encryption algorithm Alice and Bob wish to use: it may be a phrase, it may be a string of numbers, an array of bytes, etc. The crucial point is that Alice and Bob must have a secure way to establish this shared secret: they could for example meet in person and agree on the key, but may equally well use any other form of correspondence they trust will preclude Eve from listening in and gaining the key.

It may be argued that, if Alice and Bob already have a secure form of communication at their disposal, then the act of establishing a key for another mode of communication is redundant. The case may be however, that the current secure communication channel is only temporary, yet a permanent one is required; for example, the two may happen to be in the same city, and thus able to meet in person, for a few days only, but they need to continue corresponding beyond that time frame, say via e-mail. They would hence meet in person, establish a shared secret and henceforward use that to encrypt their e-mail messages. Alternatively, whatever secure channel is already in place might not be suitable to pass across the kind of message that Alice and Bob need to exchange; if, for example, they have a secure phone line, but need to pass long lists of numbers to each other, the channel is not well-suited to the prospective payload.

Having established the shared secret, Alice now wishes to send a message to Bob. She composes the message and then uses the agreed-upon method of encryption, with the shared secret as key, to encrypt the message; the resulting unintelligible message is called the ciphertext. This ciphertext is what she now puts in an e-mail and sends to Bob. Eve, who is monitoring the communication line between the two, sees the message, but, not having the correct key, cannot decrypt the message and hence gains no information from it. Bob on the other hand, the intended recipient, uses the agreed-upon method of decryption, with the shared secret as key, to decrypt the message; he thus retrieves the original message composed by Alice. A schematic representation of the functioning of a symmetric encryption scheme is given in figure 1.

The upsides of symmetric cryptography are that it is computationally fast, compared to asymmetric cryptography, and the key sizes are small. The Advanced Encryption Standard (AES), which is the most widely used symmetric key algorithm, comes in different security levels, the strongest of which, AES-

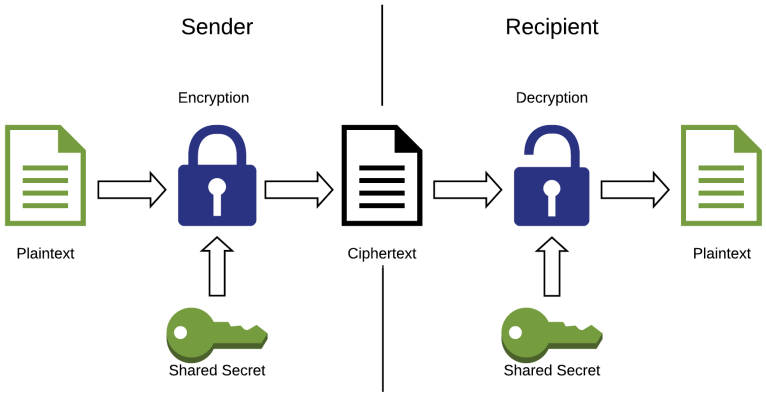

Figure 1: A schematic representation of symmetric encryption.

256, still only uses a 256-bit key. The downsides of symmetric cryptography are the issues of key establishment and key management: imagine a company that requires all communications between its employees to be encrypted. This would require that every pair of employees had a shared secret to use for their communications. In a company of n employees, this leads to the need for up to $n * (n - 1)/2$ keys. [21]

2.2.2 Public-Key Cryptography

Public-key cryptography, or asymmetric cryptography, addresses two of the detriments of private-key cryptography: the key management issue and the key establishment problem. In asymmetric cryptography, a key is no longer associated with two parties that wish to communicate with each other. Instead, every party now has two keys associated with it: one public key and one private, or secret, key. The public key is made publicly available so that anyone that may wish to enter into encrypted communication with the party that published it may find it and use it. The secret key is kept secret by the owner. The reason for the term "asymmetric" cryptography is the fact that, unlike for symmetric cryptography, encryption and decryption in a public-key system do not use the same key. Instead, encryption of a message is done using the publicly available key of the party one wishes to communicate with, while decryption is done by the recipient, using their secret key. This ensures that everyone is able to perform encryption, since everyone may use the public key, but only the person that generated the keys can perform decryption, since they alone know the secret key. Public-key cryptography is heavily used in network and internet protocols. [16]

A public-key cryptographic scheme provides three operations: key generation, which yields a private key/public key pair, encryption, using the public key, and decryption, using the private key. Imagine Alice is a person of public renown in her professional field: people she does not know frequently wish to communicate with her, yet, as the correspondence may include sensitive information, Alice prefers for the communications to be encrypted. She executes the key generation function of her chosen public-key cryptographic scheme and posts the public key on her official page on her company's website, along with the name of the scheme she used to generate it; the private key she keeps secret. Bob wishes to pitch a business idea to her, so he writes out a short introduction, encrypts it using Alice's public key and the cryptographic scheme she specified, and sends her the resulting ciphertext. Eve, a competitor, has managed to break into Bob's e-mail account and sees the message that was sent to Alice. However, since she is not in possession of Alice's secret key, she cannot decrypt the message, despite knowing what key and scheme were used to encrypt it. Alice on the other hand, using her secret key, decrypts the message she received from Bob and recovers the original introduction he wrote. Figure 2 depicts the steps of a public-key encryption/decryption cycle; it is assumed that the keys were appropriately generated and that the recipient made their public key available to the sender.

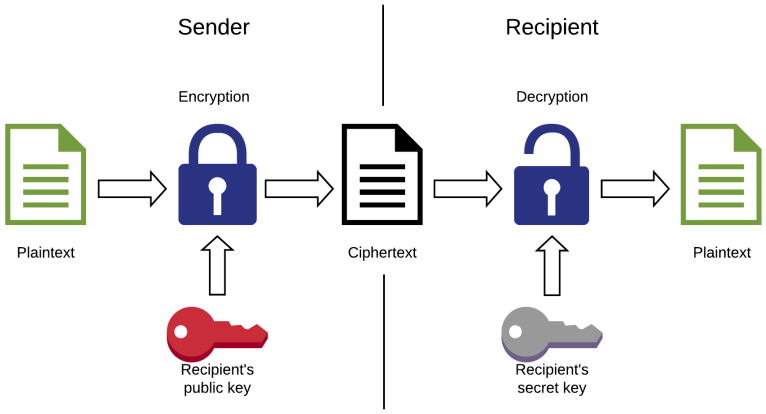

Figure 2: A schematic representation of asymmetric encryption.

Compared to symmetric cryptography, public-key cryptography removes the need for two parties to agree on a shared secret. Furthermore, re-using the previous example of the company that demands that all internal communications between its n employees be encrypted, the use of asymmetric cryptography leads to a need for only $2 * n$ keys, one public and one private key per employee, as compared to the $n * (n - 1)/2$ required with symmetric encryption. For a

company with a large number n of employees, this dramatically reduces the key management issue. On the other hand, the key-sizes required for secure public-key cryptographic schemes are far larger than those required for private-key schemes: one of the most common public-key cryptographic systems, RSA, has a minimum recommended key-length of 2048 bits with 3072 bits recommended for future-proofing. In addition to this, asymmetric encryption can generally only be used on messages of a specific length; sending anything longer would hence require the message to be split into chunks, and each chunk to be encrypted and sent individually. In comparison to this, messages of virtually any size can be encrypted and sent using symmetric encryption. [21]

2.2.3 Combining the two: Hybrid Encryption

Hybrid Encryption refers to a joint use of symmetric and asymmetric cryptographic algorithms. If Alice wants to send a message to Bob, but the two of them do not have a shared secret yet, she generates a shared secret, symmetrically encrypts her message with it, then asymmetrically encrypts the shared secret using Bob's public key, and sends Bob the two ciphertexts. He now uses his secret key to asymmetrically decrypt the first ciphertext and hence obtain the shared secret. He then uses this shared secret to symmetrically decrypt the second ciphertext and thus retrieves Alice's message. Hybrid schemes make use of the strengths of both kinds of encryption. [16]

2.3 Key Exchange

A key exchange or key establishment process refers to any method via which two parties agree on a shared key for use in symmetric encryption. This used to require some sort of secure physical communication channel, i.e. meeting face to face or sending couriers. As seen in the previous section, public-key cryptography is another tool that can be used. The first and perhaps most renowned cryptographic system that allows key establishment via an insecure communication channel, using only "forward" functions, as opposed to "forward" encryption followed by "backward" decryption, is the Diffie-Hellman key exchange (DH). [21]

The concept is readily explained, without going into the underlying mathematics, using a colour analogy, as depicted in figure 3. [29] Alice and Bob agree on a base colour; there is no need to keep this secret from potential adversaries, so they can use an insecure channel to establish this. Both of them then, independently of each other, add a colour, which they choose and keep secret, to the agreed-upon base colour. They exchange the resulting mixtures. Alice now has the shared colour mixed with Bob's colour, Bob has the shared colour mixed with Alice's colour. All that remains to do is for Alice to add her secret colour to the mix of the base colour and Bob's colour, and for Bob to add his secret colour to the mix of the base colour and Alice's colour. After those two operations, both Alice and Bob are now in possession of the same shared,

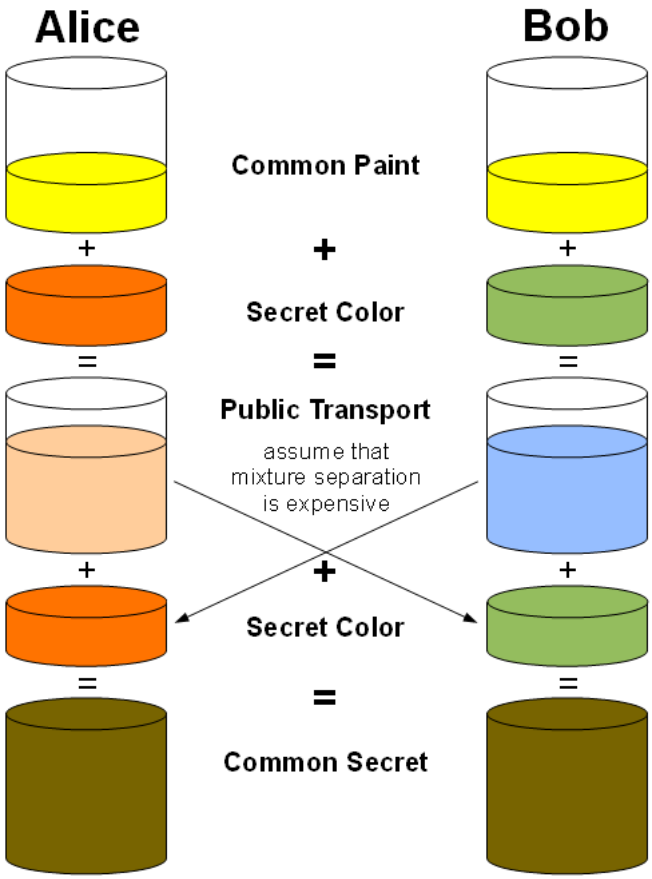

Figure 3: The Diffie-Hellman key exchange illustrated using a colour analogy. [10]

secret colour, even if the components in their respective mixtures were added in different orders. Note that colours were only ever added, and never removed. If we assume that an adversary who witnessed the exchange, so who knows what the two intermediate mixtures looked like, and who knows the base colour, is nonetheless unable to split the mixtures back up into their component colours, then the adversary cannot retrieve Alice's and Bob's individual secret colours. As a consequence, they will have no way of reconstructing the shared, secret colour, which hence is a secure shared secret for Alice and Bob.

This method of key establishment requires the following operations: the establishment of the base colour via an insecure channel; the independent manipulation of the base colour by each party; the exchange of colours, i.e. two transmissions via an insecure channel; a further independent manipulation of

the mixtures by each party.

2.4 Key Encapsulation

An alternative way for two parties to establish a shared secret via an insecure channel is by use of a Key Encapsulation Mechanism (KEM). Such a mechanism provides three operations: key generation, encapsulation using a public key and decapsulation of a ciphertext using a secret key. The encapsulation operation yields two outputs: a shared secret and a ciphertext. The ciphertext is transmitted to the intended recipient, who, using their secret key and the ciphertext as arguments, runs the KEM's decapsulation operation. This yields the same shared secret that the encapsulator obtained from the encapsulation operation. Using a KEM over a PKE hence saves the user the need to separately generate the shared secret.

As is suggested by the similarity to the operations of a public-key encryption (PKE) scheme, a KEM is generally based on a PKE. The use of a KEM is very similar to the use of public-key encryption in the discussion of hybrid schemes above and indeed, KEMs tend to be preferred to PKEs in modern hybrid systems. A difference between the two lies in the fact that the PKE encryption operation requires a message that is to be encrypted as an argument, alongside the recipient's public key. A KEM's encapsulation operation, on the other hand, requires only the recipient's public key as argument, since the message that is going to be encrypted, the prospective shared secret, is generated as part of the operation; this is what is referred to as encapsulation in this context. Furthermore, while PKEs use padding to ensure that the shared secret is of the correct size to be encrypted, KEMs remove the need for padding by hashing a randomly generated element to an output of the desired length.

Going back to the situation described while introducing public-key cryptography, assume that Bob now wants to enter into contact with Alice by making use of a KEM rather than a pure public-key cryptography scheme: he runs the encapsulation operation using the key for her favourite KEM that Alice posted on her website. He sends the resulting ciphertext to Alice while keeping the shared secret to himself. Alice decapsulates the ciphertext she receives using her secret key, which yields the shared secret. This she uses as a key to contact Bob via symmetric encryption. They are now in a position where they can communicate via fast symmetric encryption without any further need for asymmetric cryptography. Establishing a shared secret using this procedure takes the following operations, assuming that the relevant keys have been generated and are publicly available as appropriate: one key encapsulation, one message transmission and one decapsulation. Comparing this to the DH key exchange mechanism described earlier, the KEM requires fewer operations and, importantly, fewer transmissions to establish a shared secret.

2.5 The Need for Post-Quantum Security

In today's world, computers are so commonplace and ubiquitous that in many situations, we may not even be quite aware that we are dealing with them. Aside from the fact that virtually all professional, and many social and recreational, aspects of our lives are directly reliant on computers and their secure and unimpeded connection to other computers, usually via the Internet, a vast amount of things that we may not directly think of as computers now depend on them too. Modern cars, fridges, ovens, doorbells and even washing machines all come equipped with small bespoke computers, potentially connected to the Internet, without which they cannot function. The correct functioning of computers in the context of a network as complex as the Internet directly implies the correct functioning of cryptography. Unfortunately however, the most wide-spread public-key cryptographic primitives used today are threatened by an innovative area of research that has seen tireless development since the 1980s: quantum computing.

In 1994, Peter Shor developed a quantum algorithm for integer factorization, known as Shor's algorithm, that poses an existential threat to public-key cryptographic primitives like RSA and El Gamal, by rendering their underlying mathematical problems, the factorization of large integers and the calculation of discrete logarithms, respectively, computationally tractable. Further to these two, primitives based on elliptic-curve discrete logarithms would no longer be secure either. A second quantum algorithm, developed in 1996 by Lov Grover, finds some application in the breaking of symmetric cryptographic primitives like DES, but the current standard, AES, seems to be resistant given large enough key sizes. [17]

Although no quantum computers large enough to threaten current-day cryptography have been developed yet, it may be nothing but a matter of time. In order to prepare for this scenario, it was deemed wise to start exploring cryptographic primitives based on mathematical problems expected to be just as hard to break using a quantum computer as they are using a classical binary computer. Current candidates to fill this position and become the new standard for public-key cryptography include lattice-based cryptography, multivariate cryptography, hash-based cryptography and code-based cryptography. [30] These approaches to public-key cryptography were not all invented once the need for post-quantum security was agreed upon; rather, most of these had been known and studied to some extent beforehand and merely gained in popularity when they were deemed suitable to rise to this new challenge and be part of the next generation of cryptographic primitives.

In December 2016, the National Institute of Standards and Technology (NIST) in the USA issued a call for proposals for post-quantum cryptographic algorithms with the goal of determining a post-quantum cryptographic standard. The submission deadline was on the 30^{th} November 2017 and, as of this writing, the submissions are being evaluated. [30]

3 Mathematical Background

This section serves as a short introduction to the mathematical concepts used in the later parts of this work, like those of *lattices* and *rings*. It is by no means exhaustive, but ought to help the reader's understanding of the later sections. Familiarity with the basic concepts of vectors and matrices is assumed.

3.1 Lattices

Definition 3.1 (Lattice)**.** [12] Let $\mathbf{v}_1, ..., \mathbf{v}_n \in \mathbb{R}^m$ be a set of linearly independent vectors. The *lattice* $\mathcal{L}$ *generated by* $\mathbf{v}_1, ..., \mathbf{v}_n$ is the set of linear combinations of $\mathbf{v}_1, ..., \mathbf{v}_n$ with coefficients in $\mathbb{Z}$,

$$\mathcal{L} = \{a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + ... + a_n\mathbf{v}_n : a_1, a_2, ..., a_n \in \mathbb{Z}\}.$$

The integer n is called the *rank* of the lattice. If $n = m$, the lattice is said to be *full rank*. A lattice can be thought of as a subset of a vector space, comprising only those points given by linear combinations with integer coefficients of the basis vectors. [9] Figure 4 shows an example of a 2-dimensional lattice.

A *basis* for $\mathcal{L}$ is any set of independent vectors that generates $\mathcal{L}$; this set can be represented as a *generator matrix* $\mathbf{B}$, whose columns are the vectors that constitute the set. Any two bases have the same number of elements, i.e. if represented in matrix form, both generator matrices have the same dimension. The *dimension* of $\mathcal{L}$ is the number of vectors in a basis for $\mathcal{L}$. [12]

Theorem 3.1. [9] *Two matrices* $\mathbf{B}$ *and* $\mathbf{B}'$ *generate the same lattice if and only if there exists a unimodular matrix* $\mathbf{U}$ *such that* $\mathbf{B}'\mathbf{U} = \mathbf{B}$.

Definition 3.2 (Additive subgroup)**.** [12] A subset $\mathcal{L}$ of $\mathbb{R}^m$ is an *additive subgroup* if it is closed under addition and subtraction. It is called a *discrete additive subgroup* if there is a positive constant $\epsilon > 0$ with the following property: for every $\mathbf{v} \in \mathcal{L}$,

$$\mathcal{L} \cap \{\mathbf{w} \in \mathbb{R}^m : \|\mathbf{v} - \mathbf{w}\| < \epsilon\} = \{\mathbf{v}\}.$$

To illustrate this, imagine drawing a ball of radius ϵ around the tip of any vector $\mathbf{v}$ of $\mathcal{L}$; if $\mathcal{L}$ is a discrete additive subgroup, the ball will contain no points of $\mathcal{L}$ other than $\mathbf{v}$ itself.[12]

Theorem 3.2. [12] *A subset of* $\mathbb{R}^m$ *is a lattice if and only if it is a discrete additive subgroup.*

A lattice can be defined by what is referred to as its *fundamental domain*; this is intricately linked to the lattice's *determinant* or *covolume*.

Definition 3.3 (Fundamental Domain)**.** [12] Let $\mathcal{L}$ be a lattice of dimension n and let $\mathbf{v}_1, \mathbf{v}_2, ...\mathbf{v}_n$ be a basis for $\mathcal{L}$. The *fundamental domain* (or *fundamental parallelepiped*) for $\mathcal{L}$ corresponding to this basis is the set

$$\mathcal{F}(\mathbf{v}_1, ...\mathbf{v}_n) = \{t_1\mathbf{v}_1 + t_2\mathbf{v}_2 + ... + t_n\mathbf{v}_n : 0 \leq t_i < 1\}.$$

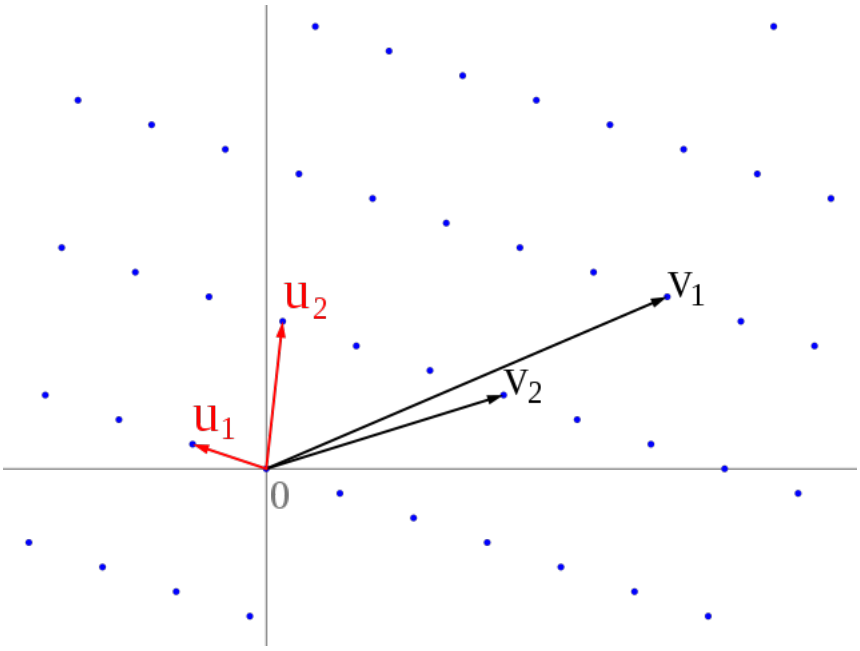

Figure 4: A 2-dimensional lattice with two bases, $\{\mathbf{u}_1, \mathbf{u}_2\}$ and $\{\mathbf{v}_1, \mathbf{v}_2\}$; the $\mathbf{u}_i$ form a good basis, the $\mathbf{v}_i$ form a bad one. [8]

Definition 3.4 (Determinant)**.** [12] Let $\mathcal{L}$ be a lattice of dimension n and let $\mathcal{F}$ be a fundamental domain for $\mathcal{L}$. Then the n-dimensional volume of $\mathcal{F}$ is called the *determinant of* $\mathcal{L}$ or the *covolume of* $\mathcal{L}$. It is denoted by $det(\mathcal{L})$.

Definition 3.5 (Euclidean Length)**.** The *Euclidean length* or *Euclidean norm* $\|\mathbf{v}\|$ of a vector $\mathbf{v}$ is the square root of the sum of the squares of its elements v_i.

$$\|\mathbf{v}\| = \sqrt{\sum_{i=1}^{n} v_i^2}$$

Definition 3.6 (Minimum Distance)**.** [22] The *minimum distance* of a lattice $\mathcal{L}$ is the Euclidean length of a shortest nonzero lattice vector.

$$\lambda_1(\mathcal{L}) = \min_{\mathbf{v} \in \mathcal{L} \setminus \{0\}} \|\mathbf{v}\|$$

Lattice bases are sometimes referred to as "good" or "bad". A "good" basis is one whose vectors are reasonably close to being orthogonal; a "bad" basis is one whose vectors only form small angles. Figure 4 shows an example of two bases that generate the same lattice; basis $\{\mathbf{u}_1, \mathbf{u}_2\}$ has highly orthogonal vectors and is "good", whereas basis $\{\mathbf{v}_1, \mathbf{v}_2\}$ has highly non-orthogonal vectors and is "bad". In order to be able to determine whether a basis is "good" or

"bad", one can calculate the basis' *Hadamard ratio*; the closer it is to 1, the better the basis.

Definition 3.7 (Hadamard ratio)**.** [12] Let $\mathbf{B} = \{\mathbf{v}_1, ..., \mathbf{v}_n\}$ be a basis of an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$. The *Hadamard ratio* of $\mathbf{B}$ is defined as

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

Definition 3.8 (Dual of a Lattice)**.** [9] The *dual* lattice of a lattice $\mathcal{L}$ is by definition

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

In other words, a dual of a lattice is a lattice such that the scalar product between any vector of the dual and any vector of the original matrix must be an integer. Given that lattice points are nothing but integral linear combinations of basis vectors, it is sufficient to ensure that the above holds for the basis vectors; if so, it necessarily holds for all other lattice vectors too. [9]

3.2 Rings

Definition 3.9 (Ring)**.** [12] A *ring* is a set R that has two operations, denoted by + and *, having the following properties:

- Properties of + :
  - Identity Law: There is an additive identity $0 \in R$ such that $0 + a = a + 0 = a$ for every $a \in R$.
  - Inverse Law: For every element $a \in R$ there is an additive inverse $b \in R$ such that $a + b = b + a = 0$.
  - Associative Law: $a + (b + c) = (a + b) + c$, for all $a, b, c \in R$
  - Commutative Law: $a + b = b + a$, for all $a, b \in R$
- Properties of * :
  - Identity Law: There is a multiplicative identity $1 \in R$ such that $1 * a = a * 1 = a$ for every $a \in R$.
  - Associative Law: $a * (b * c) = (a * b) * c$, for all $a, b, c \in R$
  - (Optional) Commutative Law: $a * b = b * a$, for all $a, b \in R$
- Property linking + and * :
  - Distributive Law: $a * (b + c) = a * b + a * c$, for all $a, b, c \in R$

If * has the optional property of possessing a Commutative Law, the ring is known as a *commutative ring*.

Definition 3.10 (Field)**.** [12] A commutative ring in which every nonzero element has a multiplicative inverse is called a *field*.

Consider the ring $R = \mathbb{Z}/n\mathbb{Z}$, known as a quotient ring. R is always a ring and if n is a prime, R is a field.

3.3 Polynomial Rings

If R is any ring, a ring of polynomials whose coefficients are taken from R can be formed and is denoted as $R[X]$.[12]

Theorem 3.3. [12] *Let $\mathbb{F}$ be a field and let $\mathbf{m} \in \mathbb{F}[X]$ be an irreducible polynomial. Then the quotient ring $\mathbb{F}[X]/\mathbf{m}$ is a field, i.e. every nonzero element of $\mathbb{F}[X]/\mathbf{m}$ has a multiplicative inverse.*

Theorem 3.4. [12] *Let $\mathbb{F}_p$ be a finite field and let $\mathbf{m} \in \mathbb{F}_p[X]$ be an irreducible polynomial of degree $d \geq 1$. Then $\mathbb{F}_p[X]/\mathbf{m}$ is a field with p^d elements.*

In further sections, we denote, following the lead of [3] and [6], by $\mathbb{Z}_q$ the quotient ring $\mathbb{Z}/q\mathbb{Z}$, with q a prime. We further define $R = \mathbb{Z}[X]/(X^n + 1)$. Combining those two, we define $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ as the ring of integer polynomials modulo $X^n + 1$, with each coefficient reduced modulo q.

3.4 Modules

A module is a mathematical structure that generalises the concept of a vector space. It is defined over a ring and a multiplication operation, which is compatible with the multiplication operation of the ring, is defined between elements of the module and elements of the ring. The operation between the ring and the module is called *scalar multiplication*, to differentiate from the operation between elements of the ring, called *ring multiplication*. This is analogous to the perhaps more familiar case of a vector space over a field, where the field's scalars act on the vectors by scalar multiplication. In a module, the scalars don't need to be a field, it is sufficient for them to be a ring. A vector space can hence be considered as a special case of a module in which the scalars are a field, which is a special case of a ring.

As seen before, a lattice corresponds to a vector space over a field of integers. Similarly, a *module lattice* is a lattice corresponding to a finitely generated module over a ring of integers.

3.5 The Number-Theoretic Transform

The number-theoretic transform (NTT) is a discrete Fourier transform generalized to finite fields. It is used to calculate convolutions of integer series, which in turn is a computational method that is employed to multiply polynomials. [18] Using NTT to multiply polynomials has been proved to be significantly faster than alternative methods, like Karatsuba multiplication.[6]

Let $\mathbb{F}$ be a finite field. In order to multiply two polynomials a and b, where $a, b \in \mathbb{F}$, the NTT is first applied individually to a and b, their convolution is then calculated (denoted by $\circ$), and the result is inverse transformed.

$$a * b = NTT^{-1}(NTT(a) \circ NTT(b))$$

4 Review of Related Work

We begin this section by introducing some of the most common computational problems underlying lattice-based cryptography. We shall then move on to give a brief overview of various notable entries in the early development of lattice-based cryptography, until we arrive at the Learning with Errors problem; as this is of particular relevance to the implementation aspect of this project, it will receive a more in-depth treatment. The section will conclude by establishing the difference between various notions of security relevant to PKEs and KEMs and how to use the Fujisaki-Okamoto transform to go from lower to higher security definitions. We stress at this point that the details of the security reductions of the schemes that are to be presented are beyond the scope of this project; we merely present their results to convey to the reader some appreciation of the hardness of the schemes.

4.1 Hard Lattice Problems

We use this section to define some of the computational lattice problems most frequently found at the root of the hardness of lattice-based cryptographic constructs.

Definition 4.1 (Shortest Vector Problem (search) (SVP))**.** [22] Given an arbitrary basis $\mathbf{B}$ of some lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$, find a shortest nonzero vector $\mathbf{v} \in \mathcal{L}$ for which $\|\mathbf{v}\| = \lambda_1(\mathcal{L})$.

Definition 4.2 (Approximate SVP (search) (SVP_γ))**.** [22] Given an arbitrary basis $\mathbf{B}$ of an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$, find a nonzero vector $\mathbf{v} \in \mathcal{L}$ for which $\|\mathbf{v}\| \leq \gamma(n) \cdot \lambda_1(\mathcal{L})$, where $\gamma(n) \geq 1$.

Note that by setting $\gamma(n) = 1$, the non-approximate SVP is recovered. SVP is hence the hardest instance of SVP_γ. SVP and SVP_γ, as described above, are known as *search* problems, because they require searching for a certain vector. There exist related *decision* problems, which confront the attacker, i.e. the person trying to solve them, with various options, among which he or she must choose the correct one.

Definition 4.3 (Shortest Vector Problem (decision) (SVP))**.** [23] Given a lattice basis $\mathbf{B}$ of some lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$ and a real $d > 0$, distinguish between the cases $\lambda_1(\mathcal{L}) \leq d$ and $\lambda_1(\mathcal{L}) > d$.

Definition 4.4 (Approximate SVP (decision) (GapSVP_γ))**.** [23] Given a lattice basis $\mathbf{B}$ of some lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$ and a real $d > 0$, distinguish between the cases $\lambda_1(\mathcal{L}) \leq d$ and $\lambda_1(\mathcal{L}) > \gamma \cdot d$.

A further similar problem is known as the *shortest independent vector problem*; it comes, once again, in an exact and an approximate version. Note that, similarly to SVP, setting γ equal to 1 in the approximate version gives the exact one. It can be formulated as either a search or a decision problem; we limit ourselves to providing the definition for the approximate search version.

Definition 4.5 (Approximate Shortest Independent Vector Problem (SIVP$_\gamma$))**.** [26] Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$, find a set of n linearly independent vectors in $\mathcal{L}(\mathbf{B})$ each of length at most $\gamma\lambda_n(\mathcal{L}(\mathbf{B}))$.

Definition 4.6 (Closest Vector Problem (CVP))**.** [12] Given a vector $\mathbf{w} \in \mathbb{R}^n$ that is not in $\mathcal{L}$, find a vector $\mathbf{v} \in \mathcal{L}$ that is closest to $\mathbf{w}$, i.e., find a vector $\mathbf{v} \in \mathcal{L}$ that minimizes the Euclidean norm $\|\mathbf{w} - \mathbf{v}\|$.

Definition 4.7 (Approximate CVP (CVP$_\gamma$))**.** [24] For $\lambda(n) \geq 1$ and given a basis $\mathbf{B}$ of a lattice $\mathcal{L} = \mathcal{L}(\mathbf{B}) \subset \mathbb{R}^n$ and a point $\mathbf{t} \in \mathbb{R}^n$ find some $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{t} - \mathbf{v}\| \leq \gamma(n) \cdot dist(\mathbf{t}, \mathcal{L})$, where $\gamma(n) \geq 1$.

Note that $dist(\mathbf{t}, \mathcal{L})$ represents the distance from $\mathbf{t}$ to the nearest lattice point.

While the fully general versions of SVP and CVP are conjectured to be extremely hard problems, this level of generality is challenging to attain in practice. Realistic implementations of cryptosystems generally rely on a specific subclass of these problems: this may be in the interest of efficiency or else may serve to allow the introduction of a trapdoor. [12] Note also that there may be more than one shortest nonzero vector in a lattice, which is why SVP and CVP ask for "a" rather than "the" shortest vector.

There is a known algorithm, called *Babai's Closest Vertex Algorithm* that solves CVP given a basis whose vectors are sufficiently orthogonal. Note that $\lfloor x \rceil$ denotes rounding of x, with ties broken upwards.

Theorem 4.1 (Babai's Closest Vertex Algorithm)**.** [12] *Let $\mathcal{L} \subset \mathbb{R}^n$ be a lattice with basis $\mathbf{v}_1, ..., \mathbf{v}_n$, and let $\mathbf{w} \in \mathbb{R}^n$ be an arbitrary vector. If the vectors in the basis are sufficiently orthogonal to one another, then the following algorithm solves* CVP*.*

1. *Write $\mathbf{w} = t_1\mathbf{v}_1 + t_2\mathbf{v}_2 + ... + t_n\mathbf{v}_n$, with $t_1, ..., t_n \in \mathbb{R}$.*
2. *Set $a_i = \lfloor t_i \rceil$ for $i = 1, 2, ..., n$.*
3. *Return the vector $\mathbf{v} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + ... + a_n\mathbf{v}_n$.*

In general, if the basis vectors are reasonably orthogonal to one another, then the algorithm solves some version of CVP$_\gamma$, but if the basis vectors are highly non-orthogonal, then the vector returned by the algorithm is generally far from the lattice vector that is closest to $\mathbf{w}$.

There are further algorithms that are known to be efficient against SVP and CVP. These algorithms are known as *lattice reduction algorithms* and include the LLL algorithm, by Lenstra, Lenstra and Lovász, which solves SVP and CVP to within a factor c^n, where c is a constant factor and n is the lattice dimension. [12] However, they go beyond the scope of this work and are mentioned here only for the sake of completeness.

A further problem akin to CVP with the difference that the target point $t \in \mathbb{R}^n$ is promised to be "rather close" to the lattice, is the Bounded Distance Decoding problem. [22]

Definition 4.8 (Bounded Distance Decoding Problem (BDD_γ))**.** [22] Given a basis $\mathbf{B}$ of an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$ and a target point $\mathbf{t} \in \mathbb{R}^n$ with the guarantee that $dist(\mathbf{t}, \mathcal{L}) < d = \lambda_1(\mathcal{L})/(2\gamma(n))$, find the unique lattice vector $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{t} - \mathbf{v}\| < d$.

4.2 The Origins of Lattice-Based Cryptography

The first cryptographic system whose security was based on the hardness of lattice problems was introduced in 1996, by Miklós Ajtai. He introduced the problem known as Short Integer Solution (SIS) and an associated family of one-way functions. [1]

Definition 4.9 (Short Integer Solution ($\mathrm{SIS}_{n,q,\beta,m}$))**.** [22] Given m uniformly random vectors $\mathbf{a}_i \in \mathbb{Z}_q^n$, forming the columns of a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, find a nonzero integer vector $\mathbf{z} \in \mathbb{Z}^m$ of norm $\|\mathbf{z}\| \leq \beta$ such that

$$f_{\mathbf{A}}(\mathbf{z}) := \mathbf{A}\mathbf{z} = \sum_i \mathbf{a}_i \cdot z_i = 0 \in \mathbb{Z}_q^n.$$

The problem is parametrised by four values, n, q, β and m; n and q define the group $\mathbb{Z}_q^n$ over which the vectors are chosen, m is the number of group elements and β is an upper bound on the length of the solution vector. In essence, the SIS problem requires a non-trivial integer combination of m uniformly random elements of $\mathbb{Z}_q^n$ that sums to zero and whose Euclidean norm does not exceed β. [22]

Ajtai presented a worst-case to average-case reduction by showing that solving SIS was on average at least as hard as solving the approximate shortest vector problem is in the worst case. [22] This reduction was the key to making the scheme practically useful, since worst-case complexity is not sufficient for cryptographic applications. Worst-case complexity means that there are instances of the problem, with specifically tuned parameters, that are hard to solve, yet the average case, with random parameters, may be easy to solve. Average-case complexity, on the other hand, implies that the average case, with random parameter values, is hard to solve, thus making the scheme suitable for cryptographic use.

Also in 1996, Hoffstein, Pipher and Silverman presented a public key encryption scheme called NTRU (or NTRUEncrypt), although it was not published until 1998 [22]. NTRU is generally defined using polynomial rings and their quotients; however, it can be described in terms of algebraically structured lattices. The problem from which it draws its intractability can be represented as SVP or CVP in a special class of lattices; the SVP version targets key recovery, whereas the CVP version aims to recover the plaintext. [12]

Following the earlier work by Ajtai, in 1997, Ajtai and Dwork put forward a lattice-based public-key encryption scheme whose security was based on the worst-case hardness of a variation of the SVP, known as the *unique shortest vector problem* uSVP$_\gamma$. Being the first lattice-based encryption scheme with provable security under a worst-case assumption, this was a major achievement. However, the scheme suffered from excessively large key and ciphertext sizes, resulting in time-consuming encryption and decryption operations, which reduced its practical applicability. In 1998, Nguyen and Stern demonstrated that the Ajtai-Dwork scheme cannot have any secure implementations that are efficient and practical. [22]

1997 also saw the introduction of the GGH lattice-based cryptosystem, named after its creators, Goldreich, Goldwasser and Halevi. Its security is based on CVP. In this scheme, the private key is a "good" basis for a lattice $\mathcal{L}$, while the public key is a "bad" basis; the message $\mathbf{m}$ is a binary vector (i.e. a vector whose coefficients can only take the values 0 or 1) and the ciphertext is the linear combination of the coefficients of $\mathbf{m}$ with the vectors $\mathbf{v}_i$ comprising the bad basis. This results in a new lattice vector, $\mathbf{v}'$. In order to obtain the final ciphertext, a small random vector $\mathbf{r}$ is added to $\mathbf{v}'$, giving a vector $\mathbf{w}$ that is close to being a lattice vector, but is not quite one. [12]

$$\mathbf{w} = \mathbf{v}' + \mathbf{r} = \sum_i m_i \mathbf{v}_i + \mathbf{r}$$

The intended recipient, knowing the "good" basis, can use Babai's algorithm, given in theorem 4.1, to solve the CVP and thus recover $\mathbf{v}$ from $\mathbf{w}$; they can then express it in terms of the bad basis to recover the message $\mathbf{m}$. A potential eavesdropper, however, knowing only the "bad" basis, cannot solve the CVP.

The next major step for lattice cryptography came in 2005, with Regev's introduction of the *Learning with Errors* problem. [27] This is of fundamental importance to the implementation aspect of this project, hence, while we only gave brief introductions to some of the previous milestones of lattice-based cryptography, we will investigate the Learning with Errors problem in more detail.

4.3 The Learning with Errors Problem Family

4.3.1 Learning With Errors

In 2005, Regev introduced the Learning with Errors (LWE) problem and proved its *quantum* reduction to hard lattice problems; this implies that solving LWE requires the existence of an efficient *quantum* algorithm for the solution of hard lattice problems like GapSVP$_\gamma$, even for small q, and search SIVP. [27] As there currently exist no known quantum algorithms for the solution of either of these problems that are siginificantly faster than classical algorithms, this is a meaningful indicator of the hardness of LWE. [22] As a security guarantee however, it remains less compelling than a comparable *classical* reduction. A

first step in this direction was achieved by Peikert, who, in 2009, provided the first classical reduction, albeit only reducing LWE to GapSVP_γ with large q, a problem less hard than the ones involved in Regev's quantum reduction. [25] In 2013, a classical reduction from n-dimensional LWE, with modulus q of polynomial order in n, to worst-case lattice problems in dimension $\sqrt{n}$ was achieved by Brakerski et al., firmly cementing the security of all applications of LWE with polynomial modulus. [7]

There exist, as for some of the lattice problems described earlier, two main versions of LWE, a search problem and a decision problem. Below we give definitions of both, but before that, we define the LWE distribution; this is parametrised by the dimension n and the modulus q, which, much like was the case for SIS, define the group over which vectors are chosen; a further parameter is χ, an error distribution over $\mathbb{Z}$. Generally speaking, χ is taken to be a discrete Gaussian distribution, parametrised by its width αq, with $\alpha < 1$. [22]

Definition 4.10 (LWE distribution)**.** [22] For a vector $\mathbf{s} \in \mathbb{Z}_q^n$ called the *secret*, the LWE distribution $A_{\mathbf{s},\chi}$ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is sampled by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \leftarrow \chi$, and outputting

$$(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e \bmod q).$$

Note that $\langle \mathbf{a}, \mathbf{b} \rangle$ denotes the inner product of two vectors $\mathbf{a}$ and $\mathbf{b}$; $e \leftarrow \chi$ denotes that e is sampled according to the distribution χ.

In addition to the parameters n, q and χ, the LWE problems are further parametrised by the number m of samples from the distribution $A_{\mathbf{s},\chi}$ that the attacker has at their disposal; m is however a circumstantial parameter and is therefore not always specified.

Definition 4.11 ($\text{LWE}_{n,q,\chi,m}$ (search))**.** [22] Given m independent samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ drawn from $A_{\mathbf{s},\chi}$ for a uniformly random $\mathbf{s} \in \mathbb{Z}_q^n$ (fixed for all samples), find $\mathbf{s}$.

Definition 4.12 ($\text{LWE}_{n,q,\chi,m}$ (decision))**.** [22] Given m independent samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where every sample is distributed according to either: (1) $A_{\mathbf{s},\chi}$ for a uniformly random $\mathbf{s} \in \mathbb{Z}_q^n$ (fixed for all samples), or (2) the uniform distribution, distinguish which is the case (with non-negligible advantage).

While it may be obvious that the ability to solve search-LWE implies the ability to solve decision-LWE, it has been proved for LWE that the two versions are in fact equivalent; hence it is equally true that solving decision-LWE means solving search-LWE. [3] This is a somewhat curious property of LWE that does by no means hold for all cryptographic problems. The designation as a *learning* problem refers to the fact that the attacker's task it to *learn* a secret given a set of noisy equations for that secret.

LWE can be considered as a generalisation of the *Learning Parity with Noise* problem to higher moduli. [27] Equivalently, the Learning Parity with Noise

problem is a special instance of LWE, with modulus $q = 2$ and $A_{\mathbf{s},\chi}$ equal to the Bernoulli distribution over $\{0, 1\}$. [22]

In the same work that introduced LWE, Regev also presented a cryptosystem based on it: this has two major advantages compared to the work of Ajtai-Dwork; firstly, public key sizes are of the order of n^2, rather than n^4, and ciphertext sizes go as $\mathcal{O}(n)$ as opposed to $\mathcal{O}(n^2)$. [27] Note that logarithmic factors are disregarded in this statement. Secondly, the hard lattice problems on which Regev's system is based seem to present less mathematical structure than those underlying the Ajtai-Dwork system; since adversaries might be able to devise attack schemes that exploit any underlying mathematical structure, this increased generality benefits the system's security. [22]

Since its inception, LWE has proved to be a highly flexible problem with a broad application spectrum in cryptography: it has been used in the construction of both IND-CPA and IND-CCA secure public key encryption schemes, has been employed in leakage-resilient cryptography and has been successfully applied to fully homomorphic encryption, among others. [7] Due to its significant potential, LWE has enjoyed considerable attention from the cryptographic community, and continues to do so. Effort is being expended on the task of gaining a better understanding of its security implications and on devising increasingly efficient constructs based, in general, on specific variations of the general LWE problem. One such adaptation of LWE goes by the name of *Learning with Rounding* (LWR), where the sampling of noise from an error distribution is replaced by deterministic rounding of the product $\mathbf{a}\cdot\mathbf{s}$. Variations like this one are generally designed to speed up the computational processes while at the same time minimising key and ciphertext sizes; however, such advantages may come at the price of increased structure in the underlying mathematical constructs and hence potentially weaker security. The next subsection is devoted to the introduction of two such variations of LWE.

4.3.2 Ring-LWE and Module-LWE

Two of the currently most popular variations of LWE are the *Ring-LWE* and the *Module-LWE* problems. While both of these schemes fall under the category of LWE variants that rely on constructs sporting increased structure compared to general LWE, to date, no attacks have been found that exploit this potential caveat. Ring-LWE (RLWE) was introduced in 2010 by Lyubashevsky, Peikert and Regev [15]; Module-LWE (MLWE) was introduced two years later, in 2012, by Langlois and Stehlé. [14]

Similar to LWE, RLWE is parametrised by three principal parameters, n, q and χ. Together, n and q define the quotient ring R_q; the ring R is a ring over $\mathbb{Z}$ with degree n, generally a cyclotomic ring, and R_q is defined as R/qR. [22] As it was in the LWE case, χ is the error distribution, generally a discrete Gaussian.

Definition 4.13 (RLWE distribution)**.** [22] For an $s \in R_q$ called the *secret*, the Ring-LWE distribution $A_{s,\chi}$ over $R_q \times R_q$ is sampled by choosing $a \in R_q$ uniformly at random, choosing $e \leftarrow \chi$, and outputting

$$(a, b = a \cdot s + e \bmod q).$$

The original definition of the distribution requires considerably more complex mathematical concepts; however, it has been shown that the definition given here is largely equivalent. It can be seen that the definitions of the LWE and the RLWE distributions differ in the key point that $\mathbf{s}$ and $\mathbf{a}$ are vectors in $\mathbb{Z}_q^n$ for LWE, but a and s are polynomials in R_q for RLWE. [22]

We now give the definition of the decision version of the RLWE problem, which is the one generally used in cryptographic applications; note that we once again make use of m to denote the number of samples the attacker has at their disposal.

Definition 4.14 (Ring-LWE$_{q,\chi,m}$ (decision))**.** [22] Given m independent samples $(a_i, b_i) \in R_q \times R_q$ where every sample is distributed according to either: (1) $A_{s,\chi}$ for a uniformly random $s \in R_q$ (fixed for all samples), or (2) the uniform distribution, distinguish which is the case (with non-negligible advantage).

The main advantages of cryptographic constructs built on RLWE rather than LWE are key- and ciphertext sizes as well as efficiency. One sample from $A_{s,\chi}$ yields one scalar $b_i \in \mathbb{Z}$ in LWE; in RLWE, it yields an n-dimensional element $b \in R_q$. In light of this, RLWE can be interpreted as LWE with correlated samples; this is what is referred to as the additional mathematical structure inherent to RLWE. [22] Not only does this vastly reduce the number of computations required for RLWE, but in addition, polynomial multiplications over certain rings can be performed very efficiently using NTT, which significantly speeds up the remaining computations.

MLWE adheres to the same general form, where samples of the distribution are given as $(a, b = a \cdot s + e)$. In LWE, $\mathbf{a}$ and $\mathbf{s}$ are vectors in $\mathbb{Z}_q^n$, in RLWE, a and s are polynomials in R_q and now in MLWE, $\mathbf{a}$ and $\mathbf{s}$ are modules in $(R_q)^k$, where k is called the *rank* of the module. This means that the difference between RLWE and MLWE is that the single ring elements (a and s) have now been replaced by modules over the same ring. As such, RLWE can be viewed as a special instance of MLWE, i.e. MLWE with rank $k = 1$.[2]

Definition 4.15 (MLWE distribution)**.** [6] For an $\mathbf{s} \in (R_q)^k$ called the *secret*, the Module-LWE distribution $A_{\mathbf{s},\chi}$ over $(R_q)^k \times R_q$ is sampled by choosing $\mathbf{a} \in (R_q)^k$ uniformly at random, choosing $e \leftarrow \chi$, and outputting

$$(\mathbf{a}, b = \mathbf{a}^T \cdot \mathbf{s} + e).$$

Note that $\mathbf{a}^T$ denotes the transpose of $\mathbf{a}$.

Definition 4.16 (Module-LWE$_{q,k,\chi,m}$ (decision))**.** [6] Given m independent samples $(\mathbf{a}_i, b_i) \in (R_q)^k \times R_q$ where every sample is distributed according to either: (1) $A_{\mathbf{s},\chi}$ for a uniformly random $\mathbf{s} \in (R_q)^k$ (fixed for all samples), or (2) the uniform distribution, distinguish which is the case (with non-negligible advantage).

MLWE shines when it comes to fine-tuning the trade-off between efficiency and security. Since rings whose dimension n is a power of two allow such efficient implementation using the number-theoretic transform, imagine that the ring R_q for an MLWE scheme is taken to be of dimension 256. This allows straightforward changing of the effective dimension, $n \cdot k$, by varying the rank while keeping the dimension at its highly efficient value. The scheme could therefore be used with an effective dimension of 256, 512, 768, etc. Those values that are not powers of two, like 768, cannot be attained with an RLWE scheme, making MLWE the better option when it comes to tailoring a cryptographic system to a specific security level in between the power-of-two levels. [2]

As k is generally taken to be quite a small value, like 3, the computational overhead of MLWE compared to RLWE is not huge: MLWE requires k multiplications of polynomials of dimension n, whereas RLWE requires only 1 multiplication of polynomials of dimension n', where, for comparable security, $n' = n \cdot k$. This decrease in efficiency is balanced by the decrease in structure. As an MLWE sample can be viewed as k distinct polynomials each of size n, only the n coefficients of each polynomial are correlated, whilst being independent of the coefficients of the other polynomials; recall that in RLWE, all coefficients of the single poylnomial are correlated. Figure 5 recapitulates the key differences between LWE, RLWE and MLWE by showing the (a, b) tuples returned by the distributions of the three problems.

Both RLWE and MLWE enjoy strong security guarantees. To date, no attacks, quantum or classical, have been devised that take advantage of the added structure of these algorithms compared to regular LWE. Because of this, when it comes to assessing their security, they are generally treated as regular LWE. The fact remains, however, that the added mathematical structure could lead to future vulnerabilities. [3] [6]

4.4 Security Definitions

Security notions of cryptographic systems are customarily defined using *games* played between a *challenger* and an *attacker* or *adversary*. The system is considered secure if the attacker, a *probabilistic polynomial time Turing machine*, after performing a polynomial number of computations, does not have a chance of winning that is non-negligibly higher than if they were guessing. In this section, we will give the defining games for two different notions of security for public-key encryption schemes: *Indistinguishability under Chosen Plaintext Attack* (IND-CPA), also known as *semantic security*, and *Indistinguishability under Chosen Ciphertext Attack* (IND-CCA). IND-CCA comes in two flavours,

non-adaptive, IND-CCA1, and *adaptive*, IND-CCA2, which is the stronger of the two definitions. Similar security definitions apply for KEMs.

$KG(\lambda)$ denotes key generation under security parameter λ (generally related to the length of the key), $Enc(pk, m)$ denotes encryption of a message m using public key pk and $Dec(sk, c)$ denotes decryption of a ciphertext c using secret key sk.

Definition 4.17 (IND-CPA)**.** The IND-CPA game is played between the challenger and an attacker.

- The challenger runs $(pk, sk) \leftarrow KG(\lambda)$ and passes pk to the attacker.
- The attacker performs a polynomial number of computations.
- The attacker submits two messages, m_0 and m_1, of equal length to the attacker.
- The challenger selects a bit $b \in \{0, 1\}$ at random.
- The challenger returns $c_b = Enc(pk, m_b)$ to the attacker.
- The attacker performs a polynomial number of computations.
- The attacker outputs a bit b'.

The attacker wins the game if $b = b'$. Let $Pr[b = b']$ be the probability that the attacker wins the IND-CPA game, taken over all randomness involved in the game. A PKE scheme satisfies indistinguishability under chosen plaintext attack if

$$\left|Pr[b = b'] - 1/2\right|$$

is negligible as a function of λ.

Definition 4.18 (IND-CCA)**.** The IND-CCA game is played between the challenger and an attacker.

- The challenger runs $(pk, sk) \leftarrow KG(\lambda)$ and passes pk to the attacker.
- The attacker is given access to a decryption oracle that returns $Dec(sk, c)$ on input a ciphertext c, for a polynomial number of inputs.
- The attacker submits two messages, m_0 and m_1, of equal length to the attacker.
- The challenger selects a bit $b \in \{0, 1\}$ at random.
- The challenger returns $c_b = Enc(pk, m_b)$ to the attacker.
- The attacker gets help from the decryption oracle on a polynomial number of inputs $c \neq c_b$.
- The attacker outputs a bit b'.

The attacker wins the game if $b = b'$. Let $Pr[b = b']$ be the probability that the attacker wins the IND-CCA game, taken over all randomness involved in the game. A PKE scheme satisfies indistinguishability under chosen ciphertext attack if

$$\left|Pr[b = b'] - 1/2\right|$$

is negligible as a function of λ.

The difference between IND-CCA1 and IND-CCA2 lies in the phase after the challenger returns a ciphertext to the attacker: in the non-adaptive case, i.e. for IND-CCA1 security, the attacker cannot make any further use of the decryption oracle. In the adaptive case, i.e. for IND-CCA2, the attacker may make a polynomial number of further calls to the oracle, with the limitation that they may not ask for the encryption of either of the messages submitted to the challenger.

A further notion of security that is frequently used to quantify the strength of PKEs are NIST-defined security levels. The levels range from 1 to 5 and are specified in terms of the difficulty of breaking various versions of AES. For instance, a PKE with security level 1 is *at least* as hard to brute-force as AES-128. Level 3 corresponds to AES-192 and level 5, the highest defined level, corresponds to brute-forcing AES-256.

4.5 The Fujisaki-Okamoto Transform

The Fujisaki-Okamoto transform, introduced in 1999 by Fujisaki and Okamoto, allows the conversion of a weakly secure asymmetric encryption scheme to a strongly secure one (IND-CCA2); it further shows how to create strongly secure hybrid encryption schemes from a weak symmetric and a weak asymmetric scheme. [11] Many others have followed up on this work in the years after, including Hofheinz, Hövelmanns and Kiltz, who, in 2017, showed how to use an extension of the Fujisaki-Okamoto transform to create an IND-CCA2 secure KEM from a PKE with a less strong security definition. [13] This is achieved through the use of cryptographic *hash functions*; these functions are strictly one-way, efficiently computable and, very importantly, collision resistant. In order to transform an IND-CPA PKE into an IND-CCA KEM, three hash functions are required: one that maps from the scheme's message space to its randomness space, one that maps from a bitstream (a sequence of bits of undefined length) to a bit string of length n, and a final one that maps from a bitstring of length n to another bitstring of length n. [13] Among other things, these hash functions are used to hash the random coins from which some of the values used in the encryption cycle are derived, in order to protect against attacks that involve the disclosure of system randomness. [3]

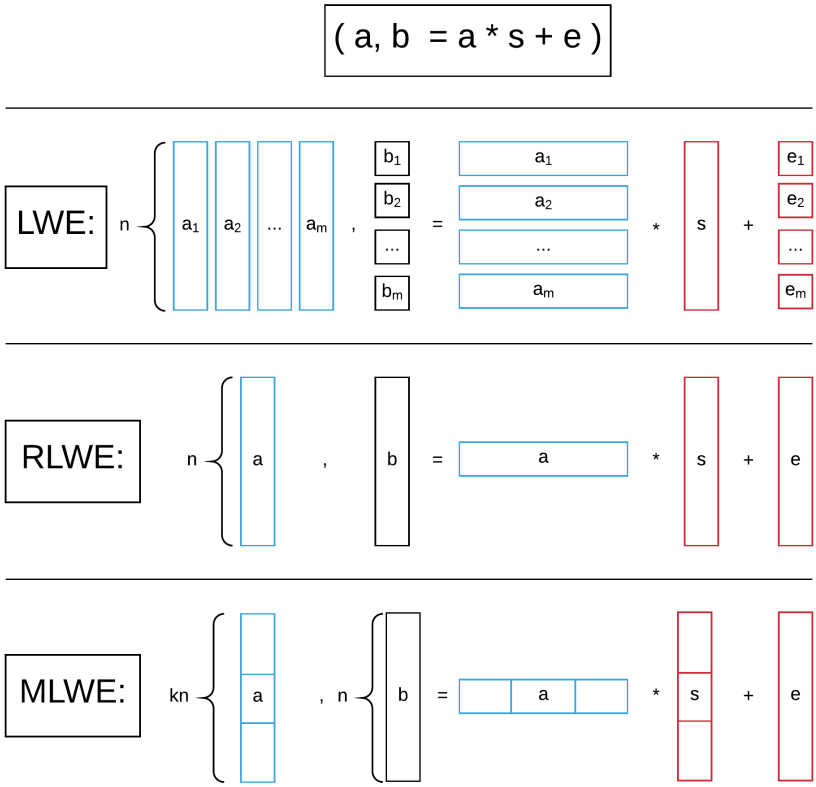

Figure 5: Diagrammatic representation of the samples output by the distributions of the three members of the LWE family presented in this work. In the LWE case, the $\mathbf{a}_i$ and $\mathbf{s}$ are vectors in $\mathbb{Z}_q^n$, the b_i and e_i are values in $\mathbb{Z}_q$. In the RLWE case, a and s are polynomials over R_q, as are b and e. In the MLWE case, $\mathbf{a}$ and $\mathbf{s}$ are modules over $(R_q)^k$, b and e are polynomials over R_q.

5 Key Encapsulation Tool

5.1 Introduction

The second part of this project was devoted to the creation of a practical and easily accessible tool for key encapsulation. The tool is based on implementations of the New Hope [3] and CRYSTALS Kyber [6] (henceforth Kyber) key encapsulation schemes, to which the user has access via a graphical user interface (GUI). The user is given the choice between three key encapsulation systems: an IND-CPA secure version of New Hope, and IND-CCA secure version of New Hope and an IND-CCA secure version of Kyber. For the first two, the user may choose a NIST security level of either 1 or 5, for the latter, the user may choose from among 1, 3 and 5. For each of the three systems, the three required functions for any KEM, key generation, encapsulation and decapsulation, can be run.

All the coding involved in the creation of this tool was completed in Python 3.7, using the PyCharm editor by JetBrains. The New Hope and Kyber implementations were written from scratch, following the pseudocode in [3] and [6], as well as the reference implementations submitted to the NIST call for proposals for the standardisation of post-quantum cryptography. The GUI was equally built from scratch. The only code library that was used that is not part of the Python standard library on Windows 10 systems is the library for the number-theoretic transform, credit for which goes to Project Nayuki [19]; explicit permission to use this library for academic purposes was obtained from the library author. For a list of cryptographic libraries used and their purpose, see section 5.7.

To the best of our knowledge, no tool comparable to the one created during this project is available at the time of writing. While two Python implementations of New Hope are available, at [5] and [20], no use was made of these during the inception or the creation of the implementation presented in this work. To the best of our knowledge, no Python implementations of Kyber are available.

5.2 Inception Process

The original project idea was to divide the project into a first phase, devoted to a comparison of the various KEMs submitted to the NIST call for proposals, and a second phase, reserved for the implementation of a single one of those KEMs into a key encapsulation tool. The first phase would hence serve as a preparatory step for the second one, as its ultimate goal was to decide upon one KEM that was objectively the most well-suited to the task. This rather naive idea was however quickly crushed when it became clear, upon taking a first few steps into the world of post-quantum cryptography, that comparing the various systems on any meaningful level was no mean feat. After all, even NIST allowed for a period of up to five years to conduct its evaluation of the submissions. When it thus turned out that lattice-based cryptography was a highly complex field

at the intersection of advanced mathematics and cutting-edge cryptography, for which the author's very limited training in classical cryptography was no match, the focus of the project phases shifted.

It was decided that the evaluation of the NIST submissions, which was still required to choose the system or systems that were to form the basis of the encapsulation tool, would be less in-depth and based largely on the papers that accompanied the submissions, as well as other cryptographic publications. While still a major task, this did allow for phase one to also incorporate a heavy element of edification in matters of general post-quantum cryptography, as well as lattice-based cryptography and its underlying mathematics in particular. It was during this stage that it became clear that, in order to increase the tool's future-proofness and longevity, as well as its flexibility, it would be preferable to base it on two distinct cryptosystems, rather than just a single one. Furthermore, it would be good if those systems were based on different hard lattice problems, so that, in the event that one of the systems should be broken, the tool could continue to be used.

The choice of Python as the programming language for this project was not arbitrary. Initially, as the degree in the context of which this project is being conducted focused a large amount of effort on teaching Java, this was assumed to be the language in which the implementation would be written. However, it took little effort to find out that this was not an ideal choice. Partially, this is due to the limitations of Java in terms of libraries for vector and matrix operations; however, it is also due to the fact that it would simply take a great many lines of Java code to perform the same mathematical operations that could be done quite concisely in a language like Python. The C language was excluded from consideration, as, first of all, learning C from scratch within the time frame of this project, in addition to all the other project work, seemed daunting; secondly, all NIST submissions came with obligatory C implementations, making the task of re-implementing them quite useless. The choice was narrowed down to either a mathematical software tool like SageMath, or the latest version of Python, a language renowned for its prowess in the domain of mathematical computation. In addition to this, we had some familiarity with Python from past experience, making the prospect of learning it properly less intimidating. A third option would have been to use an older version of Python (2.x), which allows the usage of Sage as a library. Python was chosen over SageMath as it would allow the tool to be easily and widely used, without need for specialised software. Python 3.7, the latest version, was chosen over 2.x to make the implementation future-proof. To summarise, Python was chosen to provide a relatively pain-free implementation process, allow the tool to be widely and conveniently used and overall, make the project more "realistic", i.e. employ industry-standard tools as opposed to purely academic ones.

5.3 The Cryptosystems

The NIST call for proposals received 70 submissions aiming to become the new standards for public-key cryptographic applications like digital signature schemes, encryption schemes, KEMs etc. Out of those, and after some of them had been retracted by the submitters, 26 remained that were based on lattice problems. Of those 26, 5 are digital signature schemes, 1 is a key exchange protocal, 1 is a public-key encryption scheme and 19 are KEMs. The KEM submissions are mostly based on various versions of the LWE and LWR problems, with a few based on the NTRU scheme.

In order to ensure that the tool would be responsive and fast to use, schemes based on general LWE/LWR, rather than their more structured variants, were excluded at an early stage of the selection process. It seemed desirable to implement two schemes based on different variants of lattice problems to assure that, should one of the systems be broken due to its mathematical structure, the tool would continue to be useful. However, it was soon decided that it would be beneficial to either implement two schemes based on LWE variants, or else two schemes based on LWR variants; this would serve the purpose of making the implementations easier, as they would be based on the same underlying logic, with only the mathematical structures and the details of the support functions differing from one another. As the general versions of LWE and LWR are unlikely to be broken soon, this decision did not seem to introduce any untoward security risks. Alternatively, it may be argued that, if LWE or LWR can be broken, it might be reasonably expected that all of the submissions considered here would be broken too. Furthermore, it seemed prudent not to implement two different versions, based on different lattice problems, of the same submitted scheme, as these would share support methods. This reasoning was again based on the desire to avoid the possibility of the entire tool being crippled by the discovery of a single weakness.

Consideration was furthermore given to the standing of the various systems in the cryptographic community. Some schemes had been tested, scrutinised and discussed much more exhaustively than others, which served as an assurance of their quality. Finally, the clarity and quality of each submission's specification paper, and the expected difficulty of basing a successful implementation on it, were taken into account. It is based on these criteria that, after careful consideration, the New Hope and Kyber schemes were chosen to serve as the basis of the key encapsulation tool.

5.3.1 New Hope

New Hope is an RLWE based key encapsulation mechanism first introduced in 2015 by Alkim, Ducas, Pöppelmann and Schwabe. [4] The scheme has since then been extensively tested and was even chosen by Google to star in an experiment designed to determine the suitability of lattice based encryption schemes in Transport Layer Security (TLS), a protocol used in Internet communica-

tions. The experiment was a success and Google concluded that New Hope was well-suited for real-world deployment in Internet protocols. The New Hope submission contains two KEMs with different security notions, New Hope IND-CPA and New Hope IND-CCA, both of which are implemented in this tool. [28]

Like many KEMs, the two New Hope KEMs are based on an IND-CPA secure PKE, dubbed NewHope-CPA-PKE; this PKE is not intended to be used as a free-standing PKE since it does not accept variable length messages, but only messages that are exactly 256-bits (32 bytes) long. [3] The New Hope KEMs are parametrised by the values n, q, k, and γ: n is the polynomial dimension, q is the modulus, k is the width of the centered binomial distribution that serves as the noise distribution, and γ is the square root of an n-th root of unity reduced modulo q, a value used in the NTT. The two different parameter sets for each KEM that give different NIST security levels, differ only in their value of n; the other parameters are identical.

Part of the efficiency of New Hope comes from the use of NTT for polynomial multiplication. In order to maximise this efficiency, the decision was made to include NTT in the specification of the scheme and even transmit some values in NTT domain. [3] This reduces the flexibility of implementers: even if they choose to use a different multiplication algorithm, like Karatsuba, they will still be required to transform the result into NTT domain, thus making NTT a de-facto obligation.

New Hope makes use of two hash functions of the Keccak family (standardised as SHA3) for all it's hashing needs: SHAKE-128 and SHAKE-256, two extendable output functions, i.e. hash functions whose output length can be specified. The three hash functions that are needed in the Fujisaki-Okamoto transform to turn an IND-CPA PKE into an IND-CCA KEM are all taken to be SHAKE-256.

In order to reduce the size of the ciphertext, NewHope-CPA-PKE uses a compression function that performs so-called "bit-dropping": by performing a modulus-switch from modulus q to modulus 8, the number of bits used to represent each coefficient is reduced. This means, however, that a lot of information is lost in that operation, which is why decompression, which performs modulus switching from modulus 8 to modulus q, cannot be an exact inverse of compression. However, it is sufficient for it to be an approximate inverse, as the information that is lost during compression, the "low bits", are not essential to the success of the decryption operation.

The nature of the scheme, which is based on the recovery, from noisy equations, of a secret value, implies a possibility that even with the correct private key, decryption in the PKE, and hence decapsulation in the KEM, may not be successful: the only way to reduce this possibility not just to a negligible value, but to exactly zero, is to significantly lower the security by reducing the

amount of noise. Most lattice-based PKEs hence suffer from a decryption error probability, which in the case of New Hope, is reassuringly small: for NewHope-CPA-PKE with $n = 512$, it lies at 2^{-213}, with $n = 1024$, it lies at 2^{-216}. These values are so small that the possibility of decryption failure was ignored in the design of this tool. Tables 1 and 2 give a summary of the parameters and object sizes of New Hope and offer a direct comparison with the corresponding values for Kyber.

5.3.2 Kyber

The CRYSTALS Kyber scheme [6] is one element of the CRYSTALS (CRYptographic SuiTe for Agebraic LatticeS) cryptographic suite that, in addition to Kyber, an IND-CCA KEM, offers Dilithium, a digital signature algorithm. A subgroup of the creators of Kyber are also a subgroup of the team that created New Hope; Kyber is hence seen as the natural successor to New Hope. Unlike New Hope, it is based on MLWE, not RLWE, implying that it uses mathematical constructs that present less structure than New Hope's. Similarly to New Hope, and many other KEMs, the Kyber KEM is based on an IND-CPA secure PKE, Kyber.CPAPKE.

It is worth noting that Kyber does not use MLWE exactly as defined in section 4.3.2; that definition states that the samples of the MLWE distribution are of the form

$$(\mathbf{a}, b = \mathbf{a}^T \cdot \mathbf{s} + e),$$

where $\mathbf{a}$ and $\mathbf{s}$ are modules over $(R_q)^k$, while b and e are polynomials in R_q. In the variant used by Kyber, the samples are of the form

$$(\mathbf{A}, \mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}),$$

where $\mathbf{A}$ is a matrix over $(R_q)^{k \times k}$, with a small rank, like 3, while $\mathbf{b}$, $\mathbf{s}$ and $\mathbf{e}$ are modules over $(R_q)^k$. The rest of the discussion given in 4.3.2 is still applicable.

Kyber comes in three security strengths, NIST levels 1, 3 and 5. The corresponding schemes have been dubbed Kyber512, Kyber768 and Kyber1024, in reference to the effective dimension $k \cdot n$ of the underlying modules. Kyber instances are defined by 7 parameters: n, k, q, η, d_u, d_v and d_t. n is the dimension of the polynomials, k is the rank of the modules, q is the modulus, η is the logarithm in base 2 of the width of the binomial error distribution, and d_u, d_v and d_t are used with the encoding and compression functions inside Kyber.CPAPKE. These last three parameters are the same for all three security levels, as are n and q. The only values that change are k and η: for Kyber512, $k = 2, \eta = 5$, for Kyber768, $k = 3, \eta = 4$ and for Kyber1024 $k = 4, \eta = 3$.

Just like New Hope, Kyber too suffers from a decryption failure probability. While it is not quite as small as in the case of New Hope, it is nonetheless still small enough to be negligible and was ignored in the creation of the tool

described in this report. The decryption/decapsulation failure for Kyber512 is 2^{-145}, for Kyber768 it is 2^{-142} and for Kyber1024 it is 2^{-169}.

Again similarly to New Hope, the specification of Kyber includes NTT, for the same reasons: fast in-place polynomial multiplication. Kyber also performs bit-dropping via compression and decompression operations, in the encryption and decryption stages of Kyber.CPAPKE respectively, in order to reduce the ciphertext size. The functions themselves however, are not quite the same as for New Hope.

One aspect in which Kyber differs from New Hope is in the selection of cryptographic hash functions: where New Hope makes use of SHAKE-128 for the sampling of a and SHAKE-256 for all the rest, Kyber employs 4 different functions of the Keccak family: SHAKE-128, SHAKE-256, SHA3-256 and SHA3-512. SHAKE-128 is used as an extendable output function, SHAKE-256 as a pseudo-random number generator and SHA3-256 and SHA3-512 as hash functions. Tables 1 and 2 summarise the key characteristics of Kyber and contrast them to the ones of New Hope.

	New Hope	Kyber
lattice problem	RLWE	MLWE
NIST security levels	1, 5	1, 3, 5
PKE security	IND-CPA, IND-CCA	IND-CCA
R_q	$\frac{\mathbb{Z}_q[X]}{(X^n+1)}$	$\frac{\mathbb{Z}_q[X]}{(X^n+1)}$
n	512, 1024	256
q	12289	7681
module rank	not applicable	2, 3, 4
noise parameter	8	32, 16, 8

Table 1: A summary of the parameters and characteristics of the New Hope and Kyber KEMs.

5.4 Development Methodology

The implementation of the key encapsulation tool in Python 3.7 was roughly divided into three phases: firstly, the implementation of the New Hope scheme. Secondly, the construction of the GUI providing full access to the functionalities of New Hope. Thirdly, the implementation of the Kyber KEM and an upgrade of the GUI to incorporate it and offer full access to it. The implementation of the two KEMs was largely based on the descriptions and the pseudocode provided in the specification papers submitted to NIST; occasionally, clarification

	$\|pk\|$	$\|sk\|$	$\|ciphertext\|$
NewHope512-CPA	928	869	1088
NewHope1024-CPA	1824	1792	2176
NewHope512-CCA	928	1888	1120
NewHope1024-CCA	1824	3680	2208
Kyber512	736	1632	800
Kyber768	1088	2400	1152
Kyber1024	1440	3168	1504

Table 2: Key and ciphertext sizes of the New Hope and Kyber KEMs in bytes.

was gained by inspection of the relevant parts of the official C implementations. Phases 1 and 3 involved constant unittesting of all the functions that lend themselves to this form of testing. Phase 2 was largely guided by simple inspection testing, as not much can be done to see if a GUI works other than running it and using it. All the Python modules on the GitLab repository linked to this project are of our own design and creation, except for the files contained in the NTT folder; as stated before, these were obtained from a third party [18].

Each of the three phases outlined above was approached in the following way: first, a list of all the required fields and methods, complete with their parameters and outputs, was compiled on paper. After that, it was translated into a .py file with method stubs, and each stub in addition gained a brief description of the intended use of the method. Only then did the coding part begin, starting with the most low-level methods, building up slowly to the highest level ones: this way, when a method was completed, it could immediately be tested, as all the methods it needed to call had already been implemented. Finally, once all the methods were working as expected, full docstring was written.

The proper functioning of the tool can be determined by checking that the shared secret generated during encapsulation is identical to the one obtained by decapsulation of the relevant ciphertext, when the correct corresponding keys are used. This has been done systematically by providing unittests that generate keys, run encapsulation and decapsulation with the correct keys and then compare the obtained shared secrets. These tests are provided for all specified parameter sets of the two New Hope KEMs as well as the Kyber KEM. To further cement the correct functioning of the underlying PKEs, similar tests that generate keys, encrypt random messages and then decrypt the resulting ciphertexts, are provided for the PKEs.

5.5 Using the Tool

The tool is designed to be quick and easy to use. It features a straight-forward, neat presentation, to allow fast access to the functionalities it offers and is devoid of any features that are non-essential to the successful execution of the three operations of a KEM: key generation, encapsulation and decapsulation. Figure 6 shows how to start the program: all that is required is to run the KEMGUI.py file in the GUI folder of the KeyEncapsulationTool project.

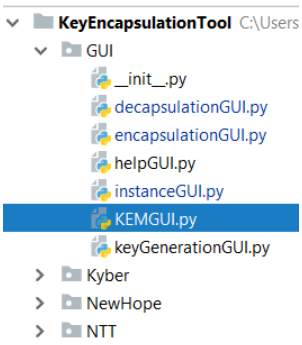

Figure 6: The program is started by running the KEMGUI.py file in the GUI folder of the KeyEncapsulationTool project.

Once the program has been started, the user is confronted with the main menu, shown in figure 7. This allows the user to choose between the three KEMs that have been implemented, New Hope IND-CPA secure, New Hope IND-CCA secure and Kyber IND-CCA secure. Additionally, the user can choose from among the various security levels supported by the KEMs. Clicking one of the buttons translates into an instance of the relevant KEM class with the appropriate parameters being created; this is done behind the scenes however and the user need not concern themself with these implementation details. Should the user not know which option to choose, they can click on the 'Help' button, which will bring up the Help page, shown in figure 8, which contains a very short introduction to the various KEMs and security levels and may aid the user in making their choice.

After the user has decided which KEM and which security level they would like to use, they click on the corresponding button, which brings up a new window, shown in figure 9. This new window asks them which KEM related action they'd like to perform, key generation, encapsulation, or decapsulation. Which choice they make determines which window they are confronted with next; the three options are shown in figures 10, 12 and 14. Note that the encapsulation and decapsulation windows appear with all their fields empty; it is then up to the user to input the desired values.

If the user clicks the 'Key Generation' button, the window shown in figure

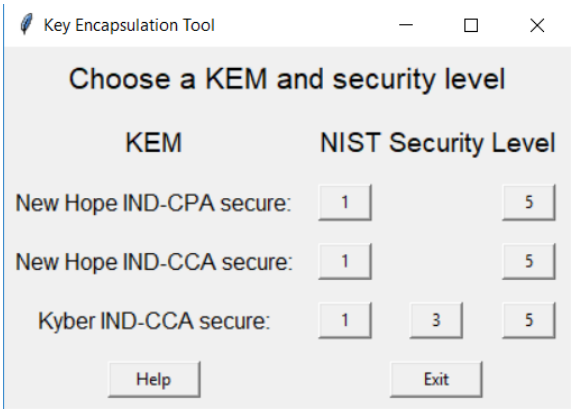

Figure 7: The main menu of the key encapsulation tool offering the user the choice between the three KEMs and their various security levels. It further allows the user to bring up a Help page for guidance on which option to choose.

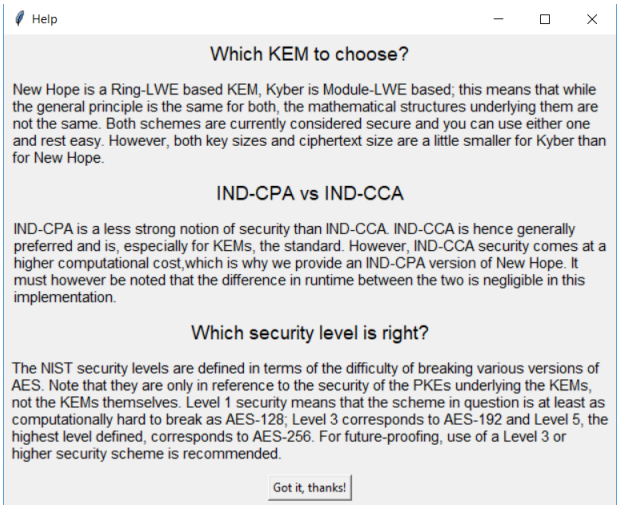

Figure 8: The help page of the key encapsulation tool.

10 pops up. All the user then has to do is to hit the 'Run' button, which will cause the underlying KEM object, which is hidden from the user, to execute its key generation method. The resulting values are displayed in the corresponding text boxes in the key generation window, as shown in figure 11. As these are text boxes, the user can select their content and use the Ctrl + C keyboard command to copy it to the clipboard.

Clicking on the 'Encapsulation' button will cause the window shown in figure 12 to appear; as noted before, it will, unlike shown in figure 12, appear with an empty public key field. Before the run operation can be successfully executed, a valid public key for the KEM that has been chosen needs to be entered into the relevant field. Once that is done, hitting the 'Run' button will execute the encapsulation operation of the underlying KEM object, with the provided public key as argument. The resulting ciphertext and shared secret are then displayed in the corresponding fields, as shown in figure 13. Again, these are text fields, so the user can select their content and copy it to the clipboard.

The third option is to use the 'Decapsulation' button, which prompts the appearance of the window shown in figure 14. Again, the figure appears with all fields blank. Once a valid ciphertext and secret key have been entered, hitting the 'Run' button executes the decapsulation method of the corresponding KEM object, with the ciphertext and secret key as arguments. This operation outputs a shared secret, which is displayed in the corresponding text field as shown in figure 15.

Note that for both encapsulation and decapsulation, the arguments the user provides to the program via the text fields are checked when the 'Run' button is hit. Should any input be invalid, either because it has incorrect size or else because it contains invalid elements, the operation will not execute and an error message will appear in the text fields reserved to display the method output.

Figure 9: Having chosen which KEM they want to use, the user is now confronted with the choice of which KEM related function they would like to execute.

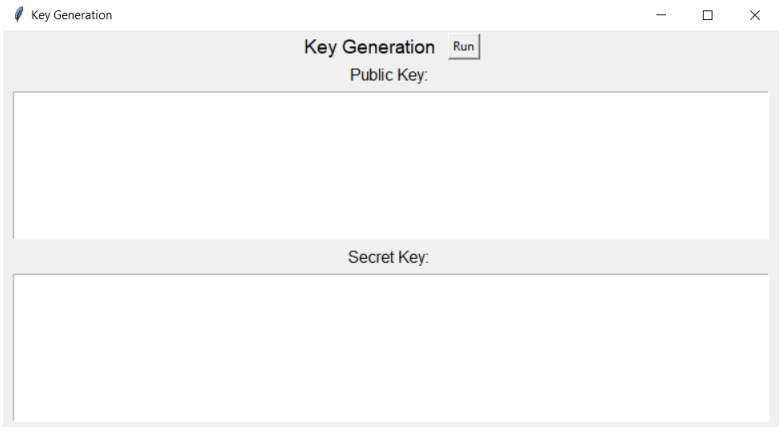

Figure 10: The key generation page, before pressing of the 'Run' button.

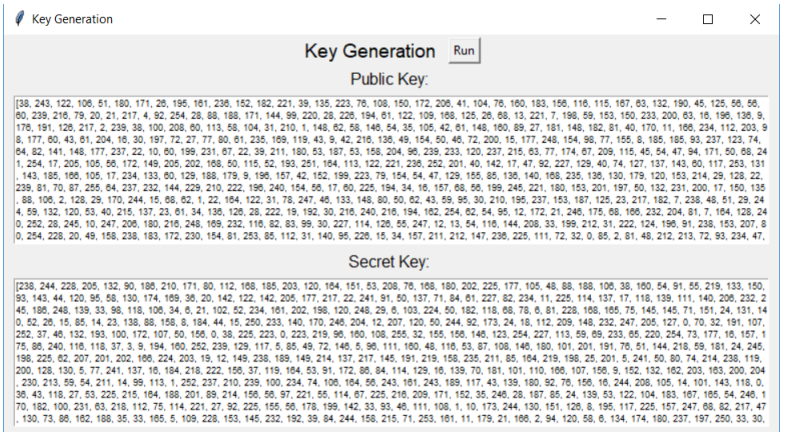

Figure 11: The key generation page, after pressing of the 'Run' button. The public and private keys are displayed in their designated fields.

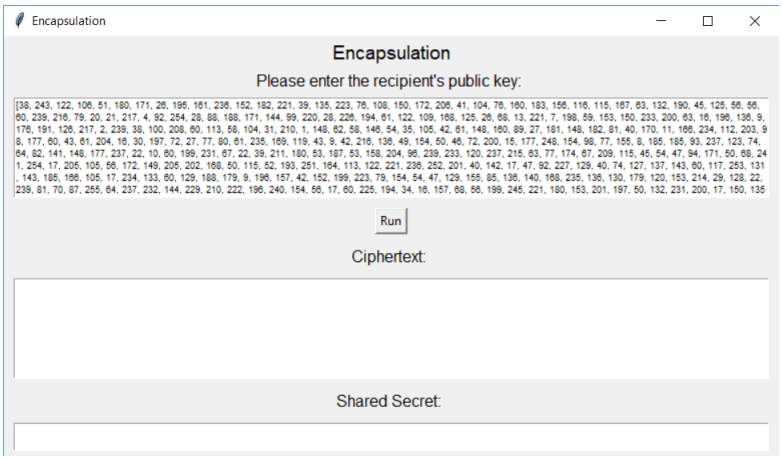

Figure 12: The encapsulation screen before the operation is run, with the recipient's public key in the designated field.

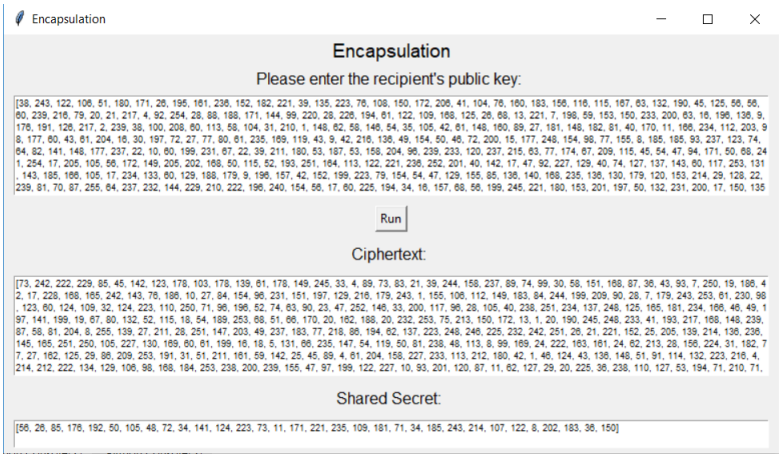

Figure 13: The encapsulation screen after the operation has been run. The recipient's public key is still visible and the ciphertext and shared secret are displayed in their respective fields.

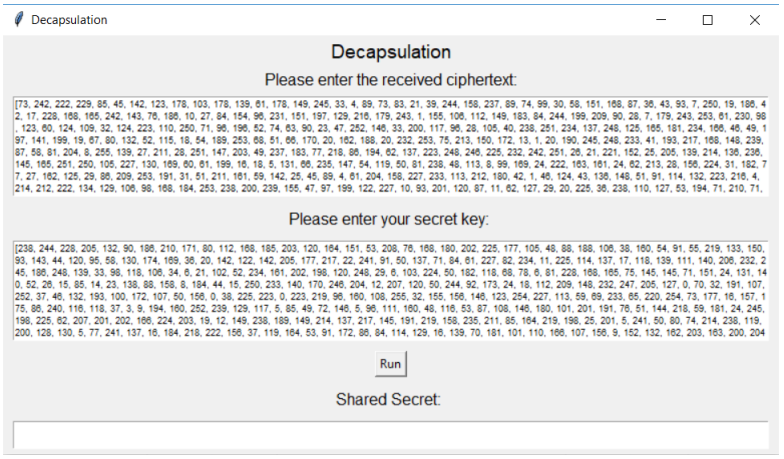

Figure 14: The decapsulation screen before the operation is run. The ciphertext and the recipient's private key have been pasted into their designated fields.

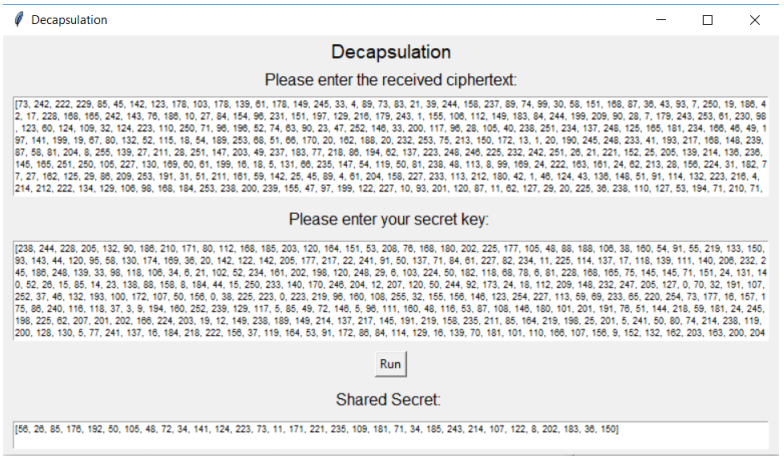

Figure 15: The decapsulation screen after the operation has been run. The ciphertext and private key are still visible and the shared secret is displayed.

5.6 Performance

Rigorous unittesting and comprehensive use testing have established that the tool performs correctly and the shared secrets that the two parties obtain are indeed identical. While this is quite obviously the most important result for the tool to achieve, the performance in terms of execution speed leaves something to be desired. As New Hope is based on RLWE and Kyber is based on MLWE, one would expect New Hope to be faster than Kyber, if two instantiations with equal security levels are compared. This assumption is supported by the cycle counts reported in [3] and [6], for the three KEM operations of the reference implementations of New Hope and Kyber, respectively. These report that, on a single thread of an Intel Core i7-4770K, New Hope IND-CPA operations require the fewest CPU cycles, followed by New Hope IND-CCA, with Kyber IND-CCA requiring the most. As the computational bottleneck of these schemes lies in the multiplication operations, this result is not astonishing, since the multiplication of modules is more expensive than that of mere polynomials. The optimised implementations that make use of AVX vector instructions present a different picture: leaving aside the decapsulation operation, since the IND-CPA scheme does not make the same use of hash functions as the IND-CCA ones, the order of increasing cycle count now goes as follows: Kyber IND-CCA, New Hope IND-CPA, New Hope IND-CCA. This may point to the fact that with AVX vector support, multiplications of modules over a polynomial ring of smaller dimension n is faster than multiplication of simple polynomials over a ring with larger dimension.

Our findings for the runtime of this tool mirror the results of the AVX implementations: it appears that, perhaps counter-intuitively, Kyber IND-CCA is the fastest scheme, with both New Hope IND-CPA and New Hope IND-CCA being noticeably slower. While we do not know what exactly causes this, and sub-optimal implementation on our part cannot be ruled out as the cause, the hypothesis that the performance may be linked to the way Python handles vector operations is plausible. Further impact may come from the NTT library used in this code, which may fare better with polynomials of a smaller dimension. These observations are however in no way meant to be seen as an attribution of blame and are purely stated as a starting point for potential further investigation of the matter.

5.7 Cryptographic Code Libraries Employed

5.7.1 From the Python Standard Library

1. hashlib: Provides cryptographic hash functions. The following functions were used:
   - SHA3-256
   - SHA3-512
   - SHAKE-128
   - SHAKE-256
2. secrets: Provides cryptographically secure randomness generated using the most secure source of randomness offered by the operating system.

5.7.2 External

1. Number-theoretic transform, by Project Nayuki. [19]

6 Conclusion

This project led to the successful creation of a post-quantum secure, lattice-based key encapsulation tool, the likes of which, to the best of our knowledge, did not previously exist. The tool is based on two distinct submissions to the NIST call for proposals, based on two different hard lattice problems. The project furthermore led to a significant deepening of our comprehension and appreciation of cryptography in general and the challenges associated with post-quantum cryptography in particular. A solid understanding of the mathematical foundation of lattice-based cryptography was developed, together with a firm grasp of the major hard computational problems that fall under its umbrella. Additionally, our proficiency in the use of the Python programming language was greatly enhanced. As such, all the stated aims of this project have been met and the project is considered a success.

The major challenges encountered during this project were largely expected: delving into a new, advanced area of mathematics and a cutting-edge branch of cryptography were considerable hurdles, especially given the time constraints imposed on the project. Getting to grips with an essentially new programming language was a serious obstacle as well. However, one difficulty that caught us by surprise lay in the actual implementation of the two cryptographic systems; we certainly did not expect this to be trivial, yet we encountered a bigger hindrance than anticipated: although the technical specifications submitted as part of the proposals to NIST were hugely helpful and absolutely crucial to the success of this project, they nonetheless contained both pseudocode mistakes (probably due to typos) and some omissions; this means that we were required to understand the workings behind the code at a fundamental level in order to fill what gaps there were, as well as identify and overcome errors.

Had there been more time to complete this project, the next step would have been to include more cryptosystems, based on problems other than RLWE and MLWE; the next candidate we would have liked to implement would have been the Frodo KEM, based on LWE. As this would be based on entirely uncorrelated samples from the LWE distribution, it would be resistant to any attacks that may target the mathematical structure inherent to RLWE and MLWE schemes and thus assure the survival of the tool if both New Hope and Kyber should be broken by exploitation of their structure. Another useful option for further work would be in making the KEMs resistant to man-in-the-middle attacks by using authenticated keys. Finally, had there been more time, we would have liked, as mentioned in section 5.6, to conduct a thorough investigation of the execution time discrepancies between the New Hope and the Kyber schemes.

References

[1] M. Ajtai. "Generating Hard Instances of Lattice Problems (Extended Abstract)". In: *In Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*. ACM, 1996, pp. 99–108.

[2] M. R. Albrecht and A. Deo. "Large Modulus Ring-LWE ≥ Module-LWE". In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by T. Takagi and T. Peyrin. Cham: Springer International Publishing, 2017, pp. 267–296. ISBN: 978-3-319-70694-8.

[3] E. Alkim, R. Avanzi, J. Bos, L. Ducas, A. de la Piedra, T. Pöppelmann, P. Schwabe, and D. Stebila. "NewHope - Algorithm Specifications and Supporting Documentation". Version 1.0. In: Round 1, NIST Call for Proposals for Post-Quantum Cryptography Standardization. (Nov. 30, 2017).

[4] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. *Post-quantum key exchange - a new hope*. Cryptology ePrint Archive, Report 2015/1092. `https://eprint.iacr.org/2015/1092`. 2015.

[5] anupsv. *NewHope-Key-Exchange*. 2016. URL: `https://github.com/anupsv/NewHope-Key-Exchange` (visited on 08/28/2018).

[6] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanke, P. Schwabe, G. Seiler, and D. Stehlé. "CRYSTALS-KYBER - Algorithm Specifications And Supporting Documentation". In: Round 1, NIST Call for Proposals for Post-Quantum Cryptography Standardization. (Nov. 30, 2017).

[7] Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé. "Classical Hardness of Learning with Errors". In: *CoRR* abs/1306.0281 (2013). arXiv: `1306.0281`. URL: `http://arxiv.org/abs/1306.0281`.

[8] Catslash. *Lattice reduction in two dimensions: the black vectors are the given basis for the lattice (represented by blue dots), the red vectors are the reduced basis*. 2008. URL: `https://en.wikipedia.org/wiki/File:Lattice-reduction.svg` (visited on 08/25/2018).

[9] S. I. R. Costa, F. Oggier, A. Campello, J.-C. Belfiore, and E. Viterbo. *Lattices Applied to Coding for Reliable and Secure Communications*. Springer Briefs in Mathematics. Springer, 2017.

[10] Dav-FL-IN-AZ-id. *Illustration of the Diffie-Hellman key exchange*. Dec. 10, 2011. URL: `https://commons.wikimedia.org/wiki/File:Diffie-Hellman_Key_Exchange.png` (visited on 08/25/2018).

[11] E. Fujisaki and T. Okamoto. "Secure Integration of Asymmetric and Symmetric Encryption Schemes". In: *Advances in Cryptology — CRYPTO' 99*. Ed. by M. Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 537–554. ISBN: 978-3-540-48405-9.

[12] J. Hoffstein, J. Pipher, and J. H. Silverman. *An Introduction to Mathematical Cryptography*. Second Edition. Undergraduate Texts in Mathematics. Springer, 2014.

[13] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. *A Modular Analysis of the Fujisaki-Okamoto Transformation*. Cryptology ePrint Archive, Report 2017/604. https://eprint.iacr.org/2017/604. 2017.

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

[15] V. Lyubashevsky, C. Peikert, and O. Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *Advances in Cryptology – EUROCRYPT 2010*. Ed. by H. Gilbert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–23. ISBN: 978-3-642-13190-5.

[16] K. M. Martin. *Everyday Cryptography. Fundamental Principles and Applications*. Second Edition. Oxford University Press, 2017.

[17] Vasileios Mavroeidis, Kamer Vishi, Mateusz D. Zych, and Audun Jøsang. "The Impact of Quantum Computing on Present Cryptography". In: *CoRR* abs/1804.00200 (2018). arXiv: 1804.00200. URL: http://arxiv.org/abs/1804.00200.

[18] Project Nayuki. *Number-theoretic transform (integer DFT)*. June 7, 2017. URL: https://www.nayuki.io/page/number-theoretic-transform-integer-dft (visited on 08/21/2018).

[19] Project Nayuki. *Number-theoretic transform library (Python 2, 3)*. 2017. URL: https://github.com/nayuki/Nayuki-web-published-code/tree/master/number-theoretic-transform-integer-dft (visited on 07/13/2018).

[20] Scott Wyman Neagle. *PyNewHope*. 2017. URL: https://github.com/scottwn/PyNewHope/blob/master/newhope.py (visited on 08/28/2018).

[21] C. Paar and J. Pelzl. *Understanding Cryptography. A Textbook for Students and Practitioners*. Springer, 2010.

[22] C Peikert. *A Decade of Lattice Cryptography*. Cryptology ePrint Archive, Report 2015/939. https://eprint.iacr.org/2015/939. 2015.

[23] C. Peikert. *Lattices in Cryptography*. Lecture Notes, Lecture 2, Georgia Institute of Technology. 2013.

[24] C. Peikert. *Lattices in Cryptography*. Lecture Notes, Lecture 6, Georgia Institute of Technology. 2013.

[25] C. Peikert. *Public-Key Cryptosystems from the Worst-Case Shortest Vector Problem*. 2008.

[26] O. Regev. *Lattices in Computer Science*. Lecture Notes, Lecture 12, Tel Aviv University. 2004.

[27] O. Regev. "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography". In: *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*. STOC '05. Baltimore, MD, USA: ACM, 2005, pp. 84–93. ISBN: 1-58113-960-8. DOI: 10.1145/1060590.1060603. URL: http://doi.acm.org/10.1145/1060590.1060603.

[28] P. Schwabe. *Lattice-based cryptography – Episode IV. A new hope.* Lecture Slides, Radboud University. June 23, 2017.

[29] S. Singh. *The Code Book. The Secret History of Codes and Code-Breaking.* Fourth Estate, 1999.

[30] National Institute of Standards and Technology. *Post Quantum Cryptography Call for Proposals.* Jan. 3, 2017. URL: `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Call-fo%20r-Proposals` (visited on 08/21/2018).

Acronyms

AES Advanced Encryption Standard.

BDD Bounded Distance Decoding.

CVP Closest Vector Problem.

DES Data Encryption Standard.

DH Diffie Hellman.

GGH Goldreich Goldwasser Halevi.

IND-CCA Indistinguishability under Chosen Ciphertext Attack.

IND-CPA Indistinguishability under Chosen Plaintext Attack.

KEM Key Encapsulation Mechanism.

LWE Learning With Errors.

MLWE Module Learning With Errors.

NIST National Institute of Standards and Technology.

NTT Number-Theoretic Transform.

PKE Public-Key Encryption.

RLWE Ring Learning With Errors.

RSA Rivest Shamir Adleman.

SIS Short Integer Solution.

SVP Shortest Vector Problem.

Appendix: Code Repository

The code repository associated with this project can be found at:

https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2017/gas718

The repository contains the KeyEncapsulationTool project, which contains the following packages:

- .idea
- GUI: Contains all files related to the user interface.
- Kyber: Contains all files related to the Kyber cryptosystem.
- NTT: Contains all files related to the number-theoretic transform; all files in this folder were obtained from a third party and are used as a library. [19]
- NewHope: Contains all files related to the New Hope cryptosystem.

Unittest files were created for each .py file contained in the Kyber and NewHope packages and can be found alongside the files they correspond to.

In order to start the key encapsulation tool, run the KEMGUI.py file in the GUI package. This requires the secrets and the hashlib libraries from the Python standard library to be installed.