Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Interim Report

Project Title:	**Post-Quantum Key Exchange over the Internet**
Student:	**Soham Bapat**
CID:	**01382601**
Course:	**MEng Electrical and Electronic Engineering with Management**
Project Supervisor:	**David Joseph**
First Marker:	**Dr Cong Ling**
Second Marker:	**Dr Wei Dai**

Contents

We use Internet on a daily basis, whose security relies on modern public-key cryptography. However, existing key exchange protocols over the Internet based on RSA and Diffie-Hellman would be totally broken with the emergence of quantum computers. Lattice-based cryptography offers an answer to deal with this disaster. It also appears to be more efficient than existing key exchange protocols. In this project, you will firstly familiarize yourself with key exchange protocols in TLS, then implement in software a new, quantum-safe protocol based on lattices.

1. Introduction and Project Specification

Recent advancements in quantum computing have led to fears that existing key exchange protocols used on the Internet are likely to be broken soon by quantum computers, thus compromising the security of most activity online. This would have severe consequence on financial transactions, parts of national infrastructure (e.g. smart power grids) and privacy of conversations to name a few.

Researchers have come up with new protocols that are likely to be safe and suitable in this post-quantum world, which have been submitted to NIST (National Institute of Science and Technology) to be standardised for public use [32]. Many of these promising candidates are based around lattice structures, which are referred to as lattice-based cryptography.

The primary deliverable of this project is a Python implementation of the 'CRYSTAL Kyber CPAPKE' algorithm (henceforth referred to as Kyber), which is one of the three final lattice-based candidates that have passed NIST's Round 3 selection. The other candidates are 'NTRU' and 'Saber' [32]

Kyber is an algorithm where the most value can be added through this project. For example, NTRU is an encryption algorithm which has been investigated and researched thoroughly with many algorithm implementations on GitHub. At present, there is no publicly available complete Python implementation of the Kyber algorithm.

Unlike Saber, Kyber is already being implemented in industry and is the most likely candidate to be chosen [32]. Most notably, Amazon and Cloudflare have integrated Kyber in some of their services [38].

Finally, Kyber is also the simplest scheme of the three and uses a similar framework to Dilithium, a post quantum digital signature algorithm. For this reason, it is also the most suitable to implement over the time frame of a Final Year Project.

The deliverable will comprise of:

- Mathematical overview of the Kyber algorithm and associated problems
- Detailed and well documented Python implementation of Kyber, complete with testing.
- Performance testing against other encryption algorithms such as RSA and Diffie-Hellman for key generation, encryption and decryption
- Performance/security improvements to the Kyber algorithm.

The main reason behind creating a Python implementation of Kyber is to act as a proof of concept and as a test-bed for developers to ensure their app is compatible. Developers may also wish to integrate this algorithm now as an added layer of security.

Further details on the deliverable have been included in Section 3.

2. Background

Cryptography is a way of enforcing policies to ensure that only the intended people can view and/or modify data. More specifically, this project focuses on communication cryptography which is the study of secure communication practices to ensure that only the sender and intended recipient of a message can view and/or modify its contents.

The process of encoding messages has early examples dating back to the Roman Era and the use of Caeser ciphers [1]. Each letter was replaced by another letter a fixed number of positions, thus encoding the message. For instance, A would become C, C would become E under a particular key. The number and direction of shifts were determined in person prior to communications being sent.

2.1. Private Key Cryptography

Caeser cipher is an example of symmetric encryption, where the same key is used to encrypt and decrypt the message. With the example earlier, the key would be 'right shift of two'. Encoding the message would involve performing this shift whilst decoding the message would involve reversing this operation by performing a left shift of two to retrieve the original message.

Symmetric encryption has now evolved with more complicated encryption mechanisms and larger key sizes as computers have gotten more powerful. Symmetric encryption is used to share information over the Internet and is more commonly known as private key encryption. To illustrate this example, we consider two people who wish to communicate securely- Alice and Bob.

Assume Alice and Bob both have the same key in advanced. If Alice wants to send a message (referred to as plaintext) to Bob, she will encrypt the plaintext using the key. The method by which the key encrypts the message depends on the protocol being used. Examples include taking the modulus of numbers, byte substitutions and XOR operations. Once the plaintext has been encrypted it is referred to as ciphertext. The ciphertext is then sent over the Internet (which can be an insecure communication line). If the key is sufficiently strong, no middleman will be able to decode the message. Once Bob receives the message, he will decrypt the plaintext using the same key by reversing the operations used to encrypt the message.

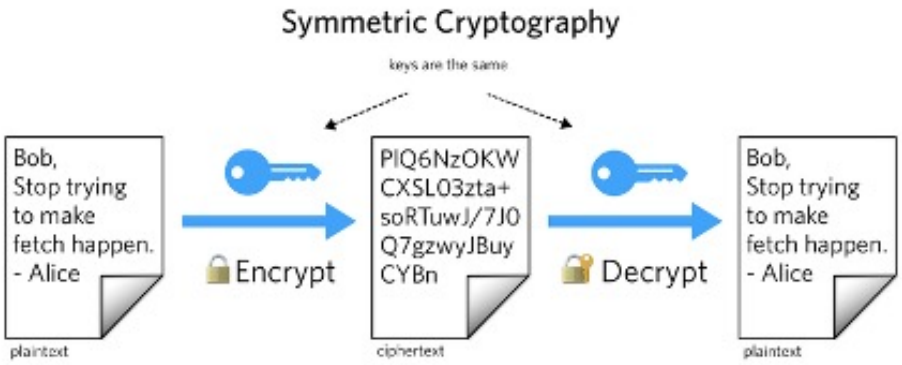

Figure 1: Private key cryptography [39]

The most widely used private key algorithm is AES (Advanced Encryption Standard), usually AES-256 (256-bit key). The only known attack against AES-256 is a brute force attack, where the attacker tries every possible key combination til the correct one is found. Assuming the most powerful supercomputer has a speed of 415 petaflops which can check 4.15×10^{14} keys per second (1000 operations taken per key check), it would take 4.42×10^{54} years to check and find the correct key, assuming on average it will take half the keyspace to find the solution. For this reason, AES-256 is considered to be a secure algorithm to communicate with.

Whilst private key cryptography is fast and secure, it requires the sender and recipient to have knowledge of the key prior to communication happening. Due to the speed and number of connections on the Internet per day, it is infeasible for each sender and receiver to meet and agree on the key to be used. Asymmetric encryption, also known as public key cryptography, aims to solve this issue.

2.2. Public Key Cryptography

With public key cryptography, the receiver (Bob) will generate a pair of keys which are mathematically linked: a private key and a public key. The public key can be shared online so anyone can communicate with the receiver, whereas the private key is kept secret to ensure the communication is secure. The sender (Alice) will encrypt the plaintext using the public key and send the ciphertext. Bob will then decrypt this ciphertext using his private key.

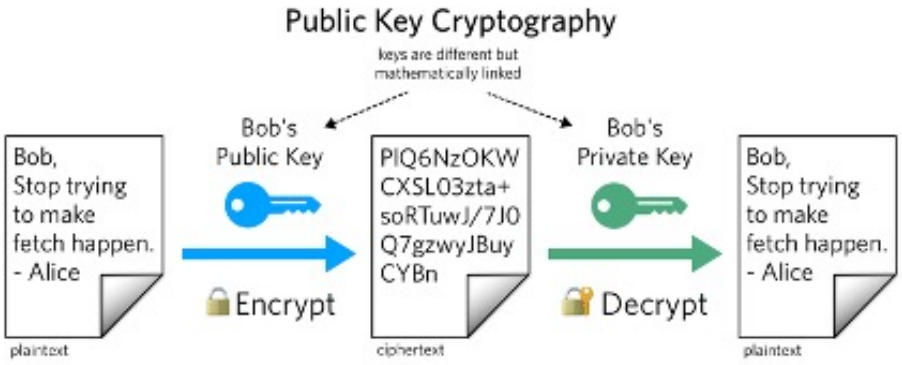

Figure 2: Public key cryptography illustrated [39]

Public key cryptography is asymmetric as different keys are used to encrypt and decrypt the plaintext message. Public key cryptography relies on the use of 'trapdoor functions' to create a pair of public and private keys. Trapdoor functions are easy to compute one way i.e. finding the public key from the private key, yet extremely difficult to compute the other way (finding private key from public key) which makes them suitable for this use.

The most common public key cryptography function is RSA. The steps to generate a pair of private and public keys using RSA are detailed below [42].

1. Choose two different large and random prime numbers: p and q. The length of these is a parameter which can be determined prior to the key generation function.
2. Compute n, where $n = pq$. n forms a part of the public key.
3. Calculate Euler's function for n, $\phi(n)$. Where p and q are prime, $\phi(n) = (p-1)(q-1)$. Alternative implementations of RSA include the use of Carmichael's totient function instead of Euler's function, however this does not change the method nor the encryption security.
4. Choose an integer e between 1 and n that is co-prime with $\phi(n)$. Usually e is taken to be 65537, as this is often co-prime due to it being the largest known Fermat prime. It should be noted that e being the same across multiple keys isn't a security risk.
5. Calculate d, where $d = e^{-1} mod(\phi(n))$

Where (n, e) forms the public key and (n, d) forms the private key.

It has been shown that the most practical way to break RSA Encryption is by using prime factorization [16] [12] to find the two prime numbers p and q from the public key n to calculate the private key d, as this allows for easy calculation of $\phi(n)$. However when the numbers are sufficiently large, there is currently no known feasible non-quantum solution for prime factorization. It is estimated that it would take 900 years to factor a 240-digit number (RSA-240) [11]. Modern RSA encryption keys range from 1024 bits to 4096 bits, which would take upwards of 500 times as long [25]. For this reason, it is assumed that with conventional computers and current computing power RSA with large bits is secure.

Public key cryptography solves the key sharing issue faced by private key cryptography; however, it is at the expense of a significant performance trade off. RSA is between 3-5 times slower in encryption and decryption compared to AES, which is a significant increase for low latency usage [33]. A big reason for this is the increased key sizes for RSA to achieve a comparable level of security to AES. Typical RSA key sizes are 1024 bits (and higher) compared to 256 bits for AES. This added size means more operations are required to encrypt and decrypt the message.

2.3. Public Key Exchange + Private Key Cryptography protocol

The performance trade off with public key cryptography led to the development of a new protocol which uses both private and public key cryptography.

The session between the two users is authenticated and created using public key cryptography. Using public key cryptography, the private key is exchanged as the message (plaintext). Once the session is established and both users have the same private key, all communication is encrypted using the private key. A prominent example of this is the TLS protocol used to transmit HTTPS web pages [6] [10]. This protocol combines the security of public key cryptography with the speed of private key cryptography.

2.4. Requirement for post quantum encryption

Whilst the RSA algorithm is not currently breakable in reasonable time with traditional computers, it may be possible to do so using quantum computers. An algorithm proposed by Peter Shor in 1994 titled "Shor's Algorithm" is a polynomial time quantum computer algorithm to factor integers[40]. Using Shor's Algorithm, one could work out the prime factors of a large number in $O((log(N))^2(log(log(N)))(log(log(log(N))))$ time and $O(log(N))$ space, where N is the size of the integer.

Shor's Algorithm solves the prime factorization problem by transforming it into a period finding problem through the use of Quantum Fourier Transform [15]. The process below shows a simplified version of the algorithm [28]:

1. Let n be the number to factorise, store this in a 'qubit register'. With a quantum computer, only those binary bits are required due to superposition. For instance, $n = 15$ requires only 4 qubits (quantum bits). However, on a classical computer, all numbers from 0 to n have to be stored separately.
2. Choose variable x at random such that $1 < x < n - 1$.
3. Raise x to powers of the qubit registers (ranging from 0 to n inclusive), divide by n and store the remainder in another qubit register. Assuming $x = 2$, $n = 15$ the output of the two registers is detailed below.
4. From the second register, a pattern of repeating sequence of numbers can be seen: $(1, 2, 4, 8)$. Let the length of this repeating sequence be f. $f = 4$
5. A possible factor can be calculated: $P = x^{f/2} - 1 = 3$. If this isn't a prime number, repeat the calculation from Step 2. If this is a prime number, calculate the other corresponding number (5) and return the result.

First register (range of integers): [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
Second register (remainders): [1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4, 8]

The full Python implementation of this algorithm can be found in Figure 9 and in the GitHub repository [8]. This was developed and used to gain better intuition and demonstrate Shor's Algorithm to the reader.

From Table 1, it can be seen that commonly used public key cryptography methods will be completely broken using Shor's Algorithm.

Cryptographic Algorithm	Type	Purpose	Impact from Quantum Computers
AES-256	Private key	Encryption	Secure
RSA	Public key	Encryption, key establishment	No longer secure
ECDSA, ECDH	Public key	Encryption, key establishment	No longer secure
DSA	Public key	Encryption, key establishment	No longer secure

Table 1: Impact of Quantum Computers on current encryption algorithms. Adapted from [28] and [14]

It should be noted that the table shows the eventual impact in the long term. Current quantum computing technology falls short of the required qubits required for Shor's Algorithm for most RSA implementations. The exact number of qubits required to factorise a number is debated amongst researchers, with some sources saying $10N$ qubit relationship [23] whilst others claiming a $2N + 3$ relationship [9] with further optimisations. The exact number is debated for a variety of reasons, most notably the variety of implementations and algorithms. Furthermore, the 'quality' of each qubit also affects its computational capability. If a qubit is noisy then substantially more qubits are required to perform the calculation to verify the results. Presently the world record for quantum computing power stands at 72 qubits. The current record for largest integer factored using Shor's Algorithm on a quantum computer is 35 [5].

Given that time will be required to standardise an encryption algorithm and IBM are targeting a 2023 release date for their 1000 qubit computer [17], it is of great importance to develop and standardise a quantum safe encryption algorithm in time. At present, there is currently no known quantum algorithm that solves hard lattice problems significantly better than current algorithms designed to work on classic computers [30]. Classic computers are also unable to solve hard lattice problems in polynomial time. For this reason, lattice-based cryptography is seen as a promising candidate for secure post-quantum encryption.

2.5. Lattices

This section outlines the basics of lattices, which will be used to define lattice-based cryptography later.

A lattice (L) is an algebraic structure that is formed of a set of n independent vectors b_1 .. $b_n \in \mathbb{R}^n$, where $L = \sum_{i=1}^{n} x_i b_i$ where $x_i \in \mathbb{Z}$. Lattices have two basic properties [30]:

1. Additive Subgroup, $0 \in L$, $-x, x + y \in L$ for every $x, y \in L$
2. Discrete, there exists a certain delta for which x is the only lattice point in the neighbourhood (where $x \in L$)

A lattice's domain is expressed through the fundamental parallelepiped. The determinant of a lattice $(det(L))$ is the n dimensional volume of the fundamental parallelepiped. The fundamental parallelepiped corresponds to the set:

$L = \sum_{i=1}^{n} z_i b_i$ where $z_i := [-0.5, 0.5)$

The above is more specifically the origin centred parallelepiped. Alternative expression of a parallelepiped may include using the range $z_i := [0, 1)$ [22].

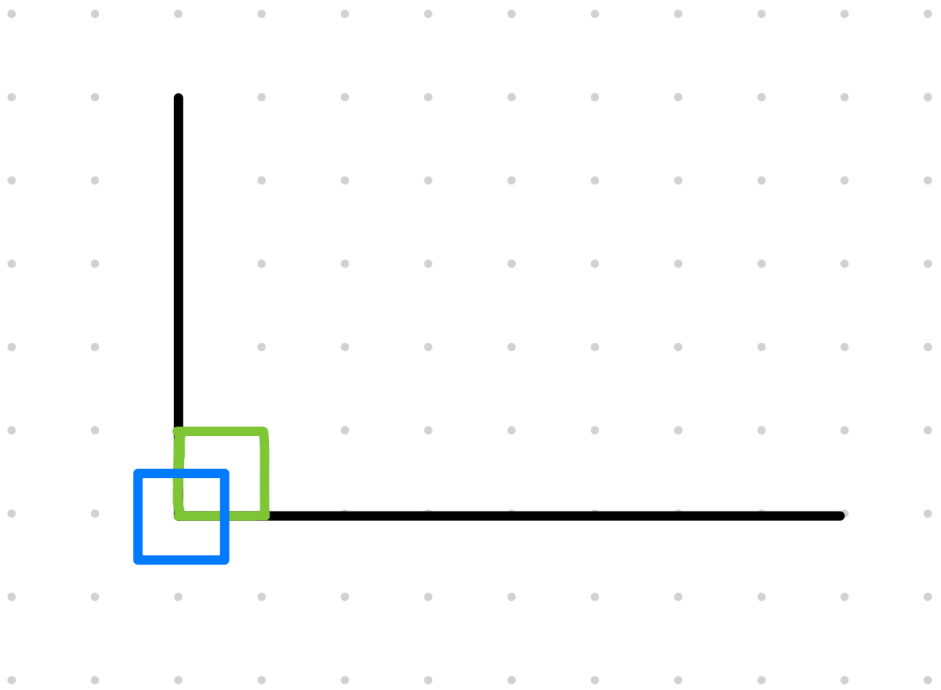

Figure 3: Fundamental basis (F) for lattice L with basis: (0,1) and (1,0). Green = Fundamental basis where $z_i := [0, 1)$. Blue = Origin fundamental basis where $z_i := [-0.5, 0.5)$.

2.6. Groups, Rings and Fields

This section details parts of set theory, which is used in lattice-based cryptography.

2.6.1. Groups

A Group G is defined to be a set which satisfies the below conditions [35]. To illustrate the properties the addition operation is considered:

1. $I \subset G$, where I is the identity element. Under addition, $I = 0$.
2. For every element $x \in G$, there exists a corresponding element $y \in G$ such that when the operator is used, the identity is the result. Under addition, $x + y = 0$ where $y = -x$.
3. Operations are associative, under addition $x, y, z \in G$, $x + (y + z) = (x + y) + z$
4. Set is closed under the operation. For $x, y \in G$ under addition, $(x + y) \in G$

2.6.2. Rings

A Ring R is defined to be a set which satisfies the below conditions. To illustrate the properties two operations are considered, addition and multiplication:

1. Set is closed under two operations. For example for $x, y \in R$ under addition, $(x + y) \in R$
2. R is an abelian group under addition
3. Associative property, for $x, y, z \in R$, $x + (y + z) = (x + y) + z$
4. Distributive property, for $x, y, z \in R$, $x \times (y + z) = (x \times y) + (x \times z)$

A ring is a set which is a group under addition and satisfies some of the properties of a group for multiplication [13]. The set $\mathbb{Z}$ is an example of a group and a ring.

A polynomial ring is defined as a ring whose coefficients are from the ring R [22]:

$R[x] = \sum_{i=1}^{n} z_i x$ where $z_i \in R$

2.6.3. Fields

A Field F is defined to be a set which satisfies the below conditions. To illustrate the properties two operations are considered, addition and multiplication:

1. Set is closed under two operations. For example for $x, y \in F$ under addition, $(x + y) \in F$
2. F is an abelian group under addition
3. $F - 0$, i.e. set F without the additive identity 0, is an abelian group under multiplication

A field is a set which is a group under addition *and* multiplication.

2.7. Hard Lattice Problems

This section details hard lattice problems. The proof of security for all lattice-based cryptography is based upon the hardness of lattice problems [34]. The problems listed below are assumed to be hard such that it is not possible to solve them in polynomial time [30].

Definition 2.1 (Shortest Vector Problem)**.** The fundamental and most studied hardness problem is the Shortest Vector Problem. The goal is to find a shortest non zero lattice vector for a given lattice. This problem forms some of the fundamental theory with lattice-based cryptography and has been estimated to be an NP-Hard problem [29].

Given lattice L and basis b_1 .. $b_n \in \mathbb{R}^n$, find a $v \in L$ for which $\|v\| = \lambda_1(L)$

Definition 2.2 (Unique Shortest Vector Problem)**.** This problem is a more specific version of the Shortest Vector Problem. Unlike the Shortest Vector Problem where multiple solutions may exist, the Unique Shortest Vector Problem guarantees that there is only a single solution to the problem [3].

Given lattice L and basis b_1 .. $b_n \in \mathbb{R}^n$, find **the** $v \in L$ for which $\|v\| = \lambda_1(L)$

Definition 2.3 (Approximate Shortest Vector Problem)**.** This is a simplification of the Shortest Vector Problem. Given a Lattice L and some basis b_1 .. $b_n \in \mathbb{R}^n$, the goal is to find a non zero vector which fits the criteria below. By setting the parameter $\gamma(n) = 1$, the Shortest Vector Problem can be recovered.

Given lattice L and basis b_1 .. $b_n \in \mathbb{R}^n$, find a $v \in L$ for which $\|v\| \leq \gamma(n) \cdot \lambda_1(L)$

Definition 2.4 (Decisional Approximate SVP)**.** This problem relates to the Approximate Shortest Vector Problem detailed earlier. Unlike the Approximate Shortest Vector Problem, there are proofs for the following decisional problem [41].

Given lattice L and basis b_1 .. $b_n \in \mathbb{R}^n$, determine whether $\lambda_1(L) \leq 1$ or $\lambda_1(L) > \gamma(n)$ (with knowledge that only one of the two is correct)

Definition 2.5 (Approximate Shortest Independent Vectors Problem)**.** This problem is a simplification of the Shortest Vector Problem. The original SVP may be recovered by setting the right hand side of the inequality to $max\|b_i\|$ for basis b_1 .. b_n.

Given lattice L and basis b_1 .. $b_n \in \mathbb{R}^n$, find a set $S \subset L$ of n linearly independent lattice vectors, where $\|s_i\| \leq \gamma(n) \cdot \lambda_n(L)$ for all vectors found.

Definition 2.6 (Bounded Distance Decoding Problem)**.** This problem can be thought of as a simplification of the Closest Vector Problem.

The Bounded Distance Decoding problem is to find the unique lattice vector (v) to a given point, with an additional guarantee the vector is within a certain distance ($d = \lambda_1(L)/2\gamma(n)$). The Closest Vector Problem does not have such a guarantee and can work for any point in the space, whereas the Bounded Distance Decoding problem only works for points close to the lattice.

Given lattice L, basis b_1 .. $b_n \in \mathbb{R}^n$ and target $t \in R^n$, find the unique lattice vector $v \in L$ such that $\|t - v\| < d$ where $d = \lambda_1(L)/2\gamma(n)$

2.8. Early achievements in lattice-based cryptography

This section introduces some of the early uses of lattice-based cryptography. Further details on the algorithms can be found in Section 7.2.

The first lattice-based cryptography algorithm was introduced by Ajtai and Dwork in 1997. The basis of this work has origins in Ajtai's lattice paper in 1996. [31][2]. Ajtai introduced the Shortest Integer Solution problem, [34] and the corresponding one way functions to be used.

2.8.1. Shortest Integer Solution

Given a lattice L whose basis is $b_1 \,..\, b_m \in \mathbb{R}^n$, find a nonzero vector z, where $\|v\| \leq \beta$, such that $\sum_{i=1}^{m} v_i b_i = 0 \in \mathbb{Z}^n$.

Where: $\beta, m \in \mathbb{R}^+, n, p \in \mathbb{Z}^+$ are parameters of the Shortest Integer Solution problem.

In the paper, Ajtai proved that solving the Shortest Integer Solution problem is at least as hard as the Approximate Shortest Independent Vector Problem, given suitable parameter choices [2]. Further research from Goldreich proved that the problem is also collision resistant [18], giving the problem a stronger security guarantee [30] [34].

2.8.2. Ajtai-Dwork Encryption

Following from Ajtai's Shortest Integer Solution problem paper, Ajtai and Dwork published the first lattice-based public key cryptography algorithm in 1997 [3]. The basis of this paper included Ajtai's findings in 1996, as well as the Unique Shortest Vector Problem [3].

To generate public and private keys, a new instance is created. The instance is a collection of hidden hyperplanes ($b_1 \,..\, b_n$) which forms the basis of the private key. From this, a point close to the private key is generated which forms the public key.

In the paper, Ajtai and Dwork proved that the constructed public key cryptography algorithm from the Hidden Hyperplane Problem had a worst case hardness of the Unique Shortest Vector Problem [3] [30]. This was seen as a breakthrough, as the Hidden Hyperplane Problem had only an average case hardness of the Unique Shortest Vector Problem.

However, this encryption algorithm was deemed to be unfeasible for use in a practical implementation. The public key size and encryption time required would be $O(n^4)$, while the private key and decryption time required would be $O(n^2)$ [34] [30]. Furthermore, since the algorithm only encrypts bit by bit, the ciphertext for a 128-bit symmetric key would be in the size of several megabits [22].

2.8.3. Other early achievements

GGH (by Goldreich, Goldwasser and Halevi) is a lattice-based public key cryptography system, which was inspired by Ajtai's early work [20]. This paper also included a digital signature scheme based on lattice problems. However, these proposals did not come with any worst case security guarantees making it unsuitable for standardisation [34].

NTRU (by Hoffstein, Pipher and Silverman) is another lattice-based public key cryptography algorithm which uses polynomial rings [21]. It has significant advantages over pure Ajtai-Dwork. For instance, the key sizes are much more compact and it has withstood cryptanalytic efforts with correct parameter initialisation. However, there is little theoretical understanding of the average case computational problems with no known reduction from any worst case lattice problem to the NTRU problem [30].

2.9. Learning with Errors

LWE (Learning with Errors) was first introduced by Regev in 2005 [36], and closely relates to the Shortest Integer Solution problem defined earlier by Ajtai [34]. The Kyber algorithm expands upon Learning with Errors to build a lattice-based public encryption algorithm.

An intuitive example of the LWE problem is to look at a system of linear equations. Suppose we want to recover a secret $s \in Z_q^n$ given a sequence of *approximate* linear equations [37]:

$5s_1 + 10s_2 + 7s_3 \approx 10 = 10 + \epsilon$

$4s_1 + 6s_2 + 5s_3 \approx 16 = 16 + \epsilon$

$9s_1 + 5s_2 + 3s_3 \approx 89 = 89 + \epsilon$

Where $\epsilon \in \mathbb{Z}$

The approximate nature of the linear equations comes from the $\approx$ sign on the right hand side. If this was a $=$ sign (i.e. no ϵ), the solution could be recovered using Gaussian elimination in polynomial time. In the LWE error problem the perturbation ϵ is not a fixed quantity but rather a variable under distribution (e.g. normally Gaussian distribution), and as such cannot be removed to solve through Gaussian elimination.

A subset of this problem, the LWE search to decision reduction problem, can be formally defined by first defining the parameters: n, m, q and $\mathcal{X}$. The former two parameters can be recalled from the Short Integer Solution problem ($m \in \mathbb{R}^+, n \in \mathbb{Z}^+$). The parameter $q \in Z^+$ is a prime number, $\mathcal{X}$ is a probability function (usually a discrete Gaussian) from which the perturbation is chosen $\epsilon \leftarrow \mathcal{X}$.

A pair (A,v) is defined, where $A \in Z_q^{m \times n}$ is chosen uniformly. $v = As + e$, where $s \in Z_q^n$ is the secret vector. As per the LWE search to decision claim, v is indistinguishable from A, even to an adversary that knows A.

This LWE problem is believed to be very hard, with current best performing algorithms running in exponential time in n [36] and no known polynomial quantum computer algorithm. For these reasons, this problem is suitable to be used in public key cryptography applications.

2.9.1. Ring-LWE

Ring-LWE was first introduced by Peikert and Regev as an adaptation of the original LWE problem [26]. The key difference between LWE and Ring-LWE is the use of a polynomial ring for a finite field for variables A and s, as formally defined below:

A pair (A,v) is defined, where $A \in R_q$ is chosen uniformly. $v = As + e$, where $s \in R_q$ is the secret vector. As per the LWE search to decision conjecture, v is indistinguishable from A, even to an adversary that knows A.

A and s are polynomials in R_q, v and e are also polynomials in R_q for Ring-LWE. $R_q = Z_q[X]/X^n + 1$ where n is a power of 2 [24]. [34]. Similar to LWE, the above Ring-LWE is believed to be NP-Hard [22].

The biggest advantage of Ring-LWE public key cryptography systems over similar LWE systems is the reduction in key size, as Ring-LWE key sizes can be reduced to be the square root of LWE key sizes. To achieve AES-128 bits of security, an LWE algorithm would require a key size of 49 million bits whereas a Ring-LWE algorithm would only require a key size of 7000 bits [26].

2.9.2. Module-LWE

Module-LWE can be thought of as a generalisation of Ring-LWE [24], as seen in Figure 4. A new parameter is introduced, d. The problem is formally defined below:

A and s are polynomials in R_q^d, v and e are polynomials in R_q. $R_q = Z_q[X]/X^n + 1$ where n is a power of 2 [24]. [34]. Similar to LWE, the above Ring-LWE is believed to be NP-Hard [22].

Intuitively, the Module-LWE problem can be seen as the Ring-LWE problem with the single ring elements replaced with module elements over the same ring [4]. By setting $d = 1$ (i.e. module rank 1), the original Ring-LWE problem is recovered.

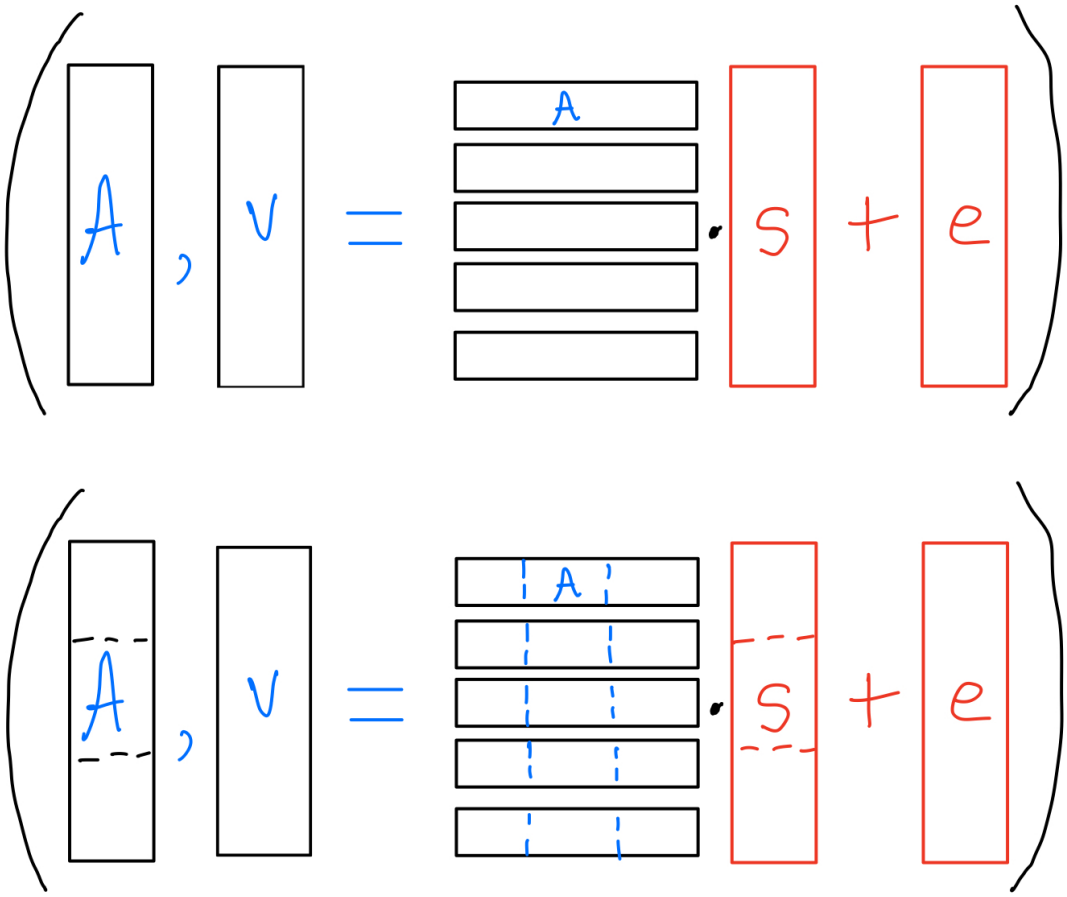

Figure 4: Module LWE and Ring LWE comparison, with d = 3. Drawing adapted from [24]

2.10. CRYSTAL Kyber CPAPKE Algorithm

The CRYSTAL Kyber CPAPKE algorithm builds upon the LPR encryption by Lyubasbevsky, Peikert and Regev [27]. The main difference between the Kyber algorithm and the LPR algorithm is the use of Module-LWE instead of Ring-LWE. There is also a slight modification of the Module LWE problem, with the dimensionality of the variables. A is a matrix with dimensions $R_q^{k \times k}$. v, s, e all have dimension R_q^k. The underlying problem (distinguishing A and v) remains the same.

The Kyber algorithm comprises of 6 individual algorithms. The first 3 algorithms are shared with the Kyber CPA-KEM algorithm. Research and developing an understanding remains to be done on the first 3 algorithms, these will be explained in the Final Report.

Where NTT has been referenced, this refers to Number-Theoretic Transform (NTT), which is an efficient way of performing fast multiplications [7].

2.10.1. Key Generation

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

Output: Secret key $sk \in \mathcal{B}^{12 \cdot k \cdot n/8}$
Output: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8+32}$
$d \leftarrow \mathcal{B}^{32}$
$(\rho, \sigma) := \mathrm{G}(d)$
$N := 0$
for i from 0 to $k-1$ **do** ▷ Generate matrix $\hat{\mathbf{A}} \in R_q^{k \times k}$ in NTT domain
  for j from 0 to $k-1$ **do**
    $\hat{\mathbf{A}}[i][j] := \mathsf{Parse}(\mathsf{XOF}(\rho, j, i))$
  end for
end for
for i from 0 to $k-1$ **do** ▷ Sample $\mathbf{s} \in R_q^k$ from B_{η_1}
  $\mathbf{s}[i] := \mathsf{CBD}_{\eta_1}(\mathsf{PRF}(\sigma, N))$
  $N := N + 1$
end for
for i from 0 to $k-1$ **do** ▷ Sample $\mathbf{e} \in R_q^k$ from B_{η_1}
  $\mathbf{e}[i] := \mathsf{CBD}_{\eta_1}(\mathsf{PRF}(\sigma, N))$
  $N := N + 1$
end for
$\hat{\mathbf{s}} := \mathsf{NTT}(\mathbf{s})$
$\hat{\mathbf{e}} := \mathsf{NTT}(\mathbf{e})$
$\hat{\mathbf{t}} := \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
$pk := (\mathsf{Encode}_{12}(\hat{\mathbf{t}} \bmod^+ q) \| \rho)$ ▷ $pk := \mathbf{As} + \mathbf{e}$
$sk := \mathsf{Encode}_{12}(\hat{\mathbf{s}} \bmod^+ q)$ ▷ $sk := \mathbf{s}$
return (pk, sk)

Figure 5: Kyber Key Generation Algorithm [7].

The key generation algorithm has a similar output to the Module-LWE problem described earlier. Here, the matrix A is sampled from the NTT domain.

A is a uniformly random polynomial in $R_q^{k \times k}$ as there is a one to one uniform mapping. s and e are generated using the Central Binomial Distribution.

Finally, the output is a key pair of $(s, As + e)$ where s is the private key and $As + e$ is the public key. This is a slight adaptation of the LWE problem, where A was the other part of the tuple, however, this does not change anything in the assumptions or the security of the problem. The private and public keys are encoded before returning the function to better support storage and transmission respectively.

2.10.2. Encryption

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

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

$N \coloneqq 0$
$\hat{\mathbf{t}} \coloneqq \mathsf{Decode}_{12}(pk)$
$\rho \coloneqq pk + 12 \cdot k \cdot n/8$
for i from 0 to $k-1$ **do** ▷ Generate matrix $\hat{\mathbf{A}} \in R_q^{k \times k}$ in NTT domain
  for j from 0 to $k-1$ **do**
    $\hat{\mathbf{A}}^T[i][j] \coloneqq \mathsf{Parse}(\mathsf{XOF}(\rho, i, j))$
  end for
end for
for i from 0 to $k-1$ **do** ▷ Sample $\mathbf{r} \in R_q^k$ from B_{η_1}
  $\mathbf{r}[i] \coloneqq \mathsf{CBD}_{\eta_1}(\mathsf{PRF}(r, N))$
  $N \coloneqq N + 1$
end for
for i from 0 to $k-1$ **do** ▷ Sample $\mathbf{e}_1 \in R_q^k$ from B_{η_2}
  $\mathbf{e}_1[i] \coloneqq \mathsf{CBD}_{\eta_2}(\mathsf{PRF}(r, N))$
  $N \coloneqq N + 1$
end for
$e_2 \coloneqq \mathsf{CBD}_{\eta_2}(\mathsf{PRF}(r, N))$ ▷ Sample $e_2 \in R_q$ from B_{η_2}
$\hat{\mathbf{r}} \coloneqq \mathsf{NTT}(\mathbf{r})$
$\mathbf{u} \coloneqq \mathsf{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$ ▷ $\mathbf{u} \coloneqq \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$
$v \coloneqq \mathsf{NTT}^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + \mathsf{Decompress}_q(\mathsf{Decode}_1(m), 1)$ ▷ $v \coloneqq \mathbf{t}^T\mathbf{r} + e_2 + \mathsf{Decompress}_q(m, 1)$
$c_1 \coloneqq \mathsf{Encode}_{d_u}(\mathsf{Compress}_q(\mathbf{u}, d_u))$
$c_2 \coloneqq \mathsf{Encode}_{d_v}(\mathsf{Compress}_q(v, d_v))$
return $c = (c_1 \| c_2)$ ▷ $c \coloneqq (\mathsf{Compress}_q(\mathbf{u}, d_u), \mathsf{Compress}_q(v, d_v))$

Figure 6: Kyber Encryption Algorithm [7].

The public key generated, $As + e_1$ is used to encrypt the plaintext to be sent. The Kyber CPAPKE algorithm supports encryption of fixed 32 bytes in length at a time. The matrix $A \in R^{k \times k}$ is generated in the NTT domain.

e_1 and e_2 are generated using the Central Binomial Distribution to generate values for the perturbations. The output generated is (u, v)

$v = t^T r + e_2 + (q/2)m$

$v = (s^T A^T + e_1^T)r + e_2 + (q/2)m$

Therefore the output can be summarised as a tuple of: $(As + e_1, s^T A^T r + e_1^T r + e_2 + (q/2)m)$ [7] [24].

2.10.3. Decryption

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

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

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

Figure 7: Kyber Decryption Algorithm [7].

The decryption algorithm reverses the process using the secret key generated in Algorithm 4. The output of the function is $v - s^T u$.

It can be shown that this output is the original message:

$v - s^T u = s^T A^T r + e_1^T r + e_2 + (q/2)m - s^T A^T r - s^T e_1$ (by recalling the definition of v from the encryption algorithm)

$v - s^T u = e_1^T r + e_2 - s^T e_1 + (q/2)m$

$v - s^T u \approx (q/2)m$, thus recovering the original message.

3. Implementation

The scope of the project can be split into 4 phases, as detailed in subsections below. The outcome of each task from each Phase forms a milestone.

3.1. Phase A: Research and Understanding

This section includes the following:

1. Research and understanding of encryption and current cryptography algorithms.
2. Understanding of the mathematical basics behind lattices and lattice-based cryptography.
3. Research into lattice-based cryptography and choice of the best candidate.
4. Understanding of the chosen lattice-based cryptography algorithm.

Task	Milestone	Start Date	End Date
1	Report Section (2.1 to 2.4)	19/10/20	16/11/20
2	Report Section (2.5 to 2.7)	16/11/20	14/12/20
3	Report Section (2.8 to 2.9)	14/12/20	04/01/21
4	Report Section (2.10)	04/01/21	08/02/21

Table 2: Phase A task split

For each section, the milestone will be to write report sections around the topic. The first three points from this phase have been covered in this report, and have been explained in the 'Background' section. The final point is partially complete, as evidenced by the end date.

Work completed to date has been on schedule. Some parts highlighted had a larger time-span allocated to account for any exam/coursework clashes.

3.2. Phase B: Implementation of Algorithm

This section includes the following:

1. Implementation of Key Generation Algorithm
2. Implementation of Encryption Algorithm
3. Implementation of Decryption Algorithm
4. End-to-end testing

Task	Milestone	Start Date	End Date
1	Unit tests and Report	08/02/21	01/03/21
2	Unit tests and Report	01/03/21	15/03/21
3	Unit tests and Report	15/03/21	29/03/21
4	Full testing and Report	29/03/21	12/04/21

Table 3: Phase B task split

This phase consists of the bulk of the project. The Kyber algorithm has been split into 3 smaller algorithms: key generation, encryption and decryption to break down the tasks. Whilst these algorithms accept inputs from one another, the functionality implemented is independent and thus can be classified as separate tasks.

The unit testing for these 3 will be done by comparing the algorithm to the official Kyber implementation in C. The report section will include results of these unit tests, as well as a description of how the algorithm operates and the process of programming the algorithm in Python. The main milestones for the first three points will be this comparison and unit testing.

The last task in this phase will be testing the entire algorithm from start to finish. Similarly, this output will be compared to the official implementation.

This phase is initially expected to take place during Spring term, although this may go beyond the budgeted time-frame due to coursework clashes. This will be budgeted for in the remaining 2 sections to ensure a smooth delivery.

3.3. Phase C: Performance Testing

This section includes the following:

1. Performance testing with existing RSA and Diffie-Hellman libraries
2. Performance testing with self developed RSA/Diffie-Hellman algorithm.
3. Performance testing with other lattice-based cryptography solutions

Task	Milestone	Start Date	End Date
1	Data and Report	29/03/21	05/04/21
2	Data and Report	05/04/21	12/04/21
3	Data and Report	12/04/21	26/04/21

Table 4: Phase C task split

This phase primarily consists of comparing the code written to other solutions, both current and quantum-safe encryption models. This phase will also serve as a time for gathering information and data on how to further optimise the program to be used in Phase D based on concepts used by other encryption algorithms. This Phase may also reveal optimisations that can be done using Python, which may be implemented in Phase D.

The main milestones will be to gather this data and document the findings in the Final Report, along with potential reasons for the difference in performance.

It should be noted that extra time has been allocated to this stage due to exam commitments.

3.4. Phase D: Algorithm Improvements

This phase has not been defined precisely yet, as the scope of potential improvements to be made will only be discovered after completion of the Implementation (Phase B). It should be noted that all tasks have the same start and end date for this reason.

Below are some ideas for potential improvement areas.

1. Optimisations in the mathematics of the algorithm
2. External optimisations (e.g. Error Correction Coding)
3. Python based optimisations (e.g. use of different data structures, reduction in complexity etc...)

Task	Milestone	Start Date	End Date
1	Tests and Report	26/04/21	31/05/21
2	Tests and Report	26/04/21	31/05/21
3	Tests and Report	26/04/21	31/05/21

Table 5: Phase D task split

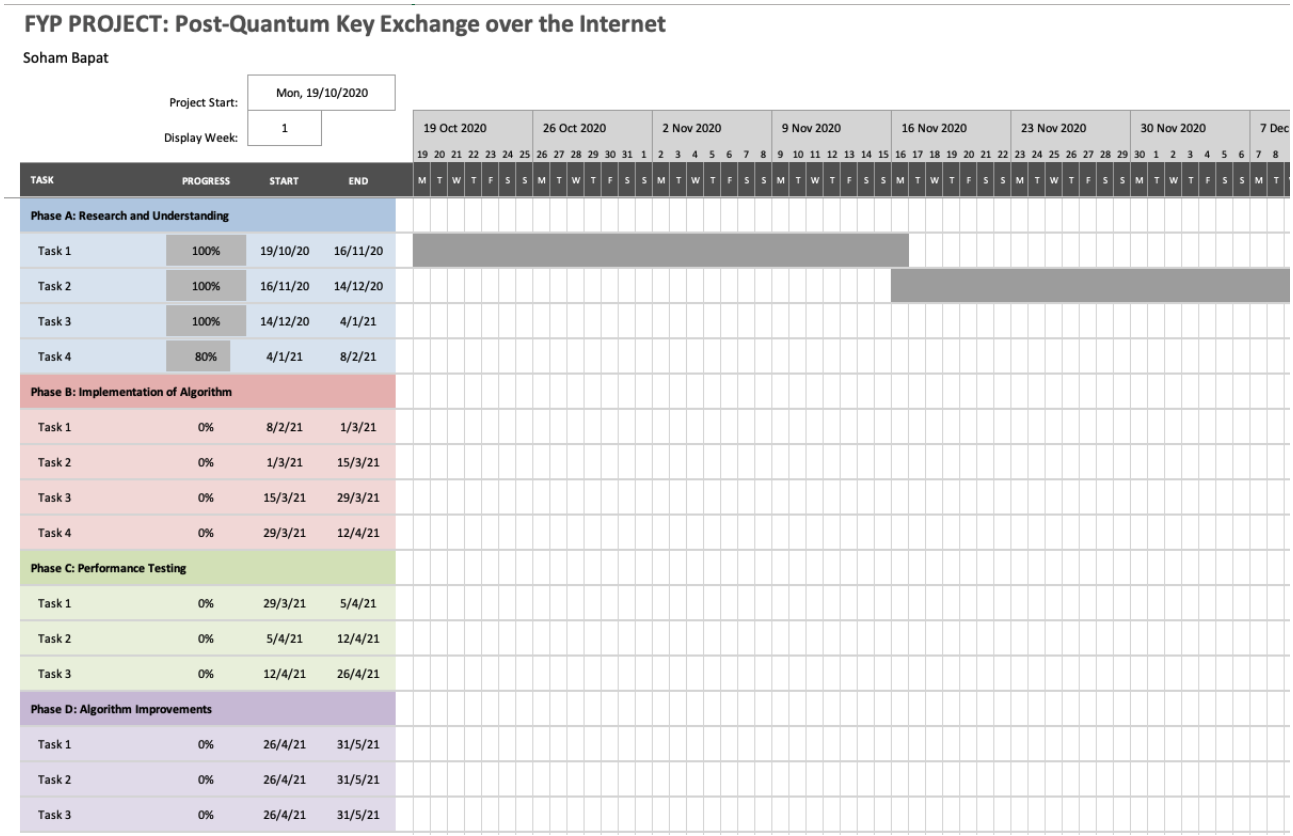

Figure 8: Gannt Chart excerpt. Full Gannt Chart can be found in the GitHub Repo [8].

4. Evaluation

This section details how the success of each Phase will be measured, including backups in case a task/phase is not able to be completed.

4.1. Phase A: Research and Understanding

The success of this phase will be measured by the quality of the report sections. The report will be the best indication of the understanding developed during this phase. The sections were also sent to appropriate people (supervisors, professors etc...) throughout the duration of this phase to ensure information was accurate and relevant.

A backup plan has not been included for this section, as it is almost completed and no issues can be foreseen in the remaining section.

4.2. Phase B: Implementation of Algorithm

The success of this phase will be the output of the milestones (unit tests and end to end tests). This phase can be regarded as a complete success if the program passes all unit and end to end tests by delivering the same results as the official Kyber algorithm.

The main risk of this phase will be the feasibility of implementing the solution in Python, i.e. whether it would take a deeper understanding of the mathematics or if certain capabilities are not possible in Python. If such a situation arises, the backup plan will be to switch to another programming language or to implement another lattice-based cryptography algorithm (such as NTRU and Saber, the other two NIST Round 3 candidates [32]).

4.3. Phase C: Performance Testing

The success of this phase will be the data gathered during the testing of the algorithm, and the final report detailing the findings. This phase can be regarded as a complete success if accurate data is gathered for a wide range of encryption algorithms under different use cases, and the report provides reasoning on the results.

The main risk of this phase is the feasibility of finding a lattice-based cryptography algorithm to test the Kyber code alongside with. If such an algorithm cannot be found, then the backup for this will be to test with a quantum-safe protocol not based around lattices.

4.4. Phase D: Algorithm Improvements

The success of this phase will be the updated figures from the updated algorithm, e.g. percentage speed improvements or improvements in the decryption failure rate compared to the same figures achieved in Phase B. The exact numbers haven't been quantified, as Phase B will indicate the scope of improvements that may be possible.

The main risk of this phase is the possibility of finding improvements to be made. As this is an algorithm well researched by mathematicians worldwide (and one published on an international level), the chances of finding improvements in the mathematics used are incredibly slim. As such, the backup plans are improvements in other areas (as defined by bullet points 2 and 3 in Section 3.4).

If a potential improvement isn't found between all 3 areas, then a discussion meeting will be held with the project supervisors and markers. One potential idea is to create a GUI (Graphical User Interface) tool to allow users to set parameters of Kyber. This, in turn, would help others experiment with Kyber and improve their understanding of how the encryption algorithm works.

5. Miscellaneous

5.1. Legal, Ethical and Safety

This project poses no major safety risk, as this is a software only project. Appropriate care will be taken to prevent strains or injuries during desk work, following guidelines from NHS and Imperial College London.

Legally this project poses no risk. As this code is to be published online (open source), it may be possible that this algorithm/code may be used to facilitate illegal practices. However, this would have no relation to the author of the code (myself), as is similarly true for other open-source code.

Ethically, however, one could argue whether this code may make it easier for a malicious actor to hide illegal activity online. However, the same argument may be applied to current encryption algorithms. Furthermore, this code is only intended to be a proof of concept and not for direct commercial use (without modification and further detailed testing).

5.2. People

One of the major challenges of this project is the dynamic nature of the situation. lattice-based cryptography and post quantum research is not only an area which isn't understood fully yet but is also one which is developing rapidly with breakthroughs every year. As such, there are limited resources available to understand some topics (in particular the mathematical concepts). During a conversation during the project supervisor and primary marker, it was understood that it could take years to understand some of the in-depth mathematical concepts associated with the project. As such, the mathematical overview will be high level throughout this project.

One solution to alleviate some of these difficulties is to contact people who are actively researching or developing in this area. Efforts have been made to contact the Kyber team to better understand the algorithm and ask questions. Additionally, efforts have also been made to contact professors in post-quantum research and former students that have researched this topic.

5.3. Organisation

All code will be published online on GitHub [8], including any intermediary code used to develop understanding/results. At the end of the project, the GitHub repository will be published online for anyone to access.

Going forward, emails will be sent detailing major progress to the supervisor (David) and primary marker (Dr. Ling) every 2-3 weeks in the Spring term, with more frequent updates (1 week) targeted during the Summer term. Meetings will be held as and when required.

6. Appendix

6.1. Programming code

```python
import random
import sympy
#Importing maths libraries for generating random numbers and prime checking

n = 15 #Number to be factorised


attempts_required = 0

while True:
    attempts_required += 1
    x = random.randint(2,n-2) #Initialises a random variable where 1 < x < n-1

    range_register = [i for i in range(n+1)] #Creates an array of numbers from 0 to n inclusive
    remainder_register = []

    for num in range_register:
        remainder_register.append(x**num % n)

    print(range_register)
    print(remainder_register)

    sequence_length_len = len(set(remainder_register)) #Gets the length of the unique sequence. e.g. [1, 2, 4, 1, 2, 4] returns 3.

    possible_factor = x^(sequence_length_len//2) - 1
    complimentary_factor = int(n / possible_factor)

    if n % possible_factor != 0:
        continue
    elif sympy.isprime(possible_factor) and sympy.isprime(complimentary_factor): #Exit condition, when we find two primes that multiply to give n
        break


print('First prime: ', possible_factor)
print('Second prime: ', complimentary_factor)
print('Attempts required: ', attempts_required)
```

Figure 9: Simplified Shor's Algorithm implementation in Python on a classical computer [8]

6.2. Lattice-based cryptography

This section includes further details on lattice-based cryptography which were omitted from the main report.

6.2.1. Ajtai-Dwork

To transmit the message, the plaintext is encrypted bit by bit (0s and 1s). If 0 is to be encrypted, the public key is used to find a random vector near one of the hyperplanes. If 1 is to be encrypted, a random vector is chosen from dimension $\mathbb{R}^n$.

To decrypt the ciphertext, the private key is used to calculate the distance of the point to the nearest hyperplane. If the distance is small then it is decrypted as a 0 else it is decrypted as a 1.

Due to the random nature of choosing 1 from the $\mathbb{R}^n$ space, there is a possibility that the random point chosen may be close to a hyperplane. The probability of this error occurring in decryption is small but polynomial [3]. Further work from Goldreich followed to make modifications such that the decryption was made error-free [19].

6.2.2. GGH

The fundamental idea behind GGH is that the private key forms a good basis of a lattice L whilst the public key forms a bad basis of the same lattice L. A good lattice is defined to be one which consists of relatively short and mostly orthogonal vectors, whilst a bad lattice is one with relatively long and non-orthogonal vectors. The keys can be generated by first defining the private key, then applying a transformation to form the public key. The sender encrypts the plaintext using the public key to choose a random lattice point ($v \in L$)and adds a small perturbation (ϵ). The resultant ciphertext (c) is the addition of the two, $c = v + \epsilon$. The value of ϵ must be small enough such the resultant point is closer to v than to any other lattice point.

The receiver then uses the good basis to retrieve the plaintext. This concept of applying a small perturbation is observed in many modern lattice-based cryptography algorithms such as Learning with Errors.

References

[1] Basim Najim Al-Din Ahmad M. Manasrah. *Security and Communication Networks*. Volume 9 Issue 11. Wiley, 2016, pp. 1450–1461.

[2] M. Ajtai. "Generating Hard Instances of Lattice Problems (Extended Abstract)". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 1996, pp. 99–108. ISBN: 0897917855. DOI: 10.1145/237814.237838. URL: https://doi.org/10.1145/237814.237838.

[3] Mikolas Ajtai and Cynthia Dwork. *A Public Key Cryptosystem with Worst Case Average Case Equivalence*. 1997.

[4] Martin R. Albrecht and Amit Deo. *Large Modulus Ring-LWE Module-LWE*. 2020. URL: https://eprint.iacr.org/2017/612.pdf.

[5] Mirko Amico. *An Experimental Study of Shor's Factoring Algorithm on IBM Q*. 2019. URL: https://arxiv.org/pdf/1903.00768.pdf.

[6] Ghada Arfaoui et al. "The privacy of the TLS 1.3 protocol". In: vol. 2019. July 2019. DOI: 10.2478/popets-2019-0065.

[7] Roberto Avanzi*. *CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation version 3.0*. 2020.

[8] Soham Bapat. *FYP Project: Post Quantum Encryption*. 2021. URL: https://github.com/ImperialCollegeLondon/FYP-Project--Post-Quantum-Encryption.

[9] St´ephane Beauregard. *Circuit for Shor's algorithm using 2n+3 qubits*. 2003. URL: https://arxiv.org/pdf/quant-ph/0205095.pdf.

[10] Nina Bindel et al. *Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange*. Cryptology ePrint Archive, Report 2018/903. 2018. URL: https://eprint.iacr.org/2018/903.

[11] F. Boudot et al. *Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment*. Cryptology ePrint Archive, Report 2020/697. 2020. URL: https://eprint.iacr.org/2020/697.

[12] Daniel R. L. Brown. *Breaking RSA May Be As Difficult As Factoring*. 2008. URL: https://eprint.iacr.org/2005/380.pdf.

[13] Brubaker. *The Very Basics of Groups, Rings, and Fields*. Date accessed: 27/01/21. URL: https://www-users.math.umn.edu/~brubaker/docs/152/152groups.pdf.

[14] Lily Chen. *Report on Post-Quantum Cryptography*. 2016. URL: https://csrc.nist.gov/publications/detail/nistir/8105/final.

[15] Lily Chen. *Shor's algorithm*. Accessed 26/01/21. URL: https://riliu.math.ncsu.edu/437/notes3se4.html.

[16] Ueli Maurer Divesh Aggarwal. *Breaking RSA Generically is Equivalent to Factoring*. 2008. URL: https://eprint.iacr.org/2008/260.pdf.

[17] Jay Gambetta. *IBM's Roadmap For Scaling Quantum Technology*. 2020. URL: https://www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap/.

[18] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. "Collision-Free Hashing from Lattice Problems". In: *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation: In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*. Ed. by Oded Goldreich. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 30–39. ISBN: 978-3-642-22670-0. DOI: 10.1007/978-3-642-22670-0_5. URL: https://doi.org/10.1007/978-3-642-22670-0_5.

[19] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. "Eliminating decryption errors in the Ajtai-Dwork Cryptosystem". In: *Advances in Cryptology — CRYPTO '97*. Ed. by Burton S. Kaliski. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 105–111. ISBN: 978-3-540-69528-8.

[20] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. "Public-Key Cryptosystems from Lattice Reduction Problems". In: *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '97. Berlin, Heidelberg: Springer-Verlag, 1997, pp. 112–131. ISBN: 3540633847.

[21] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. "NTRU: A ring-based public key cryptosystem". In: *Algorithmic Number Theory*. Ed. by Joe P. Buhler. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 267–288. ISBN: 978-3-540-69113-6.

[22] Hoffstein, Pipher, and Silverman. *An Introduction to Mathematical Cryptography*. Springer, 2008. ISBN: 9780387779942.

[23] IBM. *Shor's algorithm*. Accessed on 26/01/21. URL: https://quantum-computing.ibm.com/docs/iqx/guide/shors-algorithm.

[24] David Joseph. *CRYSTALS-Kyber: A Module-LWE KEM*. 2021.

[25] Thorsten Kleinjung. *Factorization of a 768-bit RSA modulus*. 2010. URL: https://eprint.iacr.org/2010/006.pdf.

[26] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. *On Ideal Lattices and Learning with Errors Over Rings*. Cryptology ePrint Archive, Report 2012/230. 2012. URL: https://eprint.iacr.org/2012/230.

[27] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On Ideal Lattices and Learning with Errors over Rings". In: vol. 60. May 2010, pp. 1–23. ISBN: 978-3-642-13189-9.

[28] Vasileios Mavroeidis. *The Impact of Quantum Computing on Present Cryptography*. 2018. URL: https://arxiv.org/pdf/1804.00200.pdf.

[29] Daniele Micciancio. *Inapproximability of the Shortest Vector Problem: Toward a Deterministic Reduction*. 2012. URL: http://www.theoryofcomputing.org/articles/v008a022/v008a022.pdf.

[30] Daniele Micciancio. *Lattice based cryptography*. 2008. URL: https://cims.nyu.edu/~regev/papers/pqc.pdf.

[31] Daniele Micciancio and Oded Regev. "Lattice-based Cryptography". In: *Post-Quantum Cryptography*. Ed. by Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 147–191. ISBN: 978-3-540-88702-7. DOI: 10.1007/978-3-540-88702-7_5. URL: https://doi.org/10.1007/978-3-540-88702-7_5.

[32] NIST. *Post-Quantum Cryptography*. 2017. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals.

[33] B. Padmavathi. *A Survey on Performance Analysis of DES, AES and RSA Algorithm along with LSB Substitution Technique*. 2013. URL: https://www.ijsr.net/archive/v2i4/IJSRON120134.pdf.

[34] Chris Peikert1. *A Decade of Lattice Cryptography*. 2016. URL: https://web.eecs.umich.edu/~cpeikert/pubs/lattice-survey.pdf.

[35] H. A. Priestley. *Introduction to Groups, Rings and Fields*. Date accessed: 27/01/21. URL: https://people.maths.ox.ac.uk/flynn/genus2/sheets0405/grfnotes1011.pdf.

[36] Oded 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: Association for Computing Machinery, 2005, pp. 84–93. ISBN: 1581139608. DOI: 10.1145/1060590.1060603. URL: https://doi.org/10.1145/1060590.1060603.

[37] Oded Regev. *The Learning with Errors Problem*. Date accessed: 27/01/21. URL: https://cims.nyu.edu/~regev/papers/lwesurvey.pdf.

[38] Peter Schwabe. *CRYSTALS Cryptographic Suite for Algebraic Lattices*. Date accessed: 31/01/21. URL: https://pq-crystals.org/kyber/resources.shtml.

[39] Emily Shenfield. *Transport Layer Security (TLS)*. 2019. URL: https://www.twilio.com/docs/glossary/what-is-transport-layer-security-tls.

[40] Peter Shor. *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*. 1996. URL: https://arxiv.org/pdf/quant-ph/9508027.pdf.

[41] Noah Stephens-Davidowitz. *Search-to-Decision Reductions for Lattice Problems with Approximation Factors (Slightly) Greater Than One*. 2012=7. URL: https://arxiv.org/pdf/1512.04138.pdf.

[42] Xin Zhou and Xiaofei Tang. "Research and implementation of RSA algorithm for encryption and decryption". In: *Proceedings of 2011 6th International Forum on Strategic Technology*. Vol. 2. 2011, pp. 1118–1121. DOI: 10.1109/IFOST.2011.6021216.