

# Master's Thesis

## Comparative Complexity Analysis of ML-KEM & HQC

Asymptotic Time, Space, & Bit-Level Evaluation of Post-Quantum Cryptographic Key Encapsulation Mechanisms



**Stine Byrjalsen & Julie Timmermann Werge**

Aalborg University

Mathematical Engineering

Copyright © Aalborg University 2025

This report is written in the typesetting language  $\text{\LaTeX}$  using the text editor Overleaf. Unless explicitly noted by a citation, illustrations are created using the TikZ package in  $\text{\LaTeX}$  or Python 3.11.7. References are cited using the IEEE method.



**AALBORG UNIVERSITY**  
STUDENT REPORT

Department of Mathematical Sciences  
Thomas Manns Vej 23  
DK - 9220 Aalborg  
<http://www.math.aau.dk>

**Title:**

Comparative Complexity Analysis of ML-KEM & HQC

**Subtitle:**

Asymptotic Time, Space, & Bit-Level Evaluation of Post-Quantum Cryptographic Key Encapsulation Mechanisms

**Theme:**

Master's Thesis

**Project Period:**

Spring Semester 2025

**Project Group:**

MT10-03

**Participants:**

Stine Byrjalsen

Julie Timmermann Werge

**Supervisors:**

Martin Voigt Vejling

Kristian Skafte Jensen

**RTX A/S Supervisor:**

Jens Christian Lindof

**Page Numbers:** 92

**Date of Completion:**

May 27, 2025

**Abstract:**

The aim of this master's thesis is to compare the time, space, and bit complexities of the post-quantum cryptographic schemes ML-KEM and HQC, with a focus on their suitability for implementation in resource-constrained devices. This comparative analysis is based on analytically derived asymptotic time and space complexities, as well as bit complexities, for both schemes. The complexities are evaluated across three security categories, using different input parameters as variables. Results show that ML-KEM demonstrates a slower and more predictable growth rate across all metrics. In contrast, HQC exhibits steep increases, particularly in decapsulation time, which scales quadratically and results in operation counts several orders of magnitude higher than ML-KEM. Furthermore, HQC produces significantly larger ciphertexts, posing challenges for devices operating over bandwidth-limited communication channels. HQC's memory requirements are two to four orders of magnitude greater due to larger parameter sizes. Thus, the conclusion is that ML-KEM offers more favourable complexity scaling, making it better suited for latency-sensitive, memory-limited, and bandwidth-constrained devices.

# Preface

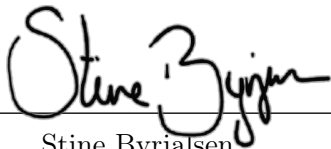
This master's thesis is written by Stine Byrjalsen and Julie Timmermann Werge during the 10th semester of studies in Mathematical Engineering at the Department of Mathematical Sciences at Aalborg University. The project is written from February 3 to May 27, 2025. The project is carried out in collaboration with RTX A/S and focuses on the topic of post-quantum cryptography.

We would like to express our sincere gratitude to our academic supervisors, Martin Voigt Vejling and Kristian Skafte Jensen, for their valuable guidance and feedback throughout the project. We also wish to thank our company supervisor at RTX, Jens Christian Lindof, for his dedicated support and insights. A special thanks goes to René Bødker Christensen for his helpful input and advice during the process.

We would like to thank RTX A/S as a whole for hosting us during the project period and for providing a welcoming and inspiring environment. In particular, we appreciate the opportunity to work from the CTO office, which gave us insight into the company's research-oriented mindset and fostered valuable professional interactions.

Finally, we wish to thank Caroline Ulstrup for designing the cover page of this thesis.

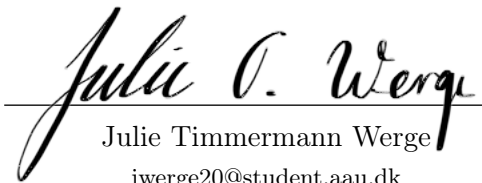
Aalborg University, May 27, 2025



---

Stine Byrjalsen

sbyrja20@student.aau.dk



---

Julie Timmermann Werge

jwerge20@student.aau.dk

# Nomenclature

Acronym	Meaning
RSA	Rivest-Shamir-Adleman
ECC	Elliptic curve cryptography
PQC	Post-quantum cryptography
NIST	National Institute of Standards and Technology
KEM	Key encapsulation mechanism
FFT	Fast Fourier transform
AES	Advanced encryption standard
LWE	Learning with errors
FIPS	Federal Information Processing Standards
PKE	Public-key encryption
CPA	Chosen plaintext attack
CCA1	Chosen ciphertext attack
CCA2	Adaptive chosen ciphertext attack
IND	Indistinguishability
ML-KEM	Module-lattice-based key encapsulation mechanism
MLWE	Module learning with errors
FO	Fujisaki-Okamoto
NTT	Number-theoretic transform
QC	Quasi-cyclic
QCSD	Quasi-cyclic syndrome decoding
DQCSD	Decisional quasi-cyclic syndrome decoding
RM	Reed-Muller
RS	Reed-Solomon
RMRS	Reed-Muller and Reed-Solomon

Notation	Description
$\mathbf{v}$	Vector
$\mathbf{A}$	Matrix
$\text{mod}$	Modulo operation
$\equiv$	Congruence relation
$\cong$	Isomorphic to
$\mathbb{Z}_q$	Ring of integers modulo $q$
$\mathcal{L}$	Lattice
$\ \cdot\ _2$	The Euclidean norm
$\mathbb{R}_{>0}$	Set of positive real numbers
$\lceil x \rceil$	The smallest integer greater than or equal to $x$
$\lfloor x \rfloor$	The largest integer less than or equal to $x$
$\lceil x \rceil$	The rounding of $x$ to the nearest integer with ties being rounded up
$\{0, 1\}^*$	Binary string of arbitrary finite length
$\{0, 1\}^n$	Binary string of length $n$
$\ \cdot\ _\infty$	The $\ell_\infty$ norm
$B_\eta$	The central binomial distribution
$a_j$	The coefficient of $x^j$ of a polynomial $a = a_0 + a_1x + \cdots + f_{n-1}x^{n-1}$
$\hat{a}$	The NTT representation of a polynomial $a$
$\mathbf{v}^\top, \mathbf{A}^\top$	The transpose of a vector $\mathbf{v}$ or a matrix $\mathbf{A}$
$\oplus$	Direct sum
$\mathbf{0}$	Null vector
$\mathcal{O}$	Big-O

# Contents

<b>1</b>	<b>Problem Analysis</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Fundamental Principles of Classical Cryptography . . . . .	1
1.3	Quantum Computing . . . . .	5
1.4	Post-Quantum Cryptography . . . . .	6
1.5	Resource-Constrained Devices . . . . .	10
1.6	Problem Statement . . . . .	11
1.7	Problem Delimitations . . . . .	11
<b>2</b>	<b>Lattices &amp; Learning With Errors</b>	<b>12</b>
2.1	Prerequisites . . . . .	12
2.2	Lattice Problems . . . . .	16
2.3	Learning With Errors . . . . .	18
<b>3</b>	<b>ML-KEM Standard</b>	<b>20</b>
3.1	Prerequisites . . . . .	20
3.2	K-PKE . . . . .	22
3.3	ML-KEM . . . . .	24
3.4	Parameter Sets . . . . .	26
3.5	Number-Theoretic Transform . . . . .	28
<b>4</b>	<b>Quasi-Cyclic Codes &amp; Syndrome Decoding Problems</b>	<b>32</b>
4.1	Quasi-Cyclic Codes . . . . .	32
4.2	Quasi-Cyclic Syndrome Decoding Problems . . . . .	34
<b>5</b>	<b>Hamming Quasi-Cyclic</b>	<b>38</b>
5.1	Prerequisites . . . . .	38
5.2	HQC.PKE . . . . .	42
5.3	HQC.KEM . . . . .	43
5.4	Parameter Sets . . . . .	45
<b>6</b>	<b>Comparative Complexity Analysis of ML-KEM &amp; HQC</b>	<b>47</b>
6.1	Time Complexity Comparison . . . . .	47
6.2	Space Complexity Comparison . . . . .	51
6.3	Bit Complexity Comparison . . . . .	55
<b>7</b>	<b>Discussion</b>	<b>64</b>
<b>8</b>	<b>Conclusion</b>	<b>66</b>
	<b>Bibliography</b>	<b>67</b>

<b>A</b>	<b>Complexity Analysis Foundations</b>	<b>71</b>
A.1	Big- $\mathcal{O}$ Notation . . . . .	71
A.2	Divide-and-Conquer Algorithms . . . . .	73
<b>B</b>	<b>ML-KEM Complexity Analysis</b>	<b>76</b>
B.1	Time Complexity Analysis of ML-KEM . . . . .	76
B.2	Space Complexity Analysis of ML-KEM . . . . .	79
B.3	Bit Complexity Analysis of ML-KEM . . . . .	81
<b>C</b>	<b>HQC Complexity Analysis</b>	<b>85</b>
C.1	Time Complexity Analysis of HQC . . . . .	85
C.2	Space Complexity Analysis of HQC . . . . .	88
C.3	Bit Complexity Analysis of HQC . . . . .	90

# 1 Problem Analysis

## 1.1 Introduction

In modern technology, public-key cryptography serves as the foundation for securing the global communication infrastructure. However, the rise of quantum computing threatens to break widely used cryptographic protocols. If compromised, it could lead to widespread data breaches, financial fraud, and vulnerabilities in essential infrastructure [1].

Recent years have brought significant breakthroughs in the field of quantum computers [2]. By leveraging the principles of superposition, entanglement, and interference, quantum computers have the potential to solve problems that were once considered insurmountable [2] [3]. However, current quantum computers remain limited by factors such as qubit count and error rates, preventing them from posing an immediate threat to classical encryption [3]. Despite these challenges, researchers anticipate that continued progress in quantum technology will soon lead to the development of more powerful quantum computers [3].

Post-quantum cryptography (PQC) has been introduced to address the security challenges posed by quantum computing. The National Institute of Standards and Technology (NIST) has been working to standardise PQC methods to ensure resilience against quantum attacks. In 2016, NIST launched a search for candidate algorithms suitable for standardisation. After four rounds of submissions and rigorous evaluations based on security, cost, performance, and implementation characteristics, NIST standardised four algorithms in 2024 [4, ch. 2] [5] [6] [7]. These include one key encapsulation mechanism (KEM) and three digital signature schemes. In 2025, NIST additionally standardised Hamming Quasi-Cyclic (HQC), a code-based KEM, as a backup to the module-lattice-based KEM (ML-KEM) in order to provide cryptographic diversity and resilience against potential future advances in cryptanalysis [8].

Despite their security advantages, PQC algorithms demand more computational power and memory than traditional methods, presenting challenges for resource-constrained devices [9]. As quantum-safe cryptography becomes essential, it is crucial to develop efficient implementations that balance security, performance, and power consumption [9]. This thesis aims to introduce and compare PQC algorithms through a complexity analysis, examining how time and memory scale with different parameter sets. This thesis will help determine which algorithm is more suited for devices with varying resource constraints, allowing for an informed choice based on the specific needs of the device. To proceed, a foundational understanding of classical cryptography is necessary.

## 1.2 Fundamental Principles of Classical Cryptography

Secure communication is essential to protect sensitive information from unauthorised access. Cryptography is the study of securing data by transforming it into an unreadable format. This enables two parties to communicate safely over an insecure channel while preventing

## 1.2. Fundamental Principles of Classical Cryptography

access by adversaries [10, p. 15]. The process of transforming an original message, known as plaintext, into a random string of symbols, known as ciphertext, using a key, is called encryption. The key is a predetermined random string of numbers. The plaintext can be any type of data with an arbitrary structure, for example, text or numerical data. The process of turning ciphertext back into plaintext is called decryption. Keys are fundamental to cryptosystems, which define how key generation, encryption, and decryption are performed [10, p. 1].

Cryptography provides four fundamental security properties:

- *Confidentiality* prevents unauthorised access to the information, ensuring that only the intended receiver can gain access.
- *Integrity* ensures that the received information has not been altered in any way from the original.
- *Authentication* is the process of proving one's identity.
- *Non-repudiation* prevents an entity from denying previous commitments or actions.

These properties ensure that information is kept private, unaltered, verifiable, and indisputable. Encryption schemes ensure confidentiality, while digital signature schemes ensure integrity, authentication, and non-repudiation [11, p. 4] [10, p. 7].

A fundamental principle in cryptography is Kerckhoffs' principle, which states that a cryptosystem's security should rely solely on the secrecy of a key, while all other components, including algorithms and protocols, should be publicly known. Thus, when designing a cryptosystem, it should always be assumed that the adversary knows the scheme being used [10, p. 10]. The usage of keys differentiates cryptography into two categories: secret-key and public-key cryptography [11, p. 15].

### Secret-Key Cryptography

Secret-key cryptography, also known as symmetric cryptography, involves the use of a single shared key for both encryption and decryption. In a typical secret-key encryption protocol, two parties first establish a shared secret key, either in person without eavesdroppers or over a secure channel. Once established, the sender can securely transmit plaintexts over an insecure channel by encrypting them using the shared secret key. The recipient then decrypts the received ciphertext using the same shared secret key to recover the plaintext [10, pp. 15–16].

Secret-key cryptosystems use relatively short keys, which require less storage space and allow faster data transmission. They also offer high data throughput, which means they can process and transmit large amounts of data quickly [11, p. 31]. However, they need a secure key exchange before communication can begin. If parties are unable to meet in secret or communicate over a secure channel, exchanging the secret key becomes a challenge [12, p. 46]. The search for a solution to this problem gave rise to public-key cryptography.

### Public-Key Cryptography

Public-key cryptography, also known as asymmetric cryptography, involves the use of a key pair: a public key and a private key. The public key is shared openly, and the private key is kept secret. Knowing the public key reveals nothing about the private key, but only the private key can decrypt ciphertexts encrypted with its corresponding public key [10, p. 2] [12, p. 46]. According to Kerckhoffs' principle, an adversary is assumed to know the public key and the encryption algorithm but not the private key [10, p. 10]. The first example of a public-key cryptosystem was the Rivest-Shamir-Adleman (RSA) cryptosystem. Today, public-key cryptography includes a range of applications, such as public-key encryption (PKE), KEM, and digital signature schemes [10, p. 3].

In PKE, the goal is to ensure confidentiality of a message. The recipient generates a key pair, shares the public key over any unsecured channel, and keeps the private key secure. The sender can then use the recipient's public key to encrypt a message, producing ciphertext that only the recipient can decrypt using their private key. Since the public key is openly available, anyone can encrypt messages, but only the recipient, possessing the corresponding private key, can decrypt them [11, pp. 25–26] [10, p. 185].

A KEM is a variant of PKE. Instead of encrypting the actual message directly, a random secret key is generated and encapsulated using the recipient's public key. The sender produces a ciphertext and a shared secret key, which the recipient can recover by decapsulating the ciphertext with their private key. This shared secret key is then used with a secret-key encryption scheme to encrypt and decrypt the actual message [13, p. 495].

In a signature scheme, the sender generates a key pair, keeps the private key secure, and publishes the public key. The sender signs the message with their private key, producing a unique signature. The recipient can then verify the signature using the sender's public key. Since the public key isn't secret, anyone can use it to verify the signature, but only the sender, possessing the private key, could have generated it [11, p. 29].

An advantage of public-key cryptosystems is that only the private key needs to be kept secret. Additionally, in large networks, the number of keys required may be considerably smaller than in secret-key cryptosystems. However, public-key cryptosystems are generally slower than secret-key cryptosystems, and public keys are typically much larger than those used in secret-key encryption [11, pp. 31–32]. Due to their slower performance, public-key cryptosystems are often combined with secret-key systems [10, p. 3].

### Hybrid Cryptography

Hybrid cryptography combines the advantages of both secret-key and public-key cryptography. When a sender wants to transmit a long plaintext but they do not have a shared secret key, they can generate a random shared secret key and use it to encrypt plaintext with a secret-key cryptosystem. The sender then encrypts the shared secret key using the recipient's public key and sends both the ciphertext and the encrypted secret key to the recipient. The recipient then decrypts the shared secret key with their private key and uses it to decrypt the

## 1.2. Fundamental Principles of Classical Cryptography

ciphertext and retrieve the plaintext. In this method, the slower public-key cryptosystem is only used to encrypt the short secret key, while the much faster secret-key cryptosystem is used to encrypt the longer plaintext. Thus, hybrid cryptography combines the efficiency of secret-key cryptography with the secure key exchange capabilities of public-key cryptography [10, p. 3].

### Cryptanalysis

Cryptanalysis is the study of methods for breaking cryptographic systems, typically with the goal of recovering secret keys or forging valid outputs without authorised access [12, p. 5]. There are various attack models, each specifying the information available to the adversary. For PKE and KEMs, attacks are typically framed within the context of indistinguishability (IND) based security models. Common attack models include:

- *Ciphertext-only attack*, the adversary has access only to ciphertext.
- *Known plaintext attack*, the adversary has access to both plaintext and its corresponding ciphertext.
- *Chosen plaintext attack* (CPA), the adversary can choose a plaintext and obtain its corresponding ciphertext [10, p. 39].
- *Chosen ciphertext attack* (CCA1), the adversary can choose a ciphertext and obtain its corresponding plaintext.
- *Adaptive chosen ciphertext attack* (CCA2), the adversary can query a decryption oracle for any ciphertext [13, p. 32].

IND means the adversary cannot distinguish between the encryptions of two chosen messages of the same length. There are different levels of IND security: IND-CPA, IND-CCA1, and IND-CCA2. IND-CCA2 is the strongest security model, and schemes secure under it are also secure under IND-CPA and IND-CCA1 [13, p. 32].

The goal of an adversary attacking a digital signature scheme is to forge a valid signature on a message that has not been signed by the legitimate signer. Attack models for digital signatures differ in focus and are categorised as follows:

- *Key-only attack*, the adversary only knows the signer's public key.
- *Known message attack*, the adversary has access to a list of valid message-signature pairs but does not control the messages.
- *Chosen message attack*, the adversary can obtain signatures on arbitrary messages of their choosing [10, p. 312].

The success of an attack is further classified by the type of forgery achieved:

- *Existential forgery*, the adversary produces a valid signature for at least one message.
- *Selective forgery*, the adversary forges a signature on a specific, pre-chosen message.

### 1.3. Quantum Computing

- *Universal forgery*, the adversary can forge signatures for any message [10, p. 312].

Classical cryptographic schemes rely on the assumed hardness of mathematical problems, but quantum computing threatens their security and undermines schemes considered secure under classical assumptions. Understanding this impact is essential, as many widely used cryptographic schemes may become vulnerable.

## 1.3 Quantum Computing

Quantum computers operate based on the fundamental principles of quantum mechanics [14, p. 3]. Unlike classical computers, which utilise bits as the smallest unit of data, quantum computers employ qubits. Qubits possess the unique ability to exist in a superposition of states, meaning they can represent 0 and 1 simultaneously, each with a specific probability amplitude [14, pp. 9–10]. This property enables quantum computers to perform certain computations exponentially faster than their classical counterparts by simultaneously exploring multiple computational pathways [14, p. 147].

### Quantum Algorithms Threatening Classical Cryptography

Classical public-key cryptosystems, such as RSA and elliptic curve cryptography (ECC), are widely used today to secure communication. The security of these systems relies on the inherent difficulty of solving specific mathematical problems. In the case of RSA, security is based on the hardness of integer factorisation, while ECC is grounded in the difficulty of solving the discrete logarithm problem over elliptic curves [15]. One of the most significant threats quantum computing poses to classical cryptography arises from Shor’s algorithm [16] [1].

Shor’s algorithm is capable of efficiently factoring large integers and solving the discrete logarithm problem [16] [3]. While classical computers require an impractically long time to solve these problems, quantum computers can perform these calculations exponentially faster due to their unique computational properties [1]. Despite this theoretical capability, implementing Shor’s algorithm on a practical quantum computer capable of breaking RSA and ECC is still a significant challenge. This is due to several challenges, including the need to correct errors, maintain stable quantum states, and scale up to a large number of reliable qubits [1].

Grover’s algorithm is another algorithm posing a threat to classical encryption. Grover’s algorithm offers a quadratic speed-up on brute-force attacks on secret-key encryption schemes, such as advanced encryption standard (AES) [17] [15]. Due to the quadratic speed-up, the key length must be doubled to preserve the security [1].

Although no quantum computer currently exists that is capable of breaking modern cryptographic schemes using Shor’s or Grover’s algorithms, it is widely regarded as only a matter of time before such a breakthrough occurs. In 2016 NIST projected that a quantum computer capable of breaking RSA via Shor’s algorithm could emerge in the 2030s [1].

## 1.4. Post-Quantum Cryptography

Moreover, the "harvest now, decrypt later" strategy presents an additional motivation for the immediate implementation of PQC. Adversaries can collect and store vast amounts of encrypted data today, intending to decrypt it once sufficiently powerful quantum computers become available [18]. This poses a severe threat to long-term confidentiality, particularly for sensitive information such as government communications, medical records, and classified data.

## 1.4 Post-Quantum Cryptography

PQC is the field dedicated to developing cryptographic algorithms that remain secure against both classical and quantum attacks. Furthermore, these algorithms should be compatible with existing communication protocols to ensure a smooth transition in the event of large-scale quantum advancements [1].

As mentioned in the previous section, doubling the key size in secret-key cryptographic algorithms is considered sufficient to mitigate the impact of Grover's algorithm [1]. Thus, the primary focus of PQC is the development of quantum-resistant public-key cryptosystems, as they are the most vulnerable to attacks from Shor's algorithm [1]. To develop quantum-safe algorithms, researchers must explore alternative mathematical problems that are believed to be resistant to quantum attacks. The main categories of PQC algorithms include:

- *Lattice-based cryptography* is based on the hardness of the shortest vector problem and learning with errors (LWE).
- *Code-based cryptography* relies on the difficulty of decoding random linear codes.
- *Multivariate polynomial cryptography* is based on the intractability of solving multivariate quadratic polynomial equations over finite fields.
- *Hash-based signatures* derive security solely from cryptographic hash functions.
- *Isogeny-based cryptography* relies on the difficulty of finding isogenies between elliptic curves.

It is not possible to guarantee absolute security for all PQC schemes against quantum attacks. The only certainty is that, as of now, no efficient quantum algorithm is known to break them [1]. However, new quantum algorithms may be discovered in the future, potentially undermining the security of some proposed PQC schemes. An exception to this uncertainty is hash-based signatures, which rely solely on the well-studied security of cryptographic hash functions and are not susceptible to quantum speed-ups beyond Grover's algorithm [1].

### Post-Quantum Cryptography Standardisation Process

Current cryptographic NIST standards will not remain secure once large-scale quantum computers exist [19]. To address this, NIST intends to standardise new quantum-resistant algorithms that will update or replace parts of the current public-key cryptographic standards.

## 1.4. Post-Quantum Cryptography

Since no direct replacement exists, transitioning to PQC will require significant effort in development, standardisation, and deployment. Therefore, in 2016, NIST initiated a public, competition-like process to select quantum-resistant public-key cryptographic algorithms for digital signatures and KEMs. It is intended that these algorithms will be capable of protecting sensitive information in the post-quantum era. This process is referred to as the NIST PQC standardisation process [19].

To evaluate the candidates in NIST’s PQC Standardisation Process, evaluation criteria have been established. These include security, cost, performance, and algorithm and implementation characteristics [19].

### Security Evaluation Criteria

The security criterion is the most important factor in the evaluation of the candidates and considers both classical and quantum attacks [19]. The criterion includes an assessment of how secure the schemes are in applications of public-key cryptography, particularly in internet protocols where current standards for digital signatures and KEMs are widely used. NIST also defines three security properties: two for KEM and one for digital signatures. For KEM schemes, the two properties ensure that a KEM scheme is IND-CCA2 and IND-CPA. Digital signature schemes should be secure against existential forgery, even when an attacker can adaptively choose which messages to be signed [19]. Furthermore, NIST designated five security strength categories for classifying the complexity of attacks that violate the security properties [19]. The security categories can be seen in Table 1.1.

Category	Security Description	Search Type
1	At least as hard to break as AES128	Exhaustive key search
2	At least as hard to break as SHA256	Collision search
3	At least as hard to break as AES192	Exhaustive key search
4	At least as hard to break as SHA384	Collision search
5	At least as hard to break as AES256	Exhaustive key search

**Table 1.1:** The five security strength categories designated by NIST.

The security strength categories are based on the computational resources required to perform certain brute-force attacks against AES and secure hash algorithms. A cryptosystem meets a security category if any attack requires resources at least as high as the defined threshold. For categories 1, 3, and 5, any attack that breaks the security properties must require computational resources comparable to or greater than those required for an exhaustive key search on a block cipher with a 128-bit key, 192-bit key, or 256-bit key, respectively. For categories 2 and 4, any attack that breaks the security properties must require computational

#### 1.4. Post-Quantum Cryptography

resources comparable to or greater than those required for collision search on a 256-bit hash function and 384-bit hash function, respectively [19].

The categories increase in strength such that a SHA256 collision attack is feasible before an AES192 key search. NIST prioritises categories 1, 2, and 3, as categories 4 and 5 are for very high security. If certain parameters meet the requirements of a higher category, they will automatically satisfy lower categories as well [19].

While security properties and strength categories address many attack scenarios, additional properties are also assessed. These include the following:

- *Perfect forward secrecy* ensures past session keys remain secure even if a server’s private key is compromised.
- *Resistance to side-channel attacks* refers to the ability to protect against attackers who attempt to gain information from physical signals during cryptographic operations.
- *Multi-key security* ensures that compromising one key does not compromise others. Attacking multiple keys at once provides no additional advantage.
- *Misuse resistance* prevents serious failures from coding errors, random number generator malfunctions, nonce reuse, and keypair reuse [19].

Additionally, a thorough understanding of the mathematical structure underlying public-key cryptography is crucial for confidence in a cryptosystem’s security. Simple schemes tend to be better understood than complex ones. Likewise, schemes based on well-established design principles are more reliable than completely new schemes or those that were designed by repeatedly patching older schemes that were shown vulnerable to cryptanalysis [19].

#### **Cost & Performance Evaluation Criteria**

Cost is identified as the second most important criterion [19]. It includes the size of public keys, ciphertexts, and signatures, as well as the computational efficiency of key operations and generation and decryption failures. The size of the public key, ciphertext, and signatures may be important consideration factors for resource-constrained devices. Schemes are assessed on the computational efficiency of public and private key operations both in hardware and software. The computational efficiency of key generation is also evaluated, especially for algorithms providing perfect forward secrecy. Furthermore, decryption failures must be minimised, as some schemes may occasionally produce ciphertexts that cannot be decrypted [19].

#### **Algorithm & Implementation Characteristics Evaluation Criteria**

The algorithm and implementation characteristics criterion relies on flexibility, simplicity, and adoption. Schemes with greater flexibility are preferred, as they can accommodate a wider range of users’ needs. Examples of flexibility may include the customisation of parameters to meet varying security and performance goals, secure and efficient implementation on diverse platforms, such as resource-constrained devices, and seamless integration into existing

#### 1.4. Post-Quantum Cryptography

protocols with minimal changes. Furthermore, the schemes are assessed based on their design simplicity. While it is hard to measure simplicity concretely, simpler designs are preferable when comparing two similar schemes. Moreover, factors affecting widespread adoption of an algorithm, such as patents, copyrights, and terms of licenses, are also considered [19].

#### Standardisation Finalists & Alternates

The first round of the NIST PQC standardisation process began in December 2017 with 82 submissions, 69 of which were accepted as complete and proper. Of these, 19 were digital signature schemes, and 45 were KEM schemes. Algorithms selected for the second round, starting in January 2019, were based on internal analysis and public feedback, with 26 candidates chosen. Of these, 9 were signature schemes, and 17 were KEM schemes. By the end of the second round, 7 finalists and 8 alternates were selected for the third round. The third round, which began in July 2020, led to the first algorithms selected for standardisation. The KEM scheme that was selected for standardisation was CRYSTALS-Kyber, and the digital signatures were CRYSTALS-Dilithium, FALCON, and SPHINCS+, with CRYSTALS-Dilithium being the primary signature algorithm [19]. Furthermore, four alternate candidates were selected to advance to a fourth round for further evaluation and study. The alternate candidates considered were the Classic McEliece, HQC, SIKE, and BIKE [19]. These candidates are all KEMs but rely on different security assumptions than CRYSTALS-Kyber. SIKE was removed early in the fourth round, as it was found to be insecure. Classic McEliece, while considered a conservative choice, suffered from a large public-key size and slow key generation, making it less desirable for many common applications. In March 2025, NIST announced the selection of HQC as the fourth-round finalist, expressing a higher level of confidence in its IND-CCA2 security over BIKE due to HQC’s more mature and stable decryption failure rate analysis [8]. An overview of the finalists can be seen in Table 1.2.

Algorithm Name	Type	Basis	Security Category	Public-Key Size	Ciphertext/Signature Size
CRYSTALS-Kyber	KEM	Lattice	1, 3, 5	Small	Small
CRYSTALS-Dilithium	Signature	Lattice	2, 3, 5	Medium	Medium
FALCON	Signature	Lattice	1, 5	Small	Small
SPHINCS+	Signature	Hash	1, 3, 5	Small	Large
HQC	KEM	Code	1, 3, 5	Medium	Large

**Table 1.2:** Overview of the finalists in the NIST PQC standardisation process. All information regarding the algorithms is sourced from [19] and [20].

## 1.5. Resource-Constrained Devices

NIST plans to standardise algorithms from various PQC categories but prioritises those that are closest to being ready for widespread adoption, rather than attempting to finalise all standards at once [19].

## 1.5 Resource-Constrained Devices

The quantum-resistant cryptographic schemes selected by NIST offer security against quantum attacks, but they require more memory, energy, and power than traditional methods, which can be problematic for resource-constrained devices [21]. Resource-constrained devices are computing systems that have limited processing power, memory, storage, and energy resources. Such devices include Internet of Things devices, embedded systems, and smart cards [21] [22]. Many of these devices are used in industries such as smart homes, medical devices, public infrastructure, and automobiles [9]. Furthermore, resource-constrained devices are prime targets for hackers due to their interaction with the physical environment and sensitive data collection. Common attacks include eavesdropping, replay, node capture, and side-channel attacks [9]. Since resource-constrained devices operate in critical infrastructures and are preferential targets, high security is required [23] [21].

### Trade-Off Between Security, Performance & Resources

Deploying cryptographic algorithms on resource-constrained devices presents several challenges. Some of these challenges include low computation power, low energy availability, and memory constraints. Many resource-constrained devices lack the processing power needed for computationally intensive cryptographic algorithms. Their reliance on batteries necessitates energy-efficient security solutions, as high-complexity cryptographic operations can drain battery life and reduce device longevity. Additionally, limited ROM and RAM capacities make it challenging to deploy memory-intensive algorithms, especially those with large key sizes [9].

In [21], it is stated that the current NIST PQC standards cannot be directly implemented on low-memory devices because of large public-key and ciphertext sizes that lead to high memory requirements. A solution could be to use smaller key sizes to improve performance and reduce memory usage, however, this may reduce security as well. In environments where speed is not the primary concern, cryptographic algorithms can be designed to minimise memory usage, even if this comes at the cost of reduced performance [21]. Thus, a trade-off between performance and resources to achieve a desired security category is required.

## 1.6 Problem Statement

This thesis focuses on KEMs, rather than digital signature schemes, due to their role in key establishment for secure communication. The performance of KEMs, particularly in terms of their computational complexity, memory usage, and communication cost, scales with the input parameters chosen. Understanding these scaling characteristics is crucial for those aiming to implement these KEMs, especially in resource-constrained devices where efficiency is essential. Among the schemes selected by NIST, CRYSTALS-Kyber and HQC are the two KEMs, each based on distinct mathematical assumptions, MLWE and code-based decoding, respectively. The NIST-standardised variant of CRYSTALS-Kyber is designated as ML-KEM. The aim of this thesis is to conduct a theoretical complexity analysis of ML-KEM and HQC, providing insights into how their time, space, and bit complexities evolve as input parameters increase. Therefore, this master’s thesis aims to answer:

*How do the time, space, and bit complexities of lattice- and code-based post-quantum key encapsulation mechanisms scale with input parameters, and what are the implications for their use in resource-constrained devices?*

## 1.7 Problem Delimitations

The primary aim is to assess the time, space, and bit complexities of both schemes and understand how they scale with different input parameters. In addition to the complexity analysis, the thesis includes a comparative theoretical analysis of both KEMs. This analysis involves plotting and evaluating how the complexities of these schemes change with various parameter sets, providing insights into the trade-off between efficiency and security.

The scope of this thesis is limited in several ways. First, security proofs are not addressed. While the underlying security assumptions for both KEMs are presented, the focus is on understanding the complexity rather than proving their security. Second, this thesis does not include any practical implementations of the schemes, the analysis is entirely theoretical. The thesis also does not consider side-channel attacks or hardware-specific optimisations, focusing purely on the theoretical computational and memory characteristics of the schemes.

## 2 Lattices & Learning With Errors

The security of the ML-KEM is based on the hardness of the module LWE (MLWE) problem. The MLWE problem is rooted in the LWE problem, which is a special case of the lattice-based problem known as bounded distance decoding [5]. Consequently, the aim of this chapter is to present the essential mathematical definitions and theories related to lattices and lattice problems, culminating in the MLWE problem, which underpins the security of ML-KEM.

### 2.1 Prerequisites

Some necessary prerequisites are needed in order to define the LWE problems.

#### Modular Arithmetic

The congruence modulo operation partitions integers into  $q$  distinct equivalence classes based on their remainders when divided by  $q$ . Hence, two integers belong to the same equivalence class if they yield the same remainder [10, p. 17]. The set of equivalence classes under the congruence relation are called the set of integers modulo  $q$  [24, pp. 8–9].

**Definition 2.1 (Integers Modulo  $q$ )**

The set of integers modulo  $q$  is defined as

$$\mathbb{Z}/q\mathbb{Z} = \{0, 1, 2, \dots, q-1\},$$

equipped with the binary operations  $+$  and  $\cdot$ , corresponding to the operations in  $\mathbb{Z}$ :

$$\begin{aligned} a + b &\equiv c \pmod{q}, & c &\in \mathbb{Z}/q\mathbb{Z} \\ a \cdot b &\equiv d \pmod{q}, & d &\in \mathbb{Z}/q\mathbb{Z}, \end{aligned}$$

for all  $a, b \in \mathbb{Z}/q\mathbb{Z}$ . Each element of  $\mathbb{Z}/q\mathbb{Z}$  corresponds to a distinct equivalence class of integers under the congruence relation. [25, p. 18]

It is essential to note that the elements of  $\mathbb{Z}/q\mathbb{Z}$  are not integers but rather equivalence classes of integers under the congruence relation. Moreover,  $\mathbb{Z}/q\mathbb{Z}$  forms a field if and only if  $q$  is prime. This is because, if  $q$  is prime, then every non-zero element in  $\mathbb{Z}/q\mathbb{Z}$  have a multiplicative inverse [24, p. 10]. In the LWE problem, the space is defined by  $\mathbb{Z}/q\mathbb{Z}$ . Since  $q$  is typically chosen to be prime, this space forms a field. Additionally, the set  $\mathbb{Z}/q\mathbb{Z}$  is a ring of integers modulo  $q$  [12, p. 22].

## Rings

A ring is a set that has two binary operations which are linked by the distributive law [12, pp. 94–95].

### Definition 2.2 (Ring)

A ring is a set  $R$  equipped with the binary operations  $+$  and  $\cdot$  satisfying the following sets of axioms:

1.  $R$  is an abelian group under addition:
  - $a + (b + c) = (a + b) + c$  for all  $a, b, c \in R$  (associative law).
  - $a + b = b + a$  for all  $a, b \in R$  (commutative law).
  - There is an additive identity  $0 \in R$  such that  $0 + a = a + 0 = a$  for every  $a \in R$  (identity law).
  - For every element  $a \in R$ , there is an additive inverse  $b \in R$  such that  $a + b = b + a = 0$  (inverse law).
2. Multiplication is associative:
  - $a \cdot (b \cdot c) = (a \cdot b) \cdot c$  for all  $a, b, c \in R$ .
3. Multiplication is distributive with respect to addition:
  - $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$  for all  $a, b, c \in R$  (left distributivity).
  - $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$  for all  $a, b, c \in R$  (right distributivity).
4.  $R$  has a multiplicative identity:
  - There is an element  $1 \in R$  such that  $1 \cdot a = a \cdot 1 = a$  for every  $a \in R$ . [12, p. 95]

If multiplication is commutative, the ring  $R$  is commutative. Hence,  $a \cdot b = b \cdot a$  for all  $a, b \in R$  [12, p. 95]. Understanding the structural relationships between rings is essential in algebra. In particular, it is often useful to recognise when two rings, though defined differently or over distinct representations, exhibit identical algebraic behaviour. This notion is formalised through the definition of a ring isomorphism, which identifies when two rings can be considered structurally equivalent. ML-KEM leverages such equivalences to enable more efficient polynomial multiplication.

**Definition 2.3 (Ring Isomorphism)**

Let  $R$  and  $S$  be rings. A function

$$\phi : R \rightarrow S$$

is called a ring isomorphism if it satisfies the following properties for all elements  $a, b \in R$ :

1. Preservation of addition:

- $\phi(a + b) = \phi(a) + \phi(b)$ .

2. Preservation of multiplication:

- $\phi(ab) = \phi(a)\phi(b)$ .

3. Preservation of identities:

- $\phi(1_R) = 1_S$ .

4. One-to-one correspondence:

- The function  $\phi$  pairs each element of  $R$  with a unique element of  $S$ , and every element of  $S$  comes from exactly one element in  $R$ .

If such a function exists, the rings  $R$  and  $S$  are said to be isomorphic. This is written as:

$$R \cong S. \quad [24, \text{p. } 95]$$

Ring isomorphisms become especially useful when working with rings whose elements are polynomials. A ring formed from the set of polynomials whose coefficients are taken from a ring is called a polynomial ring. The set  $\mathbb{Z}/q\mathbb{Z}$  can be used to form a polynomial ring [12, p. 96].

**Definition 2.4 (Polynomial Ring)**

Given a commutative ring  $R$ , the set

$$R[x] = \{a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \mid a_i \in R, n \geq 0\},$$

forms a polynomial ring under standard polynomial addition and multiplication.

[12, p. 98]

If  $a_n \neq 0$ , the polynomial is said to be of degree  $n$ ,  $a_nx^n$  is the leading term, and  $a_n$  is called the leading coefficient [24, p. 234]. The MLWE problem is defined over the ring  $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ . The polynomial ring  $\mathbb{Z}_q[x]$  denotes the ring of polynomials with coefficients from  $\mathbb{Z}/q\mathbb{Z}$ . The ring  $R_q$  is a quotient ring formed by taking the polynomial ring  $\mathbb{Z}_q[x]$  modulo the ideal generated by  $x^n + 1$  [12, p. 22].

## 2.1. Prerequisites

### Definition 2.5 (Ideal)

Let  $R$  be a commutative ring. An ideal of  $R$  is a non-empty subset  $I \subseteq R$  that satisfies the following properties:

1.  $I$  is an abelian group under addition.
2. If  $a \in I$  and  $b \in R$ , then  $a \cdot b \in I$ . [10, p. 539]

Given a ring  $R$  and an ideal  $I$ , a quotient ring is formed by dividing  $R$  by  $I$ .

### Definition 2.6 (Quotient Ring)

Let  $R$  be a commutative ring and  $I \subseteq R$  an ideal. The quotient ring  $R/I$  is the set of equivalence classes of elements of  $R$  under the ideal  $I$ , where two elements  $a, b \in R$  are congruent modulo  $I$  if their difference  $a - b \in I$ . A quotient ring is equipped with the binary operations,  $+$  and  $\cdot$ . [12, p. 98]

By taking the quotient of the polynomial ring  $\mathbb{Z}_q[x]$  by the ideal generated by  $x^n + 1$ , the ring  $R_q$  is formed.

### Definition 2.7 (The Ring $R_q$ )

Let  $q$  be a prime integer modulus,  $n$  a positive integer, and  $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$  denote the ring of integers modulo  $q$ . Then, the quotient ring

$$R_q = \mathbb{Z}_q[x]/(x^n + 1)$$

consist of polynomials with coefficients in  $\mathbb{Z}_q$ , where addition and multiplication is performed modulo  $x^n + 1$ . [25, p. 399]

The ring  $R_q$  can be extended to modules by replacing the polynomials by vectors of polynomials. This extension is used in the MLWE problem.

**Definition 2.8 (Module)**

Let  $R$  be a commutative ring. A module over  $R$  is an abelian group  $M$  under addition, equipped with scalar multiplication,  $\cdot : R \times M \rightarrow M$ , that satisfies the following axioms for every  $a, b \in R$  and  $m, n \in M$ :

1.  $1 \cdot m = m$  (identity law).
2.  $(a \cdot b) \cdot m = a \cdot (b \cdot m)$  (associativity of scalar multiplication).
3.  $(a + b) \cdot m = a \cdot m + b \cdot m$  (distributivity over ring addition).
4.  $a \cdot (m + n) = a \cdot m + a \cdot n$  (distributivity over module addition). [13, p. 590]

Often  $M$  is referred to as  $R$ -module, however, for the sake of simplicity, it will be referred to as a module [13, p. 590]. Extending the ring  $R_q$  to a module structure by considering the set of vectors whose entries are elements of  $R_q$  leads to the definition of  $R_q^k$ , which forms a module over  $R_q$ .

**Definition 2.9 (The Module  $R_q^k$ )**

Let  $k$  be a positive integer. The set of vectors of length  $k$  with entries in  $R_q$  forms a module over  $R_q$ , denoted as

$$R_q^k = \{\mathbf{v} \mid v_i \in R_q\}.$$

Addition and subtraction of elements in  $R_q^k$  is component-wise. The inner product of two vectors in  $R_q^k$  results in a polynomial in  $R_q$ . [5]

The module  $R_q^k$  is used to represent the vectors in the noisy system of equations that define the MLWE problem. By embedding modules into real vector spaces, lattices are obtained, which can also be viewed as modules over rings of integers.

## 2.2 Lattice Problems

A lattice is a subset of the vector space  $\mathbb{R}^m$ . In a real vector space, the structure is defined by a basis, and every element of the space can be written as a linear combination of the basis vectors. Similarly, a lattice is defined by a basis. However, the elements of the lattice are integer linear combinations of the basis vectors, rather than real linear combinations [13, p. 357].

**Definition 2.10 (Lattice)**

Let  $\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n \in \mathbb{R}^m$  be a set of linearly independent vectors with  $n \leq m$ . The lattice generated by them is defined as

$$\mathcal{L}(\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n) = \left\{ \sum_{i=1}^n x_i \mathbf{b}_i \mid x_i \in \mathbb{Z} \right\},$$

where  $\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n$  are referred to as the basis vectors of the lattice. The lattice rank is  $n$ , and the lattice dimension is  $m$ . If  $n = m$ , then the lattice is said to be a full rank lattice. [12, p. 388]

The basis vectors  $\mathbf{b}_i$  of a lattice  $\mathcal{L}(\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n)$  can be arranged as rows in an  $n \times m$  matrix  $\mathbf{B}$ . This matrix  $\mathbf{B}$  is referred to as a basis matrix for the lattice, where  $B_{i,j}$  denotes the  $j$ th entry of the row  $\mathbf{b}_i$ . The lattice can then be expressed as

$$\mathcal{L}(\mathbf{B}) = \{\mathbf{B}\mathbf{x} \mid \mathbf{x} \in \mathbb{Z}^n\}.$$

A lattice can have more than one basis matrix [13, p. 357].

A fundamental parameter associated with a lattice is the length of its shortest non-zero vector, commonly denoted by  $\lambda_1$ . This length is typically measured using the Euclidean norm [10, p. 349]. The problem of finding the shortest non-zero vector in a lattice is known as the shortest vector problem.

**Definition 2.11 (Shortest Vector Problem)**

Let  $\mathcal{L} = \mathcal{L}(\mathbf{B})$  be a lattice. Find a non-zero vector  $\mathbf{v} \in \mathcal{L}$ , such that  $\|\mathbf{v}\|_2$  is minimised. Such a vector  $\mathbf{v}$  is called a shortest vector in  $\mathcal{L}(\mathbf{B})$ . [12, p. 395]

Note that there may be more than one shortest non-zero vector in a lattice [12, p. 395]. A related problem is the closest vector problem, which involves finding the lattice point closest to a given point in space.

**Definition 2.12 (Closest Vector Problem)**

Consider a basis matrix  $\mathbf{B}$  for a lattice  $\mathcal{L}$  and a vector  $\mathbf{w} \in \mathbb{R}^m$  that is not in  $\mathcal{L}$ . Find a vector  $\mathbf{v} \in \mathcal{L}$  such that  $\|\mathbf{w} - \mathbf{v}\|_2$  is minimised. Such a vector  $\mathbf{v}$  is called a closest vector to  $\mathbf{w}$  in  $\mathcal{L}$ . [12, p. 395]

A special case of the closest vector problem is the bounded distance decoding problem, where the vector  $\mathbf{w}$  is sufficiently close to a lattice point, within a factor  $\alpha$  of the shortest vector of the lattice.

## 2.3. Learning With Errors

### Definition 2.13 (Bounded Distance Decoding Problem)

Let  $\alpha \in (0, 1)$ . Given a basis matrix  $\mathbf{B}$  for a lattice  $\mathcal{L}$ , let  $\mathbf{w} \in \mathbb{R}^m$  such that there exists a lattice point  $\mathbf{v} \in \mathcal{L}$  with

$$\|\mathbf{w} - \mathbf{v}\|_2 \leq \alpha \lambda_1,$$

where  $\lambda_1$  is the length of the shortest non-zero vector in the lattice  $\mathcal{L}$ . The bounded distance decoding problem is to compute  $\mathbf{v}$ . [13, p. 363]

Bounded distance decoding is a computationally hard problem in lattice-based cryptography and is closely related to the LWE problem. In fact, LWE is a special case of this problem, with reductions showing that solving bounded distance decoding is equivalent to solving LWE in certain cases [13, pp. 415–416].

## 2.3 Learning With Errors

Finding solutions to a system of linear equations with  $n$  variables over  $\mathbb{Z}_q$  given a prime  $q$ , and introducing chosen randomness, results in the LWE problem [10, p. 351].

### Definition 2.14 (Learning With Errors Problem)

Let  $q \in \mathbb{N}$  be a prime modulus,  $\sigma \in \mathbb{R}_{>0}$  a noise parameter, and  $k, \ell \in \mathbb{N}$  with  $k > \ell$ . Let the secret vector  $\mathbf{s} \in (\mathbb{Z}/q\mathbb{Z})^\ell$  be chosen uniformly at random and the matrix  $\mathbf{A} \in (\mathbb{Z}/q\mathbb{Z})^{k \times \ell}$  have entries sampled independently and uniformly from  $\mathbb{Z}/q\mathbb{Z}$ . Let the error vector  $\mathbf{e} \in \mathbb{Z}^k$  have entries sampled independently from a discrete normal distribution over  $\mathbb{Z}$  with mean 0 and standard deviation  $\sigma$ . The LWE instance is then given by

$$\mathbf{t} \equiv \mathbf{A}\mathbf{s} + \mathbf{e} \pmod{q},$$

where  $\mathbf{t} \in (\mathbb{Z}/q\mathbb{Z})^k$ . The objective of the LWE problem is to recover the secret vector  $\mathbf{s}$  given  $(\mathbf{A}, \mathbf{t})$ . [13, pp. 414–415]

Recovering the secret vector  $\mathbf{s}$  from an LWE instance is not always possible. Without noise, the problem reduces to solving a linear system over  $\mathbb{Z}/q\mathbb{Z}$ , which can be done using standard linear algebra. However, when noise is introduced, the problem becomes computationally hard. The hardness critically depends on the noise parameter  $\sigma$ . If it is too small, the structure of the system can still be exploited by an attacker. If it is too large, the noise can overwhelm the signal entirely, potentially breaking correctness. Theoretical results by Regev shows that when the noise is chosen appropriately, the LWE problem is at least as hard as well-known worst-case lattice problems [26].

A generalisation of the LWE problem is the MLWE problem, which replaces the vector and matrix operations with operations on modules over rings.

**Definition 2.15 (Module Learning With Errors Problem)**

Let  $q \in \mathbb{N}$  be a prime and  $n, k, \ell \in \mathbb{N}$  such that  $k \geq \ell$ . Additionally, let  $\eta_1, \eta_2$  be small noise parameters satisfying  $\eta_1, \eta_2 \ll q/2$ .

An MLWE problem is defined over the ring  $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ . A matrix  $\mathbf{A} \in R_q^{k \times \ell}$  is chosen uniformly at random. A secret vector  $\mathbf{s} \in R_q^\ell$  is sampled from a distribution  $S_{\eta_1}$ , and an error vector  $\mathbf{e} \in R_q^k$  is sampled from another distribution  $S_{\eta_2}$ . Here,  $S_{\eta_1}$  and  $S_{\eta_2}$  refer to distributions over polynomials in  $R_q$  whose coefficients are small integers, sampled independently from a discrete normal distribution with standard deviation  $\eta_1$  and  $\eta_2$ , respectively. The MLWE instance is then given by the equation

$$\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e},$$

where  $t \in R_q^k$ . The objective of the MLWE problem is to recover the secret vector  $\mathbf{s}$  given  $(\mathbf{A}, \mathbf{t})$ . [27, pp. 277–278]

Setting  $n = 1$  gives an instance of LWE as  $R_q$  is replaced with  $\mathbb{Z}_q$  [27, pp. 277–278]. The MLWE problem forms the hardness assumption underlying ML-KEM.

# 3 ML-KEM Standard

ML-KEM employs a two-stage approach. The first stage introduces a PKE scheme called K-PKE. However, K-PKE, in its stand-alone form, is not considered secure due to vulnerabilities in its design [5]. Specifically, K-PKE only achieves IND-CPA security. To address these concerns and ensure IND-CCA2, the second stage of ML-KEM strengthens K-PKE by applying a variant of the Fujisaki-Okamoto (FO) transformation. The security of ML-KEM relies on the computational difficulty of the MLWE problem, which is currently believed to be quantum secure. NIST specifies three parameter sets for ML-KEM, each offering increasing security strength.

## 3.1 Prerequisites

Before presenting the ML-KEM scheme, several key concepts and functions that are fundamental to the scheme must be introduced. This section outlines the essential prerequisites required for a clear understanding of the scheme's presentation later in this chapter.

### Compression & Decompression

In ML-KEM, the compression and decompression functions reduce the ciphertext size by discarding certain low-order bits while preserving sufficient information for reconstruction [28]. Let  $q$  be a prime integer, and let  $a, b$  be polynomials in  $\mathbb{Z}_q[x]$  and  $\mathbb{Z}_{2^d}[x]$ , respectively, where  $1 \leq d \leq \lfloor \log_2(q) \rfloor$ . The compression and decompression functions,

$$\text{Compress}_q(a, d) : \mathbb{Z}_q[x] \rightarrow \mathbb{Z}_{2^d}[x] \quad \text{and} \quad \text{Decompress}_q(b, d) : \mathbb{Z}_{2^d}[x] \rightarrow \mathbb{Z}_q[x],$$

are applied coefficient-wise. For each coefficient  $a_i$  of  $a$  and  $b_i$  of  $b$ , the functions are defined as

$$\text{Compress}_q(a_i, d) = \lceil (2^d/q) \cdot a_i \rceil \mod 2^d, \tag{3.1}$$

$$\text{Decompress}_q(b_i, d) = \lceil (q/2^d) \cdot b_i \rceil \mod q, \tag{3.2}$$

[28]. These functions must approximately reconstruct the original input, meaning that

$$a' = \text{Decompress}_q(\text{Compress}_q(a, d), d),$$

where  $a'$  is a close approximation of  $a$ . The approximation is bounded by the following property

$$\|a' - a\|_\infty \leq \left\lceil \frac{q}{2^{d+1}} \right\rceil, \tag{3.3}$$

where  $\|\cdot\|_\infty$  denotes the  $\ell_\infty$  norm [28].

### 3.1. Prerequisites

#### Central Binomial Distribution

The central binomial distribution  $B_\eta$  is utilised in ML-KEM to sample polynomials. Given a parameter  $\eta$ , independent uniform bits  $(x_1, y_1, x_2, y_2, \dots, x_\eta, y_\eta) \in \{0, 1\}^{2\eta}$  are sampled, and the output is computed as

$$c = \sum_{i=1}^{\eta} (x_i - y_i)$$

[28]. The coefficient  $c$  takes values in the range  $[-\eta, \eta]$  and follows a symmetric discrete distribution centred at zero. When a polynomial  $f \in R_q$  is sampled from  $B_\eta$ , each coefficient is independently drawn from  $B_\eta$ . The central binomial distribution for  $\eta = 2$  can be seen in Figure 3.1.

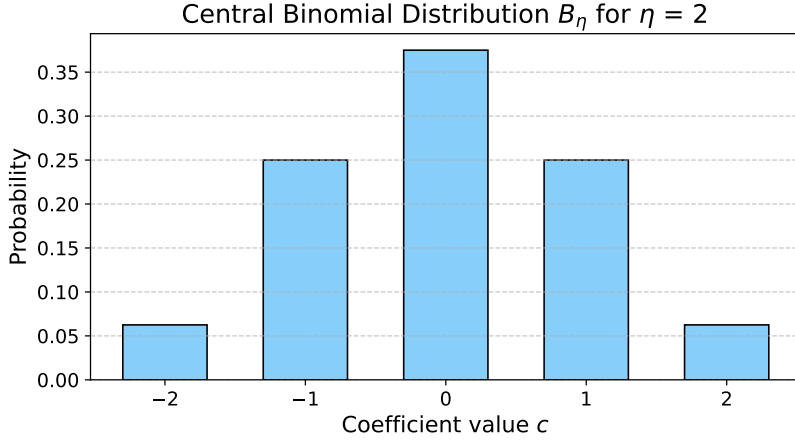


Figure 3.1

#### Hash Functions

In ML-KEM, cryptographic hash functions ensure one-wayness and collision resistance [5]. These functions map inputs of arbitrary length to fixed-length outputs. A one-way function makes it computationally infeasible to recover an input from its output. Collision resistance ensures that no two distinct inputs produce the same output. ML-KEM employs three hash functions,  $\mathcal{G}$ ,  $\mathcal{H}$ , and  $\mathcal{J}$ , defined as

$$\mathcal{G} : \{0, 1\}^* \rightarrow \{0, 1\}^{256} \times \{0, 1\}^{256}, \quad (3.4)$$

$$\mathcal{H} : \{0, 1\}^* \rightarrow \{0, 1\}^{256}, \quad (3.5)$$

$$\mathcal{J} : \{0, 1\}^* \rightarrow \{0, 1\}^{256}. \quad (3.6)$$

The functions  $\mathcal{H}$  and  $\mathcal{J}$  both take a variable-length binary input and produce a 256-bit output. Specifically,  $\mathcal{H}$  is instantiated as SHA3-256, while  $\mathcal{J}$  is instantiated using SHAKE256 with a fixed 256-bit output [5].  $\mathcal{G}$  takes a variable-length input and produces two 256-bit outputs by applying SHA3-512 and interpreting the 512-bit result as a concatenation of two 256-bit values [5]. All three functions are based on members of the SHA-3 family, which includes both fixed-output hash functions, SHA3-256, and extendable-output functions such

### 3.2. K-PKE

as SHAKE256. These hash functions are used throughout ML-KEM to derive cryptographic keys and generate randomness. Implementation and details of these functions are omitted in this thesis. For further information see [29] [5].

## 3.2 K-PKE

The K-PKE serves as a fundamental component of ML-KEM. It consists of three algorithms:

1. *Key generation*, which produces a public encryption key and private decryption key.
2. *Encryption*, which takes as input a public encryption key and a message, producing a ciphertext.
3. *Decryption*, which allows the recipient to recover the original message from the ciphertext using the private decryption key.

The scheme is parameterised by the integers  $n, k, q, \eta_1, \eta_2, d_u$ , and  $d_v$  [5]. The two communicating parties will be referred to as Alice and Bob. The following subsections provide a description of the three algorithms in K-PKE.

### Key Generation

To enable encryption and decryption, Alice must generate a public encryption key and a corresponding private decryption key. She begins by selecting a public random seed  $\rho \in \{0, 1\}^{256}$  and expands it using a cryptographic hash function to generate the matrix  $\mathbf{A} \in R_q^{k \times k}$ . She then constructs a secret vector  $\mathbf{s} \in R_q^k$  and an error vector  $\mathbf{e} \in R_q^k$  by independently sampling each of their components from  $B_\eta$  with parameters  $\eta_1$  and  $\eta_2$ , respectively. Alice then computes

$$\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}. \quad (3.7)$$

Finally, Alice's public encryption key is given by  $(\rho, \mathbf{t})$ , while her private decryption key is  $\mathbf{s}$ . The addition of the noise vector  $\mathbf{e}$  ensures that the encryption scheme maintains security based on the MLWE. The noise makes it computationally infeasible for an attacker to recover  $\mathbf{s}$  given  $(\rho, \mathbf{t})$  [5].

### Encryption

Bob aims to encrypt a message  $m \in \{0, 1\}^n$  for Alice using her public encryption key. To achieve this, Bob first obtains an authentic copy of Alice's public encryption key  $(\rho, \mathbf{t})$ . He then computes the matrix  $\mathbf{A}$  by expanding  $\rho$  using a cryptographic hash function. Next, Bob samples a random vector  $\mathbf{r} \in R_q^k$  from  $B_{\eta_1}$ , and noise terms  $\mathbf{e}_1 \in R_q^k$  and  $e_2 \in R_q$  from  $B_{\eta_2}$ . Bob computes

$$\mathbf{u} = \mathbf{A}^\top \mathbf{r} + \mathbf{e}_1, \quad (3.8)$$

$$v = \mathbf{t}^\top \mathbf{r} + e_2 + \left\lceil \frac{q}{2} \right\rceil m, \quad (3.9)$$

### 3.2. K-PKE

where the term  $\lceil \frac{q}{2} \rceil m$  is used to encode the message  $m$  into a ciphertext [5]. Bob then compresses both  $\mathbf{u}$  and  $v$  to ensure they fit within the bounds defined by the parameters  $d_u$  and  $d_v$ . Specifically, he computes

$$\begin{aligned} c_1 &= \text{Compress}_q(\mathbf{u}, d_u), \\ c_2 &= \text{Compress}_q(v, d_v), \end{aligned}$$

where  $\text{Compress}_q$  denotes the compression function defined in (3.1).

Finally, Bob outputs the ciphertext

$$c = (c_1, c_2).$$

The ciphertext can be securely sent to Alice and contains the necessary information for Alice to decrypt the message using her private decryption key.

### Decryption

To decrypt the ciphertext  $c$ , Alice follows the steps outlined below. First, Alice computes

$$\bar{\mathbf{u}} = \text{Decompress}_q(c_1, d_u), \tag{3.10}$$

$$\bar{v} = \text{Decompress}_q(c_2, d_v). \tag{3.11}$$

The decompression function (3.2) recovers the approximations  $\bar{\mathbf{u}}$  and  $\bar{v}$  by reversing the compression applied during encryption. Next, Alice computes

$$m' = \text{Compress}_q(\bar{v} - \mathbf{s}^\top \bar{\mathbf{u}}, 1) \tag{3.12}$$

[5]. The final result gives Alice an approximation  $m'$  of the original message  $m$ , completing the decryption process. The error between  $m'$  and  $m$  is bounded by (3.3).

Example 3.1 presents a simplified version of key generation, encryption, and decryption in K-PKE. Some steps are omitted for clarity, but the goal of the example is to provide an intuitive understanding of the overall scheme.

---

#### Example 3.1 (Simplified K-PKE)

Let  $q = 79$ ,  $n = 3$ ,  $k = 2$ , and  $\eta_1 = \eta_2 = 2$ . Alice selects the following

$$\mathbf{A} = \begin{bmatrix} 17 + 60x + 34x^2 & 2 + 71x + 24x^2 \\ 77 + 6x + 40x^2 & 31 + 78x + 5x^2 \end{bmatrix}, \quad \mathbf{s} = \begin{bmatrix} 2 + x - x^2 \\ -x + 2x^2 \end{bmatrix}, \quad \mathbf{e} = \begin{bmatrix} 1 + x^2 \\ 2x - x^2 \end{bmatrix}.$$

Then she computes

$$\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e} = \begin{bmatrix} 22 + 42x + 45x^2 \\ 48 + 11x + 71x^2 \end{bmatrix}.$$

### 3.3. ML-KEM

The public encryption key  $(\mathbf{A}, \mathbf{t})$  is sent to Bob.

Bob encrypts a message  $m = 011 \rightarrow x + x^2$  to Alice by first selecting

$$\mathbf{r} = \begin{bmatrix} 1 + 2x - x^2 \\ -2 + x + x^2 \end{bmatrix}, \quad \mathbf{e}_1 = \begin{bmatrix} 2 - 2x^2 \\ 1 + x + x^2 \end{bmatrix}, \quad e_2 = 2 + 2x + x^2.$$

He then calculates

$$\mathbf{u} = \mathbf{A}^\top \mathbf{r} + \mathbf{e}_1 = \begin{bmatrix} 48 + 74x + 59x^2 \\ 39 + 49x + 27x^2 \end{bmatrix}$$

and

$$v = \mathbf{t}^\top \mathbf{r} + e_2 + \left\lceil \frac{q}{2} \right\rceil m = 35 + 49x + 65x^2.$$

Alice then receives the ciphertext  $c = (\mathbf{u}, v)$ . To decrypt the ciphertext, she uses her private decryption key  $\mathbf{s}$  to compute

$$v - \mathbf{s}^\top \mathbf{u} = 74 + 45x + 50x^2.$$

Then by using the compression function, she obtains

$$\text{Compress}_q(74 + 45x + 50x^2, 1) = x + x^2,$$

which corresponds to the encoded message  $m = 011$ .

---

### 3.3 ML-KEM

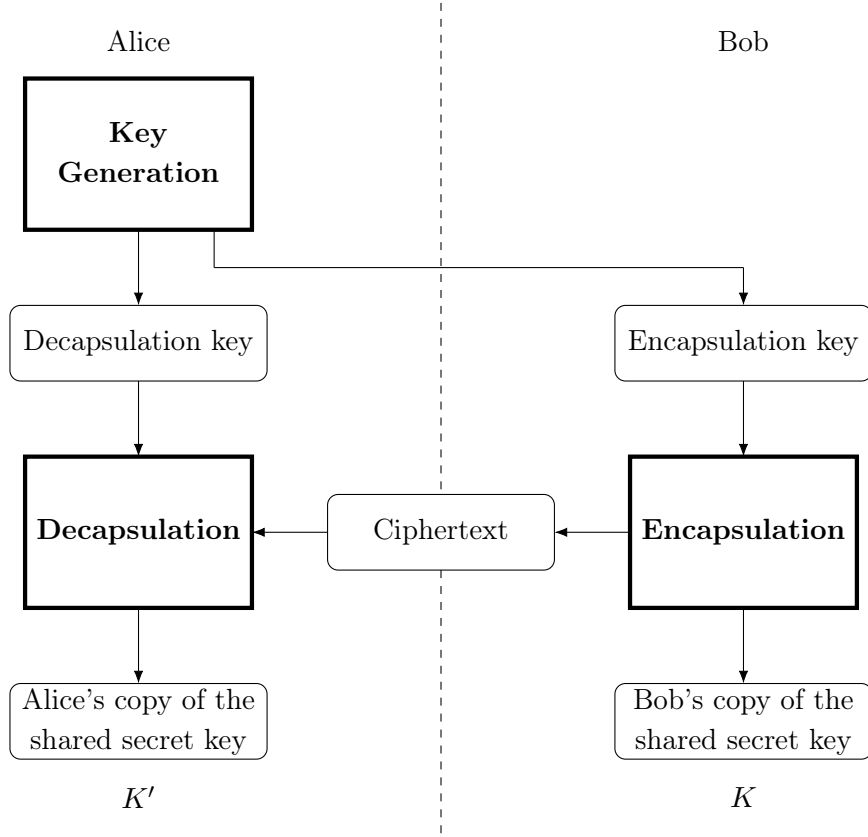
A KEM can be used to establish a shared secret key between two communicating parties [5]. This shared secret key can then be used for secret-key cryptography. A KEM consists of three algorithms:

1. *Key generation*, Alice generates a public encapsulation key  $ek$  and a private decapsulation key  $dk$ .
2. *Encapsulation*, Bob uses Alice's public encapsulation key  $ek$  to generate a shared secret key  $K$  and ciphertext  $c$ , and sends  $c$  to Alice.
3. *Decapsulation*, Alice uses her private decapsulation key  $dk$  to recover  $K$  from the ciphertext  $c$ .

While K-PKE is secure under the IND-CPA model, a variant of the FO transform enhances its security, resulting in ML-KEM, which is secure under the stronger IND-CCA2 model [28]. The FO transform incorporates a verification step during decapsulation, preventing chosen ciphertext attacks. The details regarding the FO transform are beyond the scope of this thesis and will therefore not be included. For details on IND-CPA and IND-CCA2 security models,

### 3.3. ML-KEM

see Section 1.2. An overview of key establishment using a KEM is shown in Figure 3.2.



**Figure 3.2:** Representation of the algorithms and steps involved in key establishment using a KEM. The figure is remade from [5].

The remainder of this section provides a more detailed explanation of each of the three core algorithms in ML-KEM, beginning with key generation.

#### Key Generation

Alice employs the K-PKE key generation algorithm to derive a public encryption key  $(\rho, \mathbf{t})$  and a private decryption key  $\mathbf{s}$ . The public encryption key serves as the public encapsulation key, defined as  $ek = (\rho, \mathbf{t})$ .

To construct the private decapsulation key, an implicit rejection value  $z \in \{0, 1\}^{256}$  is selected, which is used in case of decryption failure to prevent invalid ciphertexts from leaking information to an adversary [28]. The private decapsulation key is then formulated as

$$dk = (\mathbf{s}, ek, \mathcal{H}(ek), z), \quad (3.13)$$

where  $\mathcal{H}$  is a hash function defined in (3.5) [5].

### 3.4. Parameter Sets

#### Encapsulation

Given a public encapsulation key  $ek$ , Bob selects a random message representative  $m \in \{0, 1\}^n$ . Afterward, the shared secret key  $K \in \{0, 1\}^{256}$  and randomness  $r \in \{0, 1\}^{256}$  are derived as

$$(K, r) = \mathcal{G}(m, \mathcal{H}(ek)), \quad (3.14)$$

where  $\mathcal{G}$  is a hash function defined in (3.4) [5]. The randomness  $r$  serves as a seed in K-PKE, determining the pseudorandom sampling of polynomials from the central binomial distribution. Using  $ek$  and  $r$ , K-PKE encrypts  $m$  to produce the ciphertext  $c$ . Thus, the encapsulation outputs the shared secret key  $K$  and ciphertext  $c$ . The ciphertext is then sent to Alice.

#### Decapsulation

To recover the shared secret key  $K$  from the ciphertext  $c$ , Alice uses her private decapsulation key  $dk$ . First, she extracts her K-PKE private decryption key  $\mathbf{s}$  from  $dk$  and applies the K-PKE decryption algorithm with the inputs  $\mathbf{s}$  and  $c$ , resulting in a decrypted ciphertext denoted  $m' \in \{0, 1\}^n$ . Next, Alice derives the shared secret key candidate  $K' \in \{0, 1\}^{256}$  and the randomness  $r' \in \{0, 1\}^{256}$  by applying the cryptographic hash function  $\mathcal{G}$ . This is expressed as

$$(K', r') = \mathcal{G}(m', \mathcal{H}(ek)),$$

[5]. Alice then extracts  $z$  from  $dk$ , which is used to derive an alternative shared secret key  $\bar{K} \in \{0, 1\}^{256}$  as

$$\bar{K} = \mathcal{J}(z, c), \quad (3.15)$$

where  $\mathcal{J}$  is a cryptographic hash function defined in (3.6) [5].

Alice re-encrypts  $m'$  using the K-PKE encryption algorithm with the public encapsulation key  $ek$  and the derived randomness  $r'$ , producing a new ciphertext  $c'$ . If the encryption and decryption processes were performed correctly,  $c'$  should match the original ciphertext  $c$ . Alice then verifies whether the received ciphertext  $c$  matches the recomputed ciphertext  $c'$ . If  $c = c'$ , the derived shared secret  $K'$  is returned. Otherwise, if  $c \neq c'$ , indicating potential manipulation or corruption of the ciphertext, Alice returns the alternative secret  $\bar{K}$ , derived from  $z$  and  $c$  [5].

### 3.4 Parameter Sets

In ML-KEM, various parameters can be adjusted to achieve different security categories. NIST has standardised three parameter sets, each corresponding to a specific security category. These parameter sets are presented in Table 3.1.

The parameter  $n$  defines the size of the message, which is fixed at 256 bits. The prime  $q = 3329$  is selected to facilitate efficient number-theoretic transform (NTT) multiplication while maintaining a negligible failure probability [28]. The parameter  $k$  determines the lattice dimension, serving as the primary mechanism for scaling security [28]. The parameters  $\eta_1$  and

### 3.4. Parameter Sets

$\eta_2$  define the distributions used in key generation and encryption, contributing to the overall security of the scheme. Finally, the parameters  $d_u$  and  $d_v$  regulate compression, impacting both ciphertext size and error resilience.

Parameter Set	$n$	$q$	$k$	$\eta_1$	$\eta_2$	$d_u$	$d_v$	Security Category
ML-KEM-512	256	3329	2	3	2	10	4	1
ML-KEM-768	256	3329	3	2	2	10	4	3
ML-KEM-1024	256	3329	4	2	2	11	5	5

**Table 3.1:** Approved parameter sets for ML-KEM [5].

The sizes of various components in ML-KEM depend on the chosen parameter set. As shown in Table 3.2, the parameter sets each corresponds to different key and ciphertext sizes. As the parameter set increases, both the public encapsulation and private decapsulation key sizes grow to enhance security, resulting in a larger ciphertext. However, the shared secret key remains fixed at 32 bytes across all parameter sets.

Parameter Set	Public Encapsulation Key	Private Decapsulation Key	Shared Secret Key	Ciphertext
ML-KEM-512	800	1632	32	768
ML-KEM-768	1184	2400	32	1088
ML-KEM-1024	1568	3168	32	1568

**Table 3.2:** Sizes in bytes of keys and ciphertext of ML-KEM [5].

In all parameter sets, a corresponding probability of decapsulation failure exists. A decapsulation failure occurs when  $K' \neq K$ , given that all inputs are well-formed and randomness generation is successful [5]. The decapsulation failure rates for different parameter sets are presented in Table 3.3.

Parameter Set	Decapsulation Failure Rate
ML-KEM-512	$2^{-138.8}$
ML-KEM-768	$2^{-164.8}$
ML-KEM-1024	$2^{-174.8}$

**Table 3.3:** Decapsulation failure rates for different ML-KEM parameter sets, as reported in [5].

### 3.5. Number-Theoretic Transform

The decapsulation process and its associated failure rates highlight the importance of efficient computation in ML-KEM. One critical technique used to enhance efficiency in ML-KEM is the NTT.

## 3.5 Number-Theoretic Transform

The NTT is a specialised variant of the discrete Fourier transform optimised for polynomial multiplication over finite fields [5]. In the ML-KEM, it enables efficient multiplication in the quotient ring  $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ , where  $n = 256$  and  $q = 3329$ . Since  $q$  is prime and  $x^n + 1$  is irreducible over  $\mathbb{Z}_q$ , the quotient ring  $R_q$  is a finite field.

### Definition 3.2 (Primitive $n$ th Root of Unity)

Let  $\mathbb{Z}_q$  be the ring of integers modulo  $q$ , and let  $f$  and  $g$  be polynomials with a maximum degree of  $n - 1$ . In this ring, there exists a multiplicative identity of 1. A number  $\zeta$  is called a primitive  $n$ th root of unity in  $\mathbb{Z}_q$  if and only if it satisfies the following conditions

$$\zeta^n \equiv 1 \pmod{q}$$

and

$$\zeta^m \not\equiv 1 \pmod{q}, \quad \text{for } m = 1, 2, \dots, n - 1. \quad [30]$$

In the ML-KEM, the modulus  $q$  is a prime number given by

$$q = 3329 = 2^8 \cdot 13 + 1. \quad (3.15)$$

Since  $256 = 2^8$  divides  $q - 1$ , there exist 256th primitive roots of unity modulo  $q$  [5]. Specifically,  $\zeta = 17 \in \mathbb{Z}_q$  is a primitive 256th root of unity, meaning that

$$\zeta^{128} \equiv -1 \pmod{q}. \quad (3.15)$$

Therefore, the factorisation of the polynomial  $x^{256} + 1$  modulo  $q$ , splits into irreducible quadratic polynomials, given by

$$x^{256} + 1 = \prod_{i=0}^{127} (x^2 - \zeta^{2i+1}) \pmod{q} \quad (3.15)$$

[28]. The NTT is a ring isomorphism, see Definition 2.3, that maps the ring  $R_q$  to a transformed domain  $T_q$ , such that  $R_q \cong T_q$ . This is defined as

$$\text{NTT} : R_q \rightarrow T_q, \quad (3.15)$$

and it has the property that

$$\text{NTT}^{-1}(\text{NTT}(a)) = a, \quad \text{for all } a \in R_q \quad (3.15)$$

### 3.5. Number-Theoretic Transform

[30]. This property ensures that for any two polynomials  $f, g \in R_q$ , their NTT representations  $\hat{f}, \hat{g} \in T_q$  allow multiplication in  $R_q$  to be computed as

$$f \cdot g = \text{NTT}^{-1}(\hat{f} \cdot \hat{g}), \quad (3.15)$$

[5]. Since multiplication in  $T_q$  is performed component-wise on elements of lower degree, it is significantly more computationally efficient than direct polynomial multiplication in  $R_q$ .

The polynomial  $x^{256} + 1$  admits a factorisation over  $\mathbb{Z}_q$  into 128 quadratic factors as expressed in (3.5). Consequently, the transformed domain  $T_q$  can be decomposed as a direct sum of 128 smaller rings

$$T_q = \bigoplus_{i=0}^{127} \mathbb{Z}_q[x] / (x^2 - \zeta^{2i+1})$$

[5]. Let  $f \in R_q$  be a polynomial, where  $n = 2^k$  and  $q$  is a prime such that  $q - 1$  is divisible by  $n$ , but not by  $2n$ . The NTT representation of  $f$  in the transformed domain  $T_q$  is given by the vector

$$\hat{f} = (f \bmod x^2 - \zeta^1, \dots, f \bmod x^2 - \zeta^{2(n/2-1)+1}).$$

The algebraic representation of  $\text{NTT}(f) = \hat{f}$  consists of  $n/2$  polynomials of degree 1

$$\text{NTT}(f) = \hat{f} = (\hat{f}_0 + \hat{f}_1 x, \hat{f}_2 + \hat{f}_3 x, \dots, \hat{f}_{n-2} + \hat{f}_{n-1} x),$$

where the coefficients are computed as

$$\hat{f}_{2i} = \sum_{j=0}^{n/2-1} f_{2j} \zeta^{(2i+1)j} \bmod q, \quad (3.16)$$

$$\hat{f}_{2i+1} = \sum_{j=0}^{n/2-1} f_{2j+1} \zeta^{(2i+1)j} \bmod q, \quad (3.17)$$

[28]. The inverse NTT reconstructs  $f$  as

$$\text{NTT}^{-1}(\hat{f}) = f = f_0 + f_1 x + \dots + f_{n-1} x^{n-1},$$

where the coefficients are recovered by

$$f_{2j} = \frac{1}{n/2} \sum_{i=0}^{n/2-1} \hat{f}_{2i} \zeta^{-(2i+1)j} \bmod q, \quad (3.18)$$

$$f_{2j+1} = \frac{1}{n/2} \sum_{i=0}^{n/2-1} \hat{f}_{2i+1} \zeta^{-(2i+1)j} \bmod q. \quad (3.19)$$

For further insight into the NTT, an example is provided in Example 3.3.

**Example 3.3 (NTT)**

Consider the ring  $R_q = \mathbb{Z}_{13}[x]/(x^4 + 1)$ , a primitive root of unity  $\zeta$  is sought such that

$$\zeta^4 \equiv 1 \pmod{13}.$$

It is found that  $\zeta = 5$  satisfies the conditions in Definition 3.2. Let the polynomials  $a, b \in R_q$  be given by

$$a = 3 + 5x + 2x^2 + 7x^3 \quad \text{and} \quad b = 6 + 4x + x^2 + 3x^3.$$

The goal is to compute the product  $c = a \cdot b \pmod{(x^4 + 1)}$ . This can be efficiently performed using the NTT by computing

$$\hat{a} = \text{NTT}(a), \quad \hat{b} = \text{NTT}(b), \quad \hat{c} = \hat{a} \cdot \hat{b} \quad \text{and} \quad c = \text{NTT}^{-1}(\hat{c}).$$

To compute the NTT of  $a$ , the formulas in (3.16) and (3.17) are applied

$$\begin{aligned} \hat{a}_0 &= \sum_{j=0}^1 a_{2j} \zeta^j \pmod{13} = 13 \pmod{13} = 0, \\ \hat{a}_1 &= \sum_{j=0}^1 a_{2j+1} \zeta^j \pmod{13} = 40 \pmod{13} = 1, \\ \hat{a}_2 &= \sum_{j=0}^1 a_{2j} \zeta^{3j} \pmod{13} = 253 \pmod{13} = 6, \\ \hat{a}_3 &= \sum_{j=0}^1 a_{2j+1} \zeta^{3j} \pmod{13} = 880 \pmod{13} = 9. \end{aligned}$$

Thus, the NTT of  $a$  is

$$\hat{a} = \text{NTT}(a) = (x, 6 + 9x).$$

The NTT of  $b$  is calculated in the same manner, resulting in

$$\hat{b} = \text{NTT}(b) = (11 + 6x, 1 + 2x).$$

The product  $\hat{c} = \hat{a} \cdot \hat{b}$  is calculated component-wise

$$\hat{c} = (x, 6 + 9x) \cdot (11 + 6x, 1 + 2x) = (4 + 11x, 7 + 8x),$$

To recover  $c$  from  $\hat{c}$ , the inverse NTT formulas in (3.18) and (3.19) are applied. Specifically, for each coefficient  $c_i$ , the inverse NTT is computed as follows

$$\begin{aligned} c_0 &= \frac{1}{2} \sum_{i=0}^1 \hat{c}_{2i} \zeta^0 \pmod{13} = \frac{1}{2} \cdot 11 \pmod{13} = 12, \\ c_1 &= \frac{1}{2} \sum_{i=0}^1 \hat{c}_{2i+1} \zeta^0 \pmod{13} = \frac{1}{2} \cdot 19 \pmod{13} = 3, \\ c_2 &= \frac{1}{2} \sum_{i=0}^1 \hat{c}_{2i} \zeta^{-2i+1} \pmod{13} = \frac{1}{2} \cdot 72 \pmod{13} = 1, \\ c_3 &= \frac{1}{2} \sum_{i=0}^1 \hat{c}_{2i+1} \zeta^{-2i+1} \pmod{13} = \frac{1}{2} \cdot 128 \pmod{13} = 12. \end{aligned}$$

### 3.5. Number-Theoretic Transform

In these calculations, the following modular inverses are used

$$\frac{1}{2} \bmod 13 = 7, \quad \frac{1}{5} \bmod 13 = 8 \quad \text{and} \quad \frac{1}{5^3} \bmod 13 = 5.$$

Thus, the final result for  $c$  is

$$c = 12 + 3x + x^2 + 12x^3.$$

This completes the computation of the product  $c = a \cdot b \bmod (x^4 + 1)$  using the NTT and its inverse.

A summary of how and where the NTT is applied across the different algorithms of ML-KEM can be found in Table 3.4.

Algorithm	Operation	NTT Use
Key Generation	Expand matrix $\mathbf{A}$	$\hat{\mathbf{A}} = \text{Expand}_{\text{NTT}}(\rho)$
	Transform vectors	$\hat{\mathbf{s}} = \text{NTT}(\mathbf{s}), \quad \hat{\mathbf{e}} = \text{NTT}(\mathbf{e})$
	Matrix-vector product	$\hat{\mathbf{t}} = \hat{\mathbf{A}}\hat{\mathbf{s}} + \hat{\mathbf{e}}$
Encapsulation	Transform vector	$\hat{\mathbf{r}} = \text{NTT}(\mathbf{r})$
	Compute ciphertext $\mathbf{u}$	$\mathbf{u} = \text{NTT}^{-1}(\hat{\mathbf{A}}^\top \hat{\mathbf{r}}) + \mathbf{e}_1$
	Compute ciphertext $v$	$v = \text{NTT}^{-1}(\hat{\mathbf{t}}^\top \hat{\mathbf{r}}) + e_2 + \lceil \frac{q}{2} \rceil m$
Decapsulation	Transform received $\bar{\mathbf{u}}$	$\hat{\bar{\mathbf{u}}} = \text{NTT}(\bar{\mathbf{u}})$
	Inner product	$\bar{v} = \text{NTT}^{-1}(\hat{\bar{\mathbf{u}}}^\top \hat{\mathbf{s}})$

**Table 3.4:** Use of the NTT across the ML-KEM algorithms [5].

The application of the NTT significantly enhances the computational efficiency of ML-KEM, reducing the complexity of polynomial multiplications and enabling faster operations across all algorithms.

# 4 Quasi-Cyclic Codes & Syndrome Decoding Problems

The security of the code-based HQC scheme is based on the computational hardness of the decisional quasi-cyclic syndrome decoding (DQCSD) problem with parity and truncation. This problem is a decisional variant of the classical syndrome decoding problem, but for quasi-cyclic (QC) codes. QC codes are a subclass of linear error-correcting codes represented by their block-level cyclic structure, which considerably decreases the size of the keys [31]. The aim of this chapter is to introduce the mathematical framework of QC codes and quasi-cyclic syndrome decoding (QCSD) problems, culminating in the definition of the DQCSD problem with parity and truncation.

## 4.1 Quasi-Cyclic Codes

The code in the DQCSD problem is based on a systematic QC code composed of circulant matrices. QC codes are a generalisation of cyclic codes that are a specific class of linear codes. A linear block code  $C$  of length  $n$  and dimension  $k$ , denoted  $[n, k]$ , is a subspace of  $\mathbb{F}_q^n$  with  $q^k$  elements. Hence,  $C = \{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_{q^k}\}$ , where the elements  $\mathbf{c}_i = (c_{i0}, c_{i1}, \dots, c_{i(n-1)}) \in C$  are called codewords [32, pp. 3–4] [33, p. 1]. The rate or efficiency of the linear block code is the ratio of the number of information bits relative to the total number of bits in the codeword, defined as  $r = \frac{k}{n}$ . A linear block code can be represented by a generator matrix  $\mathbf{G} \in \mathbb{F}_q^{k \times n}$  such that  $C = \{\mathbf{m}\mathbf{G} : \mathbf{m} \in \mathbb{F}_q^k\}$ , where  $\mathbf{m}$  is a message, and a parity-check matrix  $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$  such that  $C = \{\mathbf{c} \in \mathbb{F}_q^n \mid \mathbf{H}\mathbf{c}^\top = \mathbf{0}\}$  [32, p. 4]. Henceforth, linear block codes will be referred to as linear codes for simplicity. Linear codes over  $\mathbb{F}_2$  are called binary codes. All codes in the HQC scheme are defined over  $\mathbb{F}_2$ , thus, unless stated otherwise, all vector spaces and definitions in this chapter are defined over  $\mathbb{F}_2$ .

A linear code  $C$  of length  $n$  and dimension  $k$  with known minimum distance  $d$  is referred to as an  $[n, k, d]$  code. For codewords  $\mathbf{x}, \mathbf{y} \in C$ , the Hamming distance  $d(\mathbf{x}, \mathbf{y})$  is defined as the number of elements in which  $\mathbf{x}$  and  $\mathbf{y}$  differ. The minimum distance of the code  $C$  is defined as the minimum Hamming distance between any two distinct codewords, denoted by

$$d(C) = \min_{\mathbf{x} \neq \mathbf{y}} d(\mathbf{x}, \mathbf{y}).$$

Equivalently, the minimum distance can be defined as the minimum Hamming weight. A Hamming weight of a codeword is the number of its elements that are non-zero, defined as  $\omega(\mathbf{x}) = d(\mathbf{x}, \mathbf{0})$  [32, pp. 7–8]. The minimum distance can be used to compute how many errors a code can correct. In general, a linear code can correct up to  $t = \lfloor \frac{d-1}{2} \rfloor$  errors [31].

The specific class of linear codes called cyclic codes can now be defined.

**Definition 4.1 (Cyclic Code)**

A linear code  $C \subseteq \mathbb{F}_2^n$  is called cyclic if for any codeword  $\mathbf{c}_i = (c_{i0}, c_{i1}, \dots, c_{i(n-1)}) \in C$ , the codeword obtained by a cyclic shift  $\mathbf{c}'_i = (c_{i(n-1)}, c_{i0}, c_{i1}, \dots, c_{i(n-2)})$  is also a codeword in  $C$ . [32, p. 121]

When examining cyclic codes over  $\mathbb{F}_2$ , the codewords of the cyclic code  $\mathbf{c}_i \in C$  can be represented as polynomials in  $R$  of degree at most  $n - 1$  such that the codeword  $\mathbf{c}_i = (c_{i0}, c_{i1}, \dots, c_{i(n-1)}) \in \mathbb{F}_2^n$  for  $i = 1, 2, \dots, q^k$  maps to the polynomial  $c_i(x) = c_{i0} + c_{i1}x + \dots + c_{i(n-1)}x^{n-1} \in \mathbb{F}_2[x]$ . Thus, a cyclic shift corresponds to multiplication by  $x \bmod x^n - 1$ . It follows that every cyclic code is a polynomial code [32, p. 121].

The idea of cyclic codes can be generalised to QC codes, where a code  $C$  has the property that there exists an integer  $s$  such that the shift of a codeword by  $s$  positions is again a codeword [32, p. 131]. These codes are based on circulant matrices. Let  $\mathbf{v} = (v_0, \dots, v_{n-1}) \in \mathbb{F}_2^n$ . The circulant matrix associated to  $\mathbf{v}$  is defined as

$$\text{rot}(\mathbf{v}) = \begin{bmatrix} v_0 & v_1 & \dots & v_{n-1} \\ v_{n-1} & v_0 & \dots & v_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ v_1 & v_2 & \dots & v_0 \end{bmatrix} \in \mathbb{F}_2^{n \times n}, \quad (4.1)$$

with each row being a single cyclic shift of the previous one. The identity matrix  $\mathbf{I}_n$  is an example of a circulant matrix [32, p. 376].

**Definition 4.2 (Quasi-Cyclic Code)**

Let  $s$  be a positive integer. A linear code  $C \subseteq \mathbb{F}_2^{sn}$  is called QC of index  $s$  if, for any codeword  $\mathbf{c}_i = (\mathbf{c}_{i0}, \mathbf{c}_{i1}, \dots, \mathbf{c}_{i(s-1)}) \in C$ , with every  $\mathbf{c}_{ij} \in \mathbb{F}_2^n$ , the codeword obtained by applying a simultaneous circular shift as in Definition 4.1 to each block  $\mathbf{c}_{ij}$  also belongs to  $C$ . [31, p. 12]

To specify the index  $s$ , QC codes are referred to as  $s$ -QC. Cyclic codes are QC codes with  $s = 1$  [32, pp. 131–132]. With  $s = 2$ , QC codes are called double circulant codes. A double circulant code  $[2n, n, d]$  is of even length [32, p. 132]. One of the two codes used in the HQC scheme is a double circulant code in systematic form [31].

## 4.2. Quasi-Cyclic Syndrome Decoding Problems

### Definition 4.3 (Systematic Quasi-Cyclic Code)

A systematic QC code  $[sn, n]$  of index  $s$  and rate  $1/s$  is a QC code with a parity-check matrix of the form:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_n & 0 & \cdots & 0 & \mathbf{A}_0 \\ 0 & \mathbf{I}_n & & & \mathbf{A}_1 \\ & & \ddots & & \vdots \\ 0 & & \cdots & \mathbf{I}_n & \mathbf{A}_{s-2} \end{bmatrix} \in \mathbb{F}_2^{(sn-n) \times sn},$$

where  $\mathbf{A}_0, \mathbf{A}_1, \dots, \mathbf{A}_{s-2}$  are circulant  $n \times n$  matrices.

[31, p. 12]

The definitions of systematic QC codes of index  $s$  can be generalised to all rates  $\ell/s$  for  $\ell = 1, 2, \dots, s-1$ , but in the HQC scheme, only the systematic QC codes of rates  $1/2$  and  $1/3$  are used [31, p. 12].

## 4.2 Quasi-Cyclic Syndrome Decoding Problems

This section describes the hard problems relevant to HQC, all of which are variants of the decoding problem, which consists of finding the closest codeword to a given vector. For linear codes, the problem remains equivalent whether the received vector or its syndrome is given [31]. Let  $C \subseteq \mathbb{F}_2^n$  be a binary code with length  $n$  and dimension  $k$ . Given a parity-check matrix  $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$  and a received codeword  $\mathbf{r} \in \mathbb{F}_2^n$ , the syndrome of  $\mathbf{r}$  is defined as  $\mathbf{s} = \mathbf{H}\mathbf{r}^\top \in \mathbb{F}_2^{n-k}$  [33, p. 4]. The syndrome plays an important role in the syndrome decoding problem.

### Definition 4.4 (Syndrome Decoding Problem)

Let  $n$ ,  $k$ , and  $t$  be positive integers. Given a parity-check matrix  $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$  and a syndrome  $\mathbf{s} \in \mathbb{F}_2^{n-k}$ , the syndrome decoding problem asks to find an error vector  $\mathbf{e} \in \mathbb{F}_2^n$  such that

$$\begin{aligned} \omega(\mathbf{e}) &= t, \\ \mathbf{H}\mathbf{e}^\top &= \mathbf{s}, \end{aligned}$$

where  $\omega(\mathbf{e})$  is the Hamming weight of  $\mathbf{e}$ .

[33, pp. 8–9]

The syndrome of a received codeword  $\mathbf{r} = \mathbf{c} + \mathbf{e}$  is the same as the syndrome of the error vector  $\mathbf{e}$ , since

$$\mathbf{H}\mathbf{r}^\top = \mathbf{H}(\mathbf{c} + \mathbf{e})^\top = \mathbf{H}\mathbf{c}^\top + \mathbf{H}\mathbf{e}^\top = \mathbf{0} + \mathbf{H}\mathbf{e}^\top = \mathbf{H}\mathbf{e}^\top.$$

## 4.2. Quasi-Cyclic Syndrome Decoding Problems

If  $\mathbf{r}$  is a valid codeword,  $\mathbf{s} = \mathbf{0}$  [33, p. 4]. As the HQC cryptosystem uses QC codes, the following definition describes the QCSD distribution [31, p. 13].

### Definition 4.5 (Quasi-Cyclic Syndrome Decoding Distribution)

Let  $n$ ,  $s$ , and  $t$  be positive integers. The QCSD distribution is the distribution over  $(\mathbf{H}, \mathbf{s})$ , where a parity-check matrix  $\mathbf{H} \in \mathbb{F}_2^{(sn-n) \times sn}$  of a systematic QC code  $C$  of index  $s$  and rate  $1/s$  is sampled uniformly at random, and an error vector  $\mathbf{e} = (\mathbf{e}_0, \mathbf{e}_1, \dots, \mathbf{e}_{s-1}) \in \mathbb{F}_2^{sn}$ , with  $\mathbf{e}_i \in \mathbb{F}_2^n$ , is sampled uniformly at random, such that

$$\begin{aligned} \omega(\mathbf{e}_i) &= t \quad \forall i \in [0, s-1], \\ \mathbf{H}\mathbf{e}^\top &= \mathbf{s}. \end{aligned}$$

The distribution outputs the pair  $(\mathbf{H}, \mathbf{s})$ .

[31, p. 13]

The QCSD distribution can be referred to as  $s$ -QCSD to specify the index  $s$ . The security in HQC is based on the assumption that it is computationally hard to distinguish the syndrome of a random error vector with a specific weight generated from a QC code from a uniform random vector. The DQCSD problem is the decisional variant of the SD problem for QC codes [31, p. 13].

### Definition 4.6 (Decisional Quasi-Cyclic Syndrome Decoding Problem)

Let  $n$ ,  $s$ , and  $t$  be positive integers. Given a pair  $(\mathbf{H}, \mathbf{s}) \in \mathbb{F}_2^{(sn-n) \times sn} \times \mathbb{F}_2^{sn-n}$ , the DQCSD problem asks to decide with non-negligible advantage whether  $(\mathbf{H}, \mathbf{s})$  came from:

1. The distribution in Definition 4.5, or
2. The uniform distribution over  $\mathbb{F}_2^{(sn-n) \times sn} \times \mathbb{F}_2^{sn-n}$ .

[31, p. 13]

The DQCSD problem is the foundational problem on which the security of the HQC scheme relies. In this problem, an adversary aims to distinguish whether  $(\mathbf{H}, \mathbf{s})$  comes from the QCSD or uniform distribution. The adversary's advantage measures how much their success exceeds random guessing. A non-negligible advantage means the adversary's probability of success is noticeably higher than random guessing. If the adversary can distinguish between the two distributions, information about the error vector or ciphertext could be gained, which could potentially lead to decrypting the secret message. The DQCSD problem is considered secure, as it is assumed that no adversary can achieve a non-negligible advantage, thus ensuring the confidentiality of the secret message [31].

Building on the DQCSD problem, HQC also addresses the need to prevent trivial distinguishing advantages by ensuring that syndromes have controlled parity. The parity of a syndrome

#### 4.2. Quasi-Cyclic Syndrome Decoding Problems

is the sum of its bits modulo 2 such that the parity is 0 when the syndrome has an even number of 1s and 1 when it has an odd number of 1s. If the parity of the syndrome leaks or is predictable, it could become a side-channel that lets an attacker distinguish information [31]. Hence, for binary vectors  $\mathbf{h}$  of length  $n$  and a parity  $b_1 \in \{0, 1\}$ , the following finite set is defined:

$$\mathbb{F}_{2,b_1}^n = \{\mathbf{h} \in \mathbb{F}_2^n \mid \mathbf{h}(1) \bmod 2 = b_1\},$$

where  $\mathbf{h}(1)$  denotes the evaluation of the polynomial corresponding to  $\mathbf{h}$  at  $x = 1$ . Similarly, for matrices, the finite sets are defined as

$$\begin{aligned} \mathbb{F}_{2,b_1}^{n \times 2n} &= \left\{ \mathbf{H} = \begin{pmatrix} \mathbf{I}_n & \text{rot}(\mathbf{h}) \end{pmatrix} \in \mathbb{F}_2^{n \times 2n} \mid \mathbf{h} \in \mathbb{F}_{2,b_1}^n \right\}, \\ \mathbb{F}_{2,b_1,b_2}^{2n \times 3n} &= \left\{ \mathbf{H} = \begin{pmatrix} \mathbf{I}_n & 0 & \text{rot}(\mathbf{h}_1) \\ 0 & \mathbf{I}_n & \text{rot}(\mathbf{h}_2) \end{pmatrix} \in \mathbb{F}_2^{2n \times 3n} \mid \mathbf{h}_1 \in \mathbb{F}_{2,b_1}^n, \mathbf{h}_2 \in \mathbb{F}_{2,b_2}^n \right\}, \end{aligned}$$

where  $\mathbf{I}_n$  is the  $n \times n$  identity matrix,  $b_2 \in \{0, 1\}$  is a parity, and  $\text{rot}(\mathbf{h})$ ,  $\text{rot}(\mathbf{h}_1)$ , and  $\text{rot}(\mathbf{h}_2)$  are circulant matrices generated from  $\mathbf{h}$ ,  $\mathbf{h}_1$ ,  $\mathbf{h}_2$ , respectively [31, p. 14]. The parity constraint is essential for preventing certain side-channel or distinguishing attacks [31].

With the additional parity constraint, the DQCSD problem with  $s = 2$  can be modified.

##### Definition 4.7 (2-QCSD With Parity Distribution)

Let  $n, t, b_1$  be positive integers and  $b_2 = t + b_1 \cdot t \bmod 2$ . The 2-QCSD with parity distribution is the distribution over  $(\mathbf{H}, \mathbf{s})$ , where a parity-check matrix  $\mathbf{H} \in \mathbb{F}_{2,b_1}^{n \times 2n}$  of a systematic QC code  $C$  of index 2 and rate  $1/2$  is sampled uniformly at random, and an error vector  $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2) \in \mathbb{F}_2^{2n}$ , with  $\mathbf{e}_1, \mathbf{e}_2 \in \mathbb{F}_2^n$ , sampled uniformly at random, such that

$$\begin{aligned} \omega(\mathbf{e}_1) &= \omega(\mathbf{e}_2) = t, \\ \mathbf{H}\mathbf{e}^\top &= \mathbf{s}. \end{aligned}$$

The 2-QCSD with parity distribution outputs the pair  $(\mathbf{H}, \mathbf{s})$ .

[31, p. 14]

##### Definition 4.8 (2-DQCSD With Parity Problem)

Let  $n, t, b_1$  be positive integers and  $b_2 = t + b_1 \cdot t \bmod 2$ . Given a pair  $(\mathbf{H}, \mathbf{s}) \in \mathbb{F}_{2,b_1}^{n \times 2n} \times \mathbb{F}_{2,b_2}^n$ , the 2-DQCSD with parity problem asks to decide with non-negligible advantage whether  $(\mathbf{H}, \mathbf{s})$  came from:

1. The distribution in Definition 4.7, or
2. The uniform distribution over  $\mathbb{F}_{2,b_1}^{n \times 2n} \times \mathbb{F}_{2,b_2}^n$ .

[31, p. 14]

#### 4.2. Quasi-Cyclic Syndrome Decoding Problems

To prevent algebraic attacks, the code length  $n$  is desired to be a primitive prime number such that  $x^n - 1$  has only two irreducible factors modulo  $q$ . However, since the encoded message length in the HQC scheme is not prime, an ambient length  $n$  is chosen. This introduces excess bits  $\ell$ , which are truncated to restore the original length. The truncation operation  $\text{truncate}(\mathbf{v}, \ell) = (v_0, v_1, \dots, v_{n-\ell-1})$  for  $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$  returns the first  $n - \ell$  bits of  $\mathbf{v}$  and discards the last  $\ell$  bits. This leads to a modified version of the DQCSD with parity problem [31, p. 14].

##### Definition 4.9 (3-QCSD With Parity & Truncation Distribution)

Let  $n, t, b_1, b_2, \ell$  be positive integers and  $b_3 = t + b_1 \cdot t \bmod 2$ . The 3-QCSD with parity and truncation distribution is the distribution over  $(\mathbf{H}, \mathbf{s})$ , where a parity-check matrix  $\mathbf{H} \in \mathbb{F}_{2, b_1, b_2}^{2n \times 3n}$  of a systematic QC code  $C$  of index 3 and rate  $1/3$  is sampled uniformly at random, and an error vector  $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3) \in \mathbb{F}_2^{3n}$ , with  $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3 \in \mathbb{F}_2^n$ , sampled uniformly at random, such that

$$\begin{aligned}\omega(\mathbf{e}_1) &= \omega(\mathbf{e}_2) = \omega(\mathbf{e}_3) = t, \\ \mathbf{H}\mathbf{e}^\top &= \mathbf{s},\end{aligned}$$

where  $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2) \in \mathbb{F}_2^{2n}$ , with  $\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{F}_2^n$ .

The 3-QCSD with parity and truncation distribution outputs the pair  $(\mathbf{H}, (\mathbf{s}_1, \text{truncate}(\mathbf{s}_2, \ell))) \in \mathbb{F}_{2, b_1, b_2}^{2n \times 3n} \times (\mathbb{F}_{2, b_3}^n \times \mathbb{F}_2^{n-\ell})$ . [31, p. 14]

##### Definition 4.10 (3-DQCSD With Parity & Truncation Problem)

Let  $n, t, b_1, b_2, \ell$  be positive integers and  $b_3 = t + b_1 \cdot t \bmod 2$ . Given a pair  $(\mathbf{H}, (\mathbf{s}_1, \text{truncate}(\mathbf{s}_2, \ell)))$ , the 3-DQCSD with parity and truncation problem asks to decide with non-negligible advantage whether  $(\mathbf{H}, (\mathbf{s}_1, \text{truncate}(\mathbf{s}_2, \ell)))$  came from:

1. The distribution in Definition 4.9, or
2. The uniform distribution over  $\mathbb{F}_{2, b_1, b_2}^{2n \times 3n} \times (\mathbb{F}_{2, b_3}^n \times \mathbb{F}_2^{n-\ell})$ .

[31, p. 15]

The hardness of the 3-DQCSD with parity and truncation problem underpins the IND-CPA security of HQC.

## 5 Hamming Quasi-Cyclic

HQC was selected by NIST in March 2025 for standardisation. In approximately two years, as of the writing of this thesis, NIST will publish the final standard version based on HQC. The standardisation of HQC will be the second PQC KEM after ML-KEM. Structurally, HQC is similar to LWE-based cryptosystems like ML-KEM, which was introduced in Chapter 3. It employs two stages: a PKE scheme, called HQC.PKE, and a KEM scheme, called HQC.KEM. Since HQC.PKE only achieves IND-CPA security, a variant of the FO transform is applied to obtain the IND-CCA2-secure HQC.KEM scheme [8].

### 5.1 Prerequisites

Before presenting the HQC scheme, Reed-Muller (RM) codes, concatenated Reed-Muller and Reed-Solomon (RMRS) codes, as well as the ring structure, codes, and hash functions used in the scheme, are introduced.

#### Reed-Muller Codes

RM codes can be seen as a generalisation of Reed-Solomon (RS) codes. Often, RM codes are binary codes. The two parameters that define an RM code are the number of variables used in the polynomials, denoted  $m$ , and the order of the polynomials, denoted  $r$  [32, p. 34].

##### Definition 5.1 (Binary $r$ th Order Reed-Muller Code)

Let  $m$  be a positive integer and  $r$  a non-negative integer with  $r \leq m$ . The binary  $r$ th order RM code, denoted  $RM(r, m)$ , is the set of all vectors  $\mathbf{f} = (f(\mathbf{v}_1), f(\mathbf{v}_2), \dots, f(\mathbf{v}_{2^m}))$  with  $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{2^m} \in \mathbb{F}_2^m$ , where  $f \in \mathbb{F}_2[x_1, x_2, \dots, x_m]$  is a polynomial of degree at most  $r$ . The binary RM code has the following parameters:

- A code length of  $n = 2^m$ .
- A dimension of  $k = \sum_{i=0}^r \binom{m}{i}$ .
- A minimum distance of  $d = 2^{m-r}$ . [34, p. 373]

In the HQC scheme, a binary first-order RM code is used. A binary first-order RM code is a code with  $r = 1$ , denoted  $RM(1, m)$  [31].

#### Concatenated Reed-Muller & Reed-Solomon Codes

A concatenated RMRS code is used in the encryption and decryption process of HQC. A concatenated code is a technique that combines two or more codes to obtain a new code with improved properties [31].

## 5.1. Prerequisites

### Definition 5.2 (Concatenated Codes)

Let  $C_1$  be an external code  $[n_1, k_1, d_1]$  over  $\mathbb{F}_q^{n_1}$  and let  $C_2$  be an internal code  $[n_2, k_2, d_2]$  over  $\mathbb{F}_2^{n_2}$ , with  $q = 2^{k_2}$ . By using a bijection between the elements of  $\mathbb{F}_q^{n_1}$  and the codewords of the internal code  $C_2$ , the following transformation is obtained

$$\mathbb{F}_q^{n_1} \longrightarrow \mathbb{F}_2^{n_1 n_2}.$$

Thus, the external code  $C_1$  is transformed into a binary code  $[n_1 n_2, k_1 k_2, \geq d_1 d_2]$ , referred to as a concatenated code. [31, p. 21]

In HQC, the external code is a shortened RS code, and the internal code is a duplicated first-order RM code. Shortening RS codes involves reducing their block length by removing information symbols. All the symbols which are dropped need not to be transmitted and at the receiving end can be reinserted. The RS codes are defined over  $\mathbb{F}_{2^8}$ , where  $q = 2^8 = 256$ , and have the following parameters:

- A block length of  $n_1 = q - 1 = 256 - 1 = 255$ .
- A dimension of  $k_1 = n_1 - 2\delta = 255 - 2\delta$ .
- A minimum distance of  $d_1 = 2\delta + 1$ ,

where  $\delta$  is the error-correcting capability of the code [31]. The specific parameter values are outlined in Table 5.1.

RS Code	$n_1$	$k_1$	$\delta$
RS-1	255	225	15
RS-2	255	223	16
RS-3	255	197	29
RS-S1	46	16	15
RS-S2	56	24	16
RS-S3	90	32	29

**Table 5.1:** Code lengths, dimension, and error-correcting capability of original and shortened RS codes. The abbreviation 'S' refer to the shortening code. Information is from [31].

The shortened RS codes are obtained by subtracting 209, 199, and 165 from  $n_1$  and  $k_1$  of RS-1, RS-2, and RS-3, respectively, while maintaining the same error-correcting capability. Encoding and decoding of shortened RS codes can be performed by treating it as a standard RS code. For more details on encoding and decoding of shortened RS codes, see [31, pp. 23–25] and [33, pp. 51–53, 57–60].

### 5.1. Prerequisites

The internal code used in HQC is a first-order RM code, denoted  $RM(1, 7)$ . This code has the following parameters:

- A code length of  $n_2 = 2^m = 2^7 = 128$ .
- A dimension of  $k_2 = \sum_{i=0}^{r-1} \binom{m}{i} = 1 + m = 1 + 7 = 8$ .
- A minimum distance of  $d_2 = 2^{m-r} = 2^{7-1} = 64$ .

To form the duplicated RM code, each codeword from  $RM(1, 7)$  is duplicated a certain number of times [31], as specified in Table 5.2.

Scheme	RM Code	Multiplicity	Duplicated RM Code
HQC-128	[128, 8, 64]	3	[384, 8, 192]
HQC-192	[128, 8, 64]	5	[640, 8, 320]
HQC-256	[128, 8, 64]	5	[640, 8, 320]

**Table 5.2:** Duplicated RM code parameters for different security categories for HQC.

For a detailed explanation of encoding and decoding of duplicated RM codes, see [31, pp. 25–26] and [34, pp. 419–425].

Encoding a concatenated code, a message  $\mathbf{m} \in \mathbb{F}_2^{k_1 k_2}$  is first encoded using the shortened RS code, yielding  $\mathbf{m}_1 \in \mathbb{F}_2^{n_1}$ . Each coordinate  $m_{1,i}$  of  $\mathbf{m}_1$  is then encoded using the  $RM(1, 7)$  code. This codeword is then duplicated depending on the multiplicity of the RM code, producing  $\tilde{\mathbf{m}}_{1,i} \in \mathbb{F}_2^{n_2}$ . Finally, the resulting encoded message is  $\mathbf{mG} = \tilde{\mathbf{m}} = (\tilde{\mathbf{m}}_{1,0} \dots, \tilde{\mathbf{m}}_{1,n_1-1}) \in \mathbb{F}_2^{n_1 n_2}$ . While the concatenated code has length  $n_1 n_2$ , calculations are carried out in an ambient space of length  $n$ , the smallest primitive prime greater than  $n_1 n_2$ . The remaining  $\ell = n - n_1 n_2$  bits are truncated. Decoding proceeds in two stages, where the internal code is decoded first, then the external code. For a detailed explanation of decoding of the concatenated RMRS codes, see [31, pp. 24–26], [33, pp. 51–53, 57–60], and [34, pp. 419–425].

### The Ring Structure in HQC

In the HQC cryptosystem, several key vectors are represented as binary vectors whose elements correspond to the coefficients of a polynomial. Therefore, a brief introduction to the underlying ring in which these polynomials are defined is provided.

## 5.1. Prerequisites

### Definition 5.3 (The Ring $R_2$ )

Let  $n$  be a positive integer and  $\mathbb{F}_2$  denote the binary finite field. Then, the quotient ring

$$R_2 = \mathbb{F}_2[x]/(x^n - 1),$$

consists of polynomials with coefficients in  $\mathbb{F}_2$ , where addition and multiplication are performed modulo  $x^n - 1$ . [31]

Each element in  $R_2$  can be uniquely represented by a polynomial of degree at most  $n - 1$ , corresponding to a binary vector of length  $n$ . Some of these vectors correspond to polynomials with a fixed Hamming weight.

### Definition 5.4 (The Subset $R_w$ )

Let  $w$  be a positive integer and  $R_2 = \mathbb{F}_2[x]/(x^n - 1)$ . For  $\mathbf{v} \in R_2$ , with Hamming weight  $\omega(\mathbf{v})$ , the subset

$$R_w = \{\mathbf{v} \in R_2 \mid \omega(\mathbf{v}) = w\},$$

consists of polynomials in  $R_2$  with exactly  $w$  non-zero coefficients. [31, pp. 10, 16]

The subset  $R_w$  is essential in HQC, as it defines the structure of shared secret keys and error vectors.

## Code Structure in HQC

HQC uses two types of codes:

1. An  $[n, k]$  linear code  $C$ , generated by the generator matrix  $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ , which can correct at least  $\delta$  errors. It is assumed that there are both an efficient encoding and decoding algorithm.
2. A random double-circulant  $[2n, n]$  code with a parity-check matrix  $\mathbf{H} = (\mathbf{I}_n, \text{rot}(\mathbf{h}))$ , where  $\mathbf{I}_n$  is a  $n \times n$  identity matrix and  $\text{rot}(\mathbf{h})$  is a circulant matrix as defined in (4.1), generated from a vector  $\mathbf{h} \in R_2$ . Here,  $R_2$  is a polynomial ring, as defined in Definition 5.3, for  $n$  prime such that  $x^n - 1$  has only two irreducible factors modulo 2 [8].

The first code is a concatenated RMRS code  $[n_1 n_2, k_1 k_2]$ , generated by the generator matrix  $\mathbf{G} \in \mathbb{F}_2^{k_1 k_2 \times n_1 n_2}$ . It is assumed that the generator matrix as well as the encoding and decoding algorithms for  $C$  are publicly known. For the second code, it is assumed that no one knows the decoding algorithm [31].

## 5.2. HQC.PKE

### Hash Functions

HQC uses two hash functions, denoted as

$$\mathcal{G} : \{0, 1\}^* \rightarrow \{0, 1\}^{512}, \quad (5.1)$$

$$\mathcal{K} : \{0, 1\}^* \rightarrow \{0, 1\}^{512}. \quad (5.2)$$

Both are based on SHAKE256 with 512 bits output length. SHAKE256 is an extendable-output function from the SHA-3 family, which means it can generate outputs of arbitrary length, unlike SHA-256, which always produces a fixed 256-bit output regardless of the input size [35]. The implementation details of these functions are beyond the scope of this thesis. For further information, see [31].

Throughout the HQC scheme, vectors that are sampled uniformly at random are generated using SHAKE256-based functions. One such function is used to derive seeds, which are then used as input to another SHAKE256-based function that generates the vectors, depending on the specific context within the HQC scheme. For further details on the SHAKE256-functions, see [31].

## 5.2 HQC.PKE

The IND-CPA secure HQC.PKE scheme consists of the following algorithms:

1. *Key generation*, which takes the input parameters and generates the public encryption key and private decryption key.
2. *Encryption*, which takes a public encryption key and a message as input and produces a ciphertext.
3. *Decryption*, which takes a private decryption key and ciphertext as input and attempts to reconstruct the original message.

The two communicating parties will be referred to as Alice and Bob. The following subsections provide a description of the algorithms in HQC.PKE.

### Key Generation

For Alice and Bob to communicate securely, they need to generate public encryption and private decryption keys. Therefore, Alice uses the input parameters to generate a public encryption key,  $pk$ , and a corresponding private decryption key,  $sk$ . She achieves this by first sampling  $\mathbf{h} \in R_2$  uniformly at random. The parity-check element  $\mathbf{h}$  forms a part of the public encryption key.

Alice then generates  $\mathbf{G} \in \mathbb{F}_2^{k_1 k_2 \times n_1 n_2}$  for a decodable concatenated RMRS code  $C$ , as defined in Section 5.1. The generator matrix  $\mathbf{G}$  is used to encode messages and enables error-correction during decryption. To construct the private decryption key, Alice samples uniformly at random the secret pair  $(\mathbf{x}, \mathbf{y}) \in R_w \times R_w$ , where  $R_w$  is the set of vectors defined in Definition 5.4 [31].

### 5.3. HQC.KEM

Finally, Alice computes her public encryption and private decryption keys as

$$\begin{aligned} pk &= (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y}), \\ sk &= (\mathbf{x}, \mathbf{y}). \end{aligned} \tag{5.3}$$

The public encryption key is sent to Bob, while the private decryption key remains with Alice [31].

#### Encryption

Bob wants to send a message  $\mathbf{m} \in \mathbb{F}_2^{k_1 k_2}$  to Alice. He does that by encrypting the message  $\mathbf{m}$  using Alice's public encryption key  $pk$ . To achieve this, Bob first encodes the message  $\mathbf{m}$  into a codeword  $\mathbf{mG}$ , as explained in Section 5.1. Then, he generates noise by sampling uniformly at random the error vector  $\mathbf{e} \in R_{w_e}$ , where  $w_e$  denotes the Hamming weight of  $\mathbf{e}$ , and  $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2) \in R_{w_r} \times R_{w_r}$ , where  $\mathbf{r}_1$  and  $\mathbf{r}_2$  each have Hamming weight  $w_r$  [8]. Then, Bob computes

$$\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2, \tag{5.4}$$

$$\mathbf{v} = \text{truncate}(\mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}, \ell), \tag{5.5}$$

where  $\text{truncate}(\mathbf{a}, \ell)$  is the truncation of a vector  $\mathbf{a}$  by the number of  $\ell$  bits. Lastly, Bob sends the ciphertext  $\mathbf{c} = (\mathbf{u}, \mathbf{v})$  to Alice [31].

#### Decryption

Alice receives the ciphertext  $\mathbf{c}$  from Bob. To decrypt the ciphertext, Alice uses her private decryption key  $sk$  to compute  $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ . Using the decoding algorithm for the concatenated RMRS code  $C$ , denoted  $\text{Decode}$ , Alice can compute

$$\mathbf{m}' = \text{Decode}(\mathbf{v} - \mathbf{u} \cdot \mathbf{y}). \tag{5.6}$$

If the decoding is successful, meaning the number of errors does not exceed the decoder's error-correction capacity  $\delta$ , Alice obtains the original message  $\mathbf{m}$  [31].

The correctness of the scheme relies on the decoding capability of the code  $C$ . Specifically, assuming  $\text{Decode}$  correctly recovers  $\mathbf{m}$  if  $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$  contains at most  $\delta$  errors. This is guaranteed whenever the following conditions hold [31]:

$$\begin{aligned} \omega(\mathbf{s} \cdot \mathbf{r}_2 - \mathbf{u} \cdot \mathbf{y} + \mathbf{e}) &\leq \delta, \\ \omega((\mathbf{x} + \mathbf{h} \cdot \mathbf{y}) \cdot \mathbf{r}_2 - (\mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2) \cdot \mathbf{y} + \mathbf{e}) &\leq \delta, \\ \omega(\mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}) &\leq \delta. \end{aligned}$$

### 5.3 HQC.KEM

The IND-CCA2 secure HQC.KEM scheme is constructed from HQC.PKE using a variant of the FO transform. The HQC.KEM scheme consists of the following algorithms [31]:

### 5.3. HQC.KEM

1. *Key Generation*, which takes the input parameters and generates the public encapsulation and private decapsulation keys.
2. *Encapsulation*, which takes the public encapsulation key as input and generates a shared secret key, a ciphertext, and a salt value.
3. *Decapsulation*, which takes the private decapsulation key, the ciphertext, and the salt value as input and attempts to reconstruct the shared secret key.

The following subsections provide a description of the algorithms in HQC.KEM, with the two communicating parties referred to as Alice and Bob.

#### Key Generation

Alice employs the HQC.PKE key generation algorithm to derive a public encryption key  $pk$  and a private decryption key  $sk$ . The public encryption key serves as the public encapsulation key. To construct the private decapsulation key, a uniformly random vector  $\sigma \in \mathbb{F}_2^{k_1 k_2}$  is sampled to enable implicit rejection during decapsulation. The private decapsulation key is then formulated as

$$sk = (\mathbf{x}, \mathbf{y}, \sigma), \quad (5.7)$$

where  $sk \in R_w \times R_w \times \mathbb{F}_2^{k_1 k_2}$  [31]. Alice then sends the public encapsulation key to Bob.

#### Encapsulation

Given Alice's public encapsulation key  $pk$ , Bob generates uniformly at random a message  $\mathbf{m} \in \mathbb{F}_2^{k_1 k_2}$  that will serve as a seed to derive the shared secret key. He also samples uniformly at random a salt,  $salt \in \mathbb{F}_2^{128}$ , which is a non-secret value that is added to ensure uniqueness and prevent deterministic outputs. Using  $\mathbf{m}$ , the first 32 bytes of the public encapsulation key  $pk$ , and the salt, he computes the randomness

$$\theta = \mathcal{G}(\mathbf{m}, \text{firstBytes}(pk, 32), salt), \quad (5.8)$$

where  $\mathcal{G}$  is the hash function defined in (5.1) and  $\text{firstBytes}(\cdot)$  extracts the first specified number of bytes from a byte string. Using the same steps as in the HQC.PKE encryption algorithm but with  $\theta$  as the seed, Bob generates  $(\mathbf{e}, \mathbf{r}_1, \mathbf{r}_2)$  uniformly at random with the specified Hamming weights  $w_{\mathbf{e}}$  and  $w_{\mathbf{r}}$ , such that  $\omega(\mathbf{e}) = w_{\mathbf{e}}$  and  $\omega(\mathbf{r}_1) = \omega(\mathbf{r}_2) = w_{\mathbf{r}}$ . Then, Bob can compute the ciphertext  $\mathbf{c} = (\mathbf{u}, \mathbf{v})$ , where  $\mathbf{u}$  and  $\mathbf{v}$  are defined as in the HQC.PKE encryption algorithm. Finally, Bob computes the shared secret key

$$K = \mathcal{K}(\mathbf{m}, \mathbf{c})$$

using the hash function  $\mathcal{K}$  as defined in (5.2), and sends  $(\mathbf{c}, salt)$  to Alice [31].

#### 5.4. Parameter Sets

### Decapsulation

To recover the shared secret key  $K$  from the ciphertext  $\mathbf{c}$ , Alice uses her private decapsulation key  $sk$ . She first computes

$$\mathbf{m}' = \text{Decrypt}(sk, \mathbf{c}),$$

by applying the HQC.PKE decryption algorithm. Then, she computes the randomness

$$\theta' = \mathcal{G}(\mathbf{m}', \text{firstBytes}(pk, 32), salt), \quad (5.9)$$

and re-encrypts  $\mathbf{m}'$  using  $\theta'$  as the seed to obtain  $(\mathbf{e}', \mathbf{r}'_1, \mathbf{r}'_2)$  such that  $\omega(\mathbf{e}') = w_{\mathbf{e}}$  and  $\omega(\mathbf{r}'_1) = \omega(\mathbf{r}'_2) = w_{\mathbf{r}}$ . The reason for re-encrypting the message after the decryption is to verify if the received ciphertext  $\mathbf{c}$  is equal to the newly computed one  $\mathbf{c}'$ . The shared secret key can only be shared if the equality is verified. Therefore, Alice calculates

$$\mathbf{u}' = \mathbf{r}'_1 + \mathbf{h} \cdot \mathbf{r}'_2, \quad (5.10)$$

$$\mathbf{v}' = \text{truncate}(\mathbf{m}'\mathbf{G} + \mathbf{s} \cdot \mathbf{r}'_2 + \mathbf{e}', \ell), \quad (5.11)$$

such that she gets  $\mathbf{c}' = (\mathbf{u}', \mathbf{v}')$ . If  $\mathbf{c} = \mathbf{c}'$ , the shared secret key is computed as

$$K' = \mathcal{K}(\mathbf{m}, \mathbf{c}).$$

Otherwise, if  $\mathbf{c} \neq \mathbf{c}'$ , decryption failed and a random shared secret key is computed as

$$\bar{K} = \mathcal{K}(\boldsymbol{\sigma}, \mathbf{c}),$$

using implicit rejection [31].

### 5.4 Parameter Sets

HQC consists of the parameters  $n_1$ ,  $n_2$ ,  $n$ ,  $k_1$ ,  $k_2$ ,  $w$ ,  $w_{\mathbf{r}}$ ,  $w_{\mathbf{e}}$ , and  $\ell$ , with three parameter sets in total, corresponding to a specific security category. The parameters  $n_1$  and  $n_2$  denote the length of the RS and RM codes, respectively. Their product,  $n_1n_2$ , gives the length of the concatenated RMRS code. Since  $n_1n_2$  is not prime, the prime integer  $n$  is used as the smallest primitive prime greater than  $n_1n_2$ . The parameters  $k_1$  and  $k_2$  denote the dimension of the RS and RM codes, respectively. The parameter  $w$  defines the Hamming weight of the secret vectors  $\mathbf{x}$  and  $\mathbf{y}$ . Similarly,  $w_{\mathbf{r}}$  and  $w_{\mathbf{e}}$  specify the Hamming weights of the vectors  $(\mathbf{r}_1, \mathbf{r}_2)$ , and  $\mathbf{e}$ , respectively. Lastly,  $\ell = n - n_1n_2$  denotes the length of the truncated part of  $\mathbf{v}$  in the ciphertext [31].

The parameter sets for each of the three security categories are listed in Table 5.3.

#### 5.4. Parameter Sets

Parameter Set	$n_1$	$n_2$	$n$	$k_1$	$k_2$	$w$	$w_r = w_e$	$\ell$	Security Category
HQC-128	46	384	17669	16	8	66	75	5	1
HQC-192	56	640	35851	24	8	100	114	11	3
HQC-256	90	640	57637	32	8	131	149	37	5

**Table 5.3:** HQC parameter sets for security category 1, 3, and 5. The information is sourced from [31].

The size of keys and ciphertexts depends on the selected parameter set. Higher security categories require longer keys and ciphertexts to strengthen security. Detailed sizes are shown in Table 5.4. The size of the shared secret key remains constant at 64 across all parameter sets.

Parameter Set	Public Encapsulation Key	Private Decapsulation Key	Shared Secret Key	Ciphertext
HQC-128	2249	56	64	4433
HQC-192	4522	64	64	8978
HQC-256	7245	72	64	14421

**Table 5.4:** Key and ciphertext sizes in bytes for each HQC parameter set [8]. Note that the private decapsulation key sizes differ slightly between [8] and [31]. It should also be noted that the ciphertext sizes listed in [31] are 64 bytes larger than the sizes computed in this thesis.

The reliability of HQC is quantified through its decapsulation failure rate. The decapsulation failure rates for different parameter sets are presented in Table 5.5.

Parameter Set	Decapsulation Failure Rate
HQC-128	$< 2^{-128}$
HQC-192	$< 2^{-192}$
HQC-256	$< 2^{-256}$

**Table 5.5:** Decapsulation failure rates for different HQC parameter sets, as reported in [31].

# 6 Comparative Complexity Analysis of ML-KEM & HQC

Understanding how ML-KEM and HQC scale helps determine their suitability for real-world applications, particularly in resource-constrained devices. Even when secure and correct, an algorithm's efficiency can make the difference between practicality and infeasibility. Therefore, in this thesis, the computational complexities of ML-KEM and HQC are analysed. The foundation for this analysis is laid in Appendix A. Detailed complexity analyses for ML-KEM and HQC are presented in Appendix B and Appendix C, respectively. This chapter presents a comparative analysis of the time and space complexities of ML-KEM and HQC. The asymptotic behaviour is first examined using big- $\mathcal{O}$  notation, which provides upper bounds on computational cost. This is followed by a review of the corresponding bit-level complexities to give a more concrete and practical view of each scheme.

## 6.1 Time Complexity Comparison

The overall time complexities for the operations in the key generation, encapsulation, and decapsulation algorithms for ML-KEM and HQC, along with the total time complexities for each scheme, are summarised in Table 6.1.

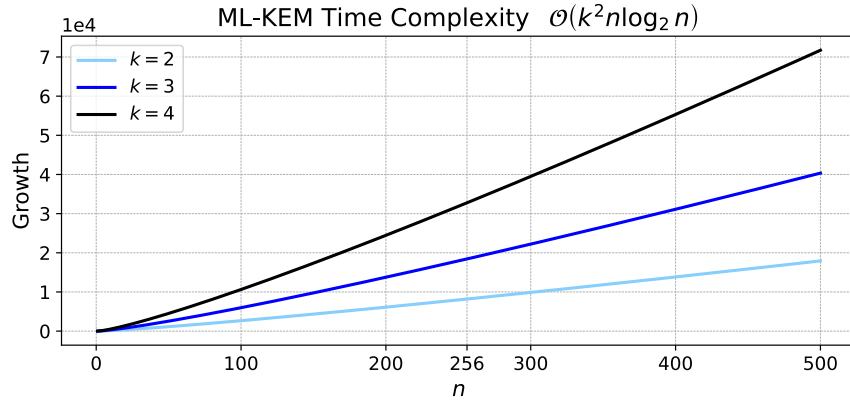
Algorithm	ML-KEM Time Complexity	HQC Time Complexity
Key Generation	$\mathcal{O}(k^2 n \log_2(n))$	$\mathcal{O}(n \log_2(n))$
Encapsulation	$\mathcal{O}(k^2 n \log_2(n))$	$\mathcal{O}(n \log_2(n))$
Decapsulation	$\mathcal{O}(kn \log_2(n))$	$\mathcal{O}(n_1^2 n_2^2)$
Total Complexity	$\mathcal{O}(k^2 n \log_2(n))$	$\mathcal{O}(n_1^2 n_2^2)$

**Table 6.1:** Summary of the time complexities for each algorithm in ML-KEM and HQC.

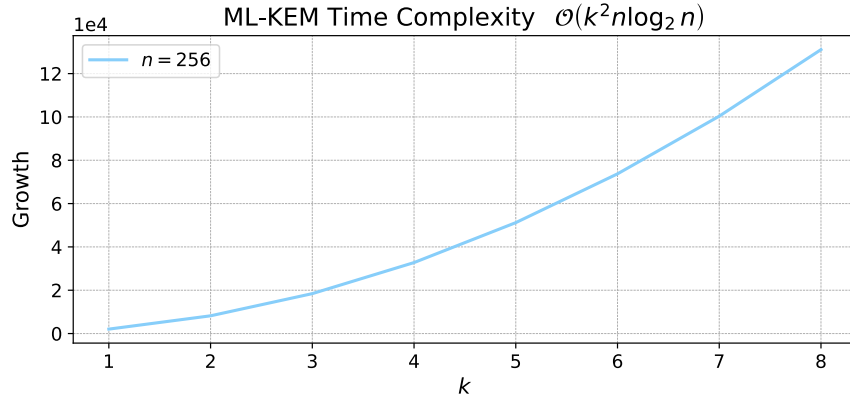
Due to the different parameter sets used in ML-KEM and HQC, their time complexities are plotted separately. In each plot, one parameter is varied while the others are kept fixed in order to observe its effect on the time complexity.

For ML-KEM, the key generation and encapsulation algorithms share the same time complexity of  $\mathcal{O}(k^2 n \log_2(n))$ . Two plots are provided to show how this time complexity grows with respect to  $n$  and  $k$ . Figure 6.1a shows the growth as a function of  $n$ . Figure 6.1b shows the growth as a function of  $k$ .

### 6.1. Time Complexity Comparison



(a) The time complexity growth as a function of  $n$ , for fixed values of  $k = 2, 3, 4$ , corresponding to the parameters listed in Table 3.1. The  $n$  value corresponding to the three security categories in Table 3.1 is indicated on the  $x$ -axis with a line at 256.



(b) The time complexity growth as a function of  $k$ , for a fixed value of  $n = 256$ , corresponding to the value listed in Table 3.1. The  $k$  values corresponding to the three security categories in Table 3.1 are indicated along the  $x$ -axis by a line at 2, 3, and 4.

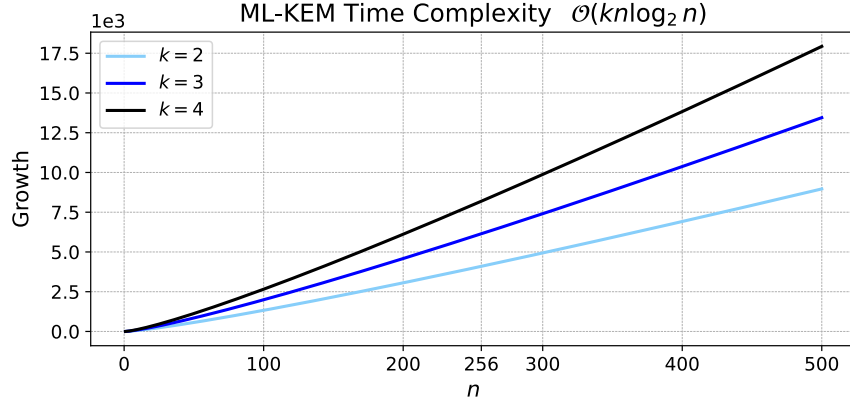
**Figure 6.1:** The time complexity growth of  $\mathcal{O}(k^2 n \log_2(n))$  as a function of  $n$  and  $k$ , respectively.

In Figure 6.1a, the time complexity increases more rapidly for higher values of  $k$ , with the gap between the curves for the three values of  $k$  becoming more pronounced as  $n$  increases. At smaller values of  $n$ , the curves are close, converging as  $n$  approaches zero. At  $n = 256$ , each of the curves yield a time complexity of  $0.8 \cdot 10^4$ ,  $1.8 \cdot 10^4$ , and  $3.3 \cdot 10^4$ , respectively. These computational costs continue to increase as  $n$  is increased, reaching  $1.8 \cdot 10^4$ ,  $4 \cdot 10^4$ , and  $7.2 \cdot 10^4$ , respectively, at  $n = 500$ .

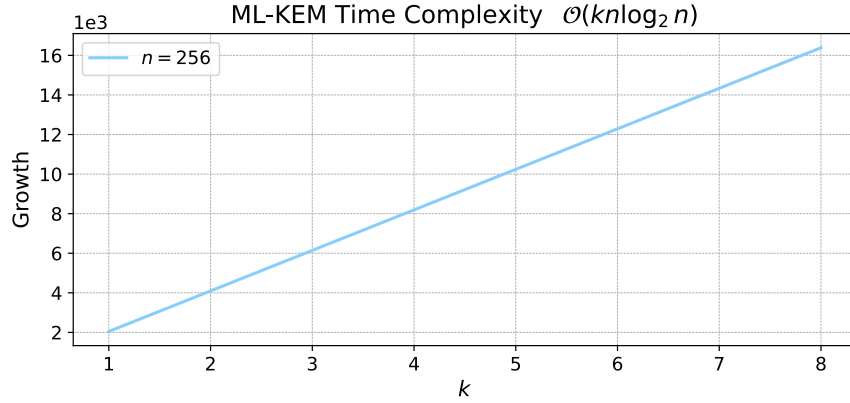
In Figure 6.1b, it can be observed that the time complexity grows quadratic with  $k$ . The time complexity at the three vertical lines that indicate the  $k$  values for the three security categories corresponds to the time complexities observed for Figure 6.1a. When  $k$  is increased to 8, the time complexity becomes  $13.1 \cdot 10^4$ . This is almost four times the value observed at  $k = 4$ , reflecting the quadratic growth in the time complexity. Thus, while higher  $k$  improves security, it significantly increases the computational cost.

## 6.1. Time Complexity Comparison

The decapsulation algorithm for ML-KEM has the time complexity  $\mathcal{O}(kn \log_2(n))$ . Two plots are provided to show how this time complexity grows with respect to  $n$  and  $k$ . Figure 6.2a shows the growth as a function of  $n$ . Figure 6.2b shows the growth as a function of  $k$ .



(a) The time complexity growth as a function of  $n$ , for fixed values of  $k = 2, 3, 4$ , corresponding to the parameters listed in Table 3.1. The  $n$  value corresponding to the three security categories in Table 3.1 is indicated on the  $x$ -axis with a line at 256.



(b) The time complexity growth as a function of  $k$ , for a fixed value of  $n = 256$ , corresponding to the value listed in Table 3.1. The  $k$  values corresponding to the three security categories in Table 3.1 are indicated along the  $x$ -axis by a line at 2, 3, and 4.

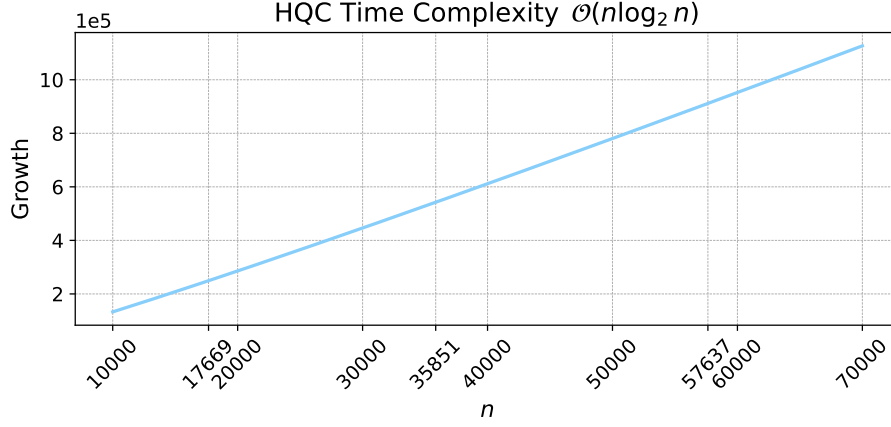
**Figure 6.2:** The time complexity growth of  $\mathcal{O}(kn \log_2(n))$  as a function of  $n$  and  $k$ , respectively.

In Figure 6.2a, the time complexity increases more rapidly for higher values of  $k$ . At smaller values of  $n$ , the curves are close, converging as  $n$  approaches zero. At  $n = 256$ , each of the curves yield a time complexity of  $4.1 \cdot 10^3$ ,  $6.2 \cdot 10^3$ , and  $8.2 \cdot 10^3$ , respectively. These computational costs continue to increase as  $n$  is increased, reaching  $9 \cdot 10^3$ ,  $13.4 \cdot 10^3$ , and  $18 \cdot 10^3$ , respectively, at  $n = 500$ .

In Figure 6.2b, it can be observed that the time complexity grows with increasing  $k$ . The time complexity at the three vertical lines that indicate the  $k$  values for the three security categories corresponds to the time complexities observed for Figure 6.2a. When  $k$  is increased to 8, the time complexity becomes  $16.4 \cdot 10^4$ . This is two times the value observed at  $k = 4$ .

### 6.1. Time Complexity Comparison

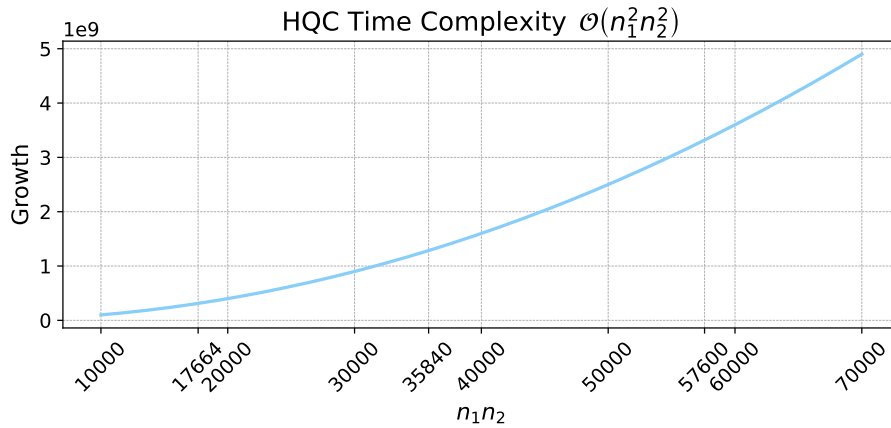
For HQC, both key generation and encapsulation have the same time complexity of  $\mathcal{O}(n \log_2(n))$ . This complexity is shown in Figure 6.3, where the growth is plotted as a function of  $n$ .



**Figure 6.3:** The time complexity growth of  $\mathcal{O}(n \log_2(n))$  as a function of  $n$ . Vertical lines at 17669, 35851, and 57637 mark the values of  $n$  corresponding to the parameter sets in Table 5.3.

In Figure 6.3, it can be observed that the time complexity grows with increasing  $n$ . At the parameter set values for  $n$ , the time complexity is  $2.5 \cdot 10^5$ ,  $5.4 \cdot 10^5$ , and  $9.1 \cdot 10^5$ , respectively. When  $n$  is increased to 70000, the time complexity reaches  $11.3 \cdot 10^5$ .

The decapsulation for HQC have the time complexity of  $\mathcal{O}(n_1^2 n_2^2)$ . Figure 6.4 presents the growth as a function of the product  $n_1 n_2$ .



**Figure 6.4:** The time complexity growth of  $\mathcal{O}(n_1^2 n_2^2)$  as a function of the product  $n_1 n_2$ . Vertical lines at 17664, 35840, and 57600 mark the values of  $n_1 n_2$  corresponding to the parameter sets in Table 5.3.

In Figure 6.4, the curve exhibits a quadratic growth, as both  $n_1$  and  $n_2$  are squared in the time complexity expression. With increasing values of  $n_1 n_2$ , the computational cost increases as

## 6.2. Space Complexity Comparison

well. At the values corresponding to the three security categories for  $n_1$  and  $n_2$ , the observed time complexities are  $0.3 \cdot 10^9$ ,  $1.3 \cdot 10^9$ , and  $3.3 \cdot 10^9$ , respectively. When  $n_1 n_2$  are increased to 70000, the time complexity reaches  $4.9 \cdot 10^9$ .

When comparing ML-KEM and HQC, differences in their computational costs and how their parameters affect the time complexities are observed. For ML-KEM, the time complexities are on the order of  $10^3$  and  $10^4$ , with  $k$  having the greatest influence on computational cost. This is observed when doubling  $n$  from the parameter set values approximately doubles the time complexity for all three  $k$  values. However, doubling  $k$  results in almost a fourfold increase in time complexity for the key generation and encapsulation algorithms. In comparison, HQC exhibits higher costs, reaching time complexities in the order of  $10^5$  and  $10^9$ . This is primarily due to a dominating quadratic term in the time complexity, which leads to much worse scaling as input size increases.

## 6.2 Space Complexity Comparison

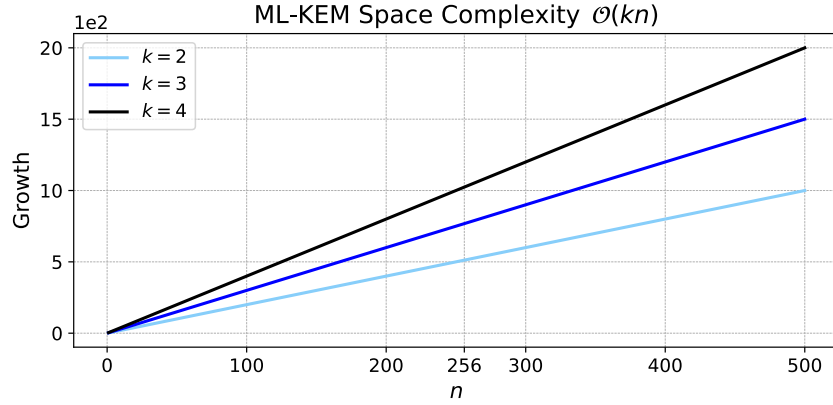
The overall space complexities of the key generation, encapsulation, and decapsulation algorithms for ML-KEM and HQC, along with the total space complexities for each scheme, are summarised in Table 6.2.

Algorithm	ML-KEM Space Complexity	HQC Space Complexity
Key Generation	$\mathcal{O}(kn)$	$\mathcal{O}(n)$
Encapsulation	$\mathcal{O}(kn)$	$\mathcal{O}(k_1 k_2 n_1 n_2)$
Decapsulation	$\mathcal{O}(kn)$	$\mathcal{O}(k_1 k_2 n_1 n_2)$
Total Complexity	$\mathcal{O}(kn)$	$\mathcal{O}(k_1 k_2 n_1 n_2)$

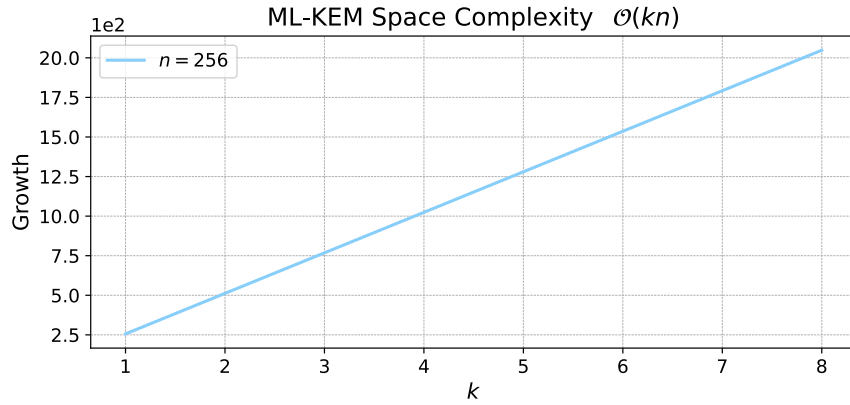
**Table 6.2:** Summary of the space complexities for each algorithm in ML-KEM and HQC.

For ML-KEM, all three algorithms share the same space complexity of  $\mathcal{O}(kn)$ . Two plots are provided to show how this complexity grows with respect to  $n$  and  $k$ . Figure 6.5a shows the growth as a function of  $n$ . Figure 6.5b shows the growth as a function of  $k$ .

## 6.2. Space Complexity Comparison



(a) The space complexity growth as a function of  $n$ , with  $k = 2, 3, 4$ , corresponding to the values in Table 3.1. The  $n$  value corresponding to the three security category in Table 3.1 is indicated on the  $x$ -axis with a line at 256.



(b) The space complexity growth as a function of  $k$ , with  $n = 256$ , corresponding to the value in Table 3.1. The  $k$  values corresponding to the three security categories in Table 3.1 are indicated on the  $x$ -axis by a line at 2, 3, and 4.

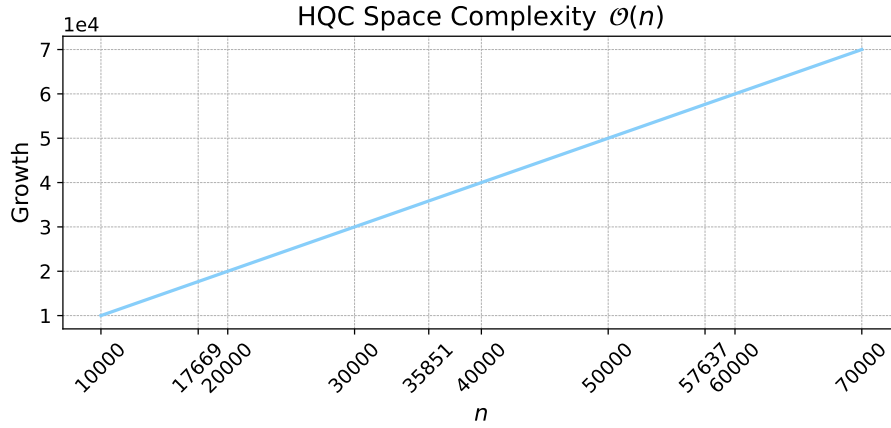
**Figure 6.5:** The space complexity growth  $\mathcal{O}(kn)$  as a function of  $n$  and  $k$ , respectively.

In Figure 6.5a, the graphs exhibit a linear growth, with a higher space complexity for larger values of  $k$ . It can be observed that for small values of  $n$ , the graphs are close, converging to zero as  $n$  approaches zero. At  $n = 256$ , each of the graphs yield a space complexity of  $5.1 \cdot 10^2$ ,  $7.7 \cdot 10^2$ , and  $10.2 \cdot 10^2$ , respectively. These computational costs continue to increase as  $n$  increases, reaching  $10 \cdot 10^2$ ,  $15 \cdot 10^2$ , and  $20 \cdot 10^2$ , respectively, at  $n = 500$ .

In Figure 6.5b, the graph exhibits a linear growth. The space complexities at the three vertical lines that indicate the  $k$  values for the three security categories correspond to the space complexities observed for Figure 6.5a. When  $k$  is increased to 8, the space complexity becomes  $20.5 \cdot 10^2$ . This is two times the value observed at  $k = 4$ .

For HQC, the key generation has the space complexity of  $\mathcal{O}(n)$ . Figure 6.6 presents the growth as a function of  $n$ .

## 6.2. Space Complexity Comparison

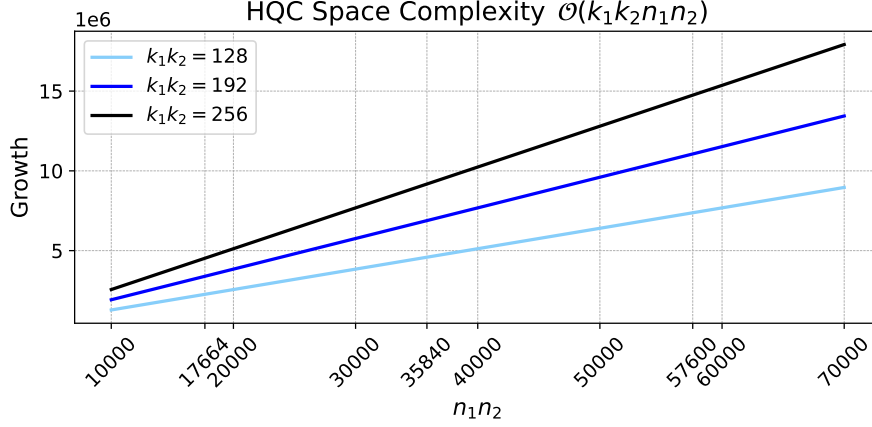


**Figure 6.6:** The space complexity growth of  $\mathcal{O}(n)$  as a function of  $n$ . The parameter values for  $n$  from Table 5.3 are marked on the  $x$ -axis by a line at 17669, 35851, and 57637.

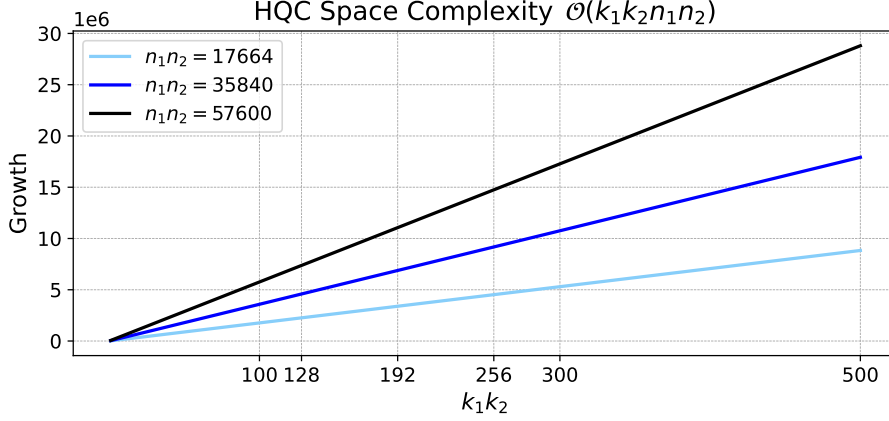
In Figure 6.6, the graph exhibits a linear growth, as the space complexity depends solely on  $n$ . Consequently, the increase in space complexity directly corresponds to the increase in  $n$ .

The encapsulation and decapsulation for HQC have the space complexity  $\mathcal{O}(k_1 k_2 n_1 n_2)$ . Two plots are provided to show how this complexity grows with respect to  $k_1 k_2$  and  $n_1 n_2$ . Figure 6.7a shows the growth as a function of  $n_1 n_2$ . Figure 6.7b shows the growth as a function of  $k_1 k_2$ .

## 6.2. Space Complexity Comparison



(a) The space complexity growth as a function of  $n_1n_2$ , with  $k_1k_2 = 128, 192, 256$ , corresponding to the values in Table 5.3. The  $n_1n_2$  values for each security category is indicated on the  $x$ -axis with a line at 17664, 35840, and 57600.



(b) The space complexity growth as a function of  $k_1k_2$ , with  $n_1n_2 = 17664, 35840, 57600$ , corresponding to the values in Table 5.3. The  $k_1k_2$  values for the three security categories are shown on the  $x$ -axis by a line at 128, 192, and 256.

**Figure 6.7:** The space complexity growth  $\mathcal{O}(k_1k_2n_1n_2)$  as a function of  $n_1n_2$  and  $k_1k_2$ , respectively.

In Figure 6.7a, the graphs exhibit linear growth. At the parameter set value for  $n_1n_2 = 17664$ , the space complexity reaches  $2.3 \cdot 10^6$ ,  $3.4 \cdot 10^6$ , and  $4.5 \cdot 10^6$ . At  $n_1n_2 = 35840$ , the space complexity reaches  $4.6 \cdot 10^6$ ,  $6.9 \cdot 10^6$ , and  $9.2 \cdot 10^6$ . At  $n_1n_2 = 57600$ , the space complexity reaches  $7.4 \cdot 10^6$ ,  $11 \cdot 10^6$ , and  $14.7 \cdot 10^6$ . These computational costs continue to increase linearly as  $n_1n_2$  increases, reaching  $9 \cdot 10^6$ ,  $13.4 \cdot 10^6$ , and  $18 \cdot 10^6$  at  $n_1n_2 = 70000$ .

In Figure 6.7b, the graphs also exhibit linear growth. The space complexities at the three vertical lines that indicate the  $k_1k_2$  values for the three security categories corresponds to the space complexities observed for Figure 6.7a. When  $k_1k_2$  is increased to 500, the space complexity becomes  $8.8 \cdot 10^6$ ,  $17.9 \cdot 10^6$ , and  $28.8 \cdot 10^6$ , respectively. This is almost two times the values observed at  $k_1k_2 = 256$ .

When comparing ML-KEM and HQC, differences in their memory requirements and how their

### 6.3. Bit Complexity Comparison

parameters affect the space complexities are observed. For ML-KEM, the space complexities are on the order of  $10^2$ , with both  $k$  and  $n$  contributing equally and linearly to the space complexity. In the case of HQC, the costs are higher, reaching space complexities in the order of  $10^4$  and  $10^6$ .

## 6.3 Bit Complexity Comparison

This section presents a comparative bit complexity analysis of ML-KEM and HQC. The first subsection focuses on space bit complexity, which measures the size of keys and ciphertexts. The second subsection examines time bit complexity, which estimates the number of bit operations required by each scheme. The bit counts do not take into account hardware and implementation specifics. However, since the bit complexities for both ML-KEM and HQC are derived in a consistent manner, they remain valid for a comparison between the schemes.

### Space Bit Complexity Comparison

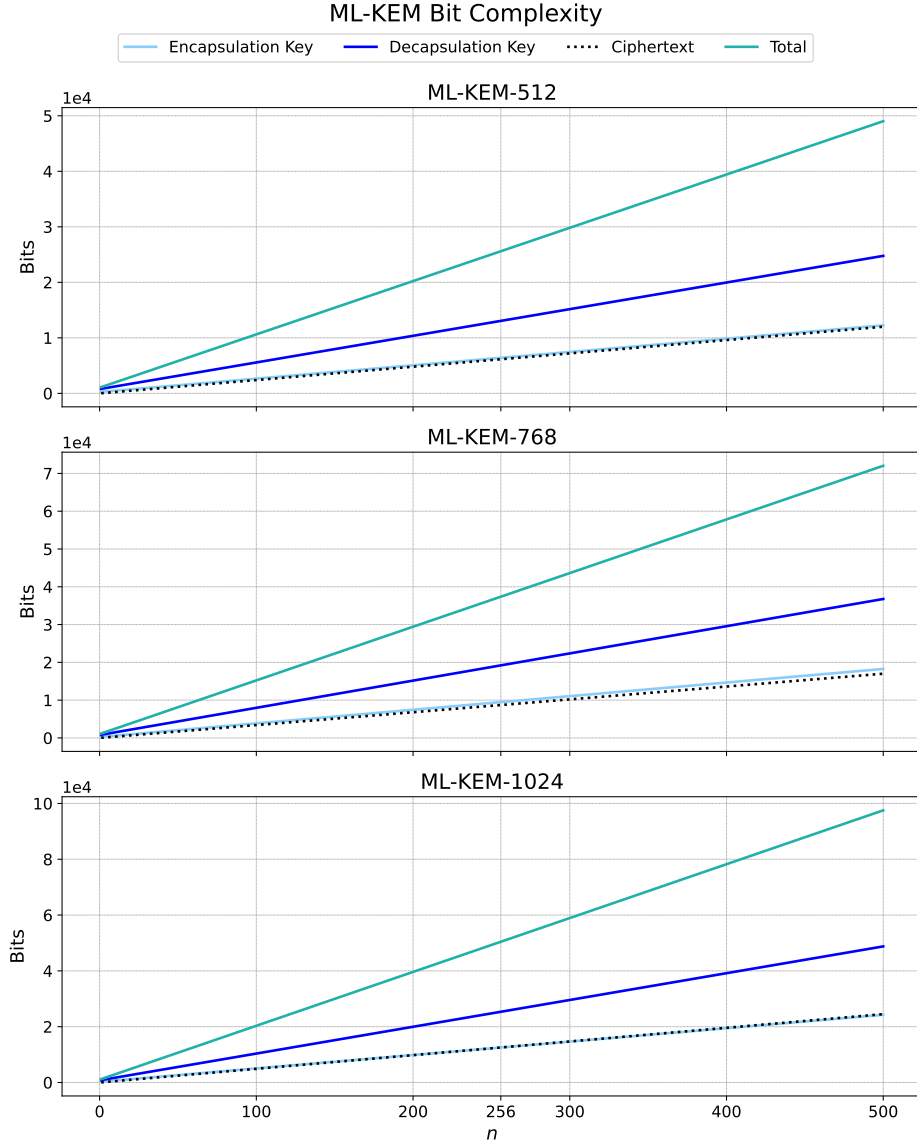
The space bit complexities for ML-KEM and HQC are summarised in Table 6.3, along with the total space required by each scheme. Throughout this subsection, all references to bit complexity, including in figures and plots, should be understood as referring to space bit complexity.

Component	ML-KEM Bit Complexity	HQC Bit Complexity
Public encapsulation Key	$12kn + 256$ bits	$320 + n$ bits
Private decapsulation Key	$24kn + 768$ bits	$320 + k_1k_2$ bits
Ciphertext	$knd_u + nd_v$ bits	$n + n_1n_2 + 128$ bits
Total Complexity	$n(36k + kd_u + d_v) + 1024$ bits	$2n + k_1k_2 + n_1n_2 + 768$ bits

**Table 6.3:** Summary of the bit complexities for each algorithm in ML-KEM and HQC.

For ML-KEM, the bit complexities depend on four parameters,  $n$ ,  $k$ ,  $d_u$ , and  $d_v$ . The parameters  $d_u$  and  $d_v$  affect only the ciphertext and total bit complexity. Each bit complexity is plotted as a function of one parameter, with the others fixed according to Table 3.1. The results appear in four figures, one per variable. Each figure contains three subplots for the three parameter sets. In Figure 6.8, the bit complexities in ML-KEM are shown as functions of  $n$ .

### 6.3. Bit Complexity Comparison

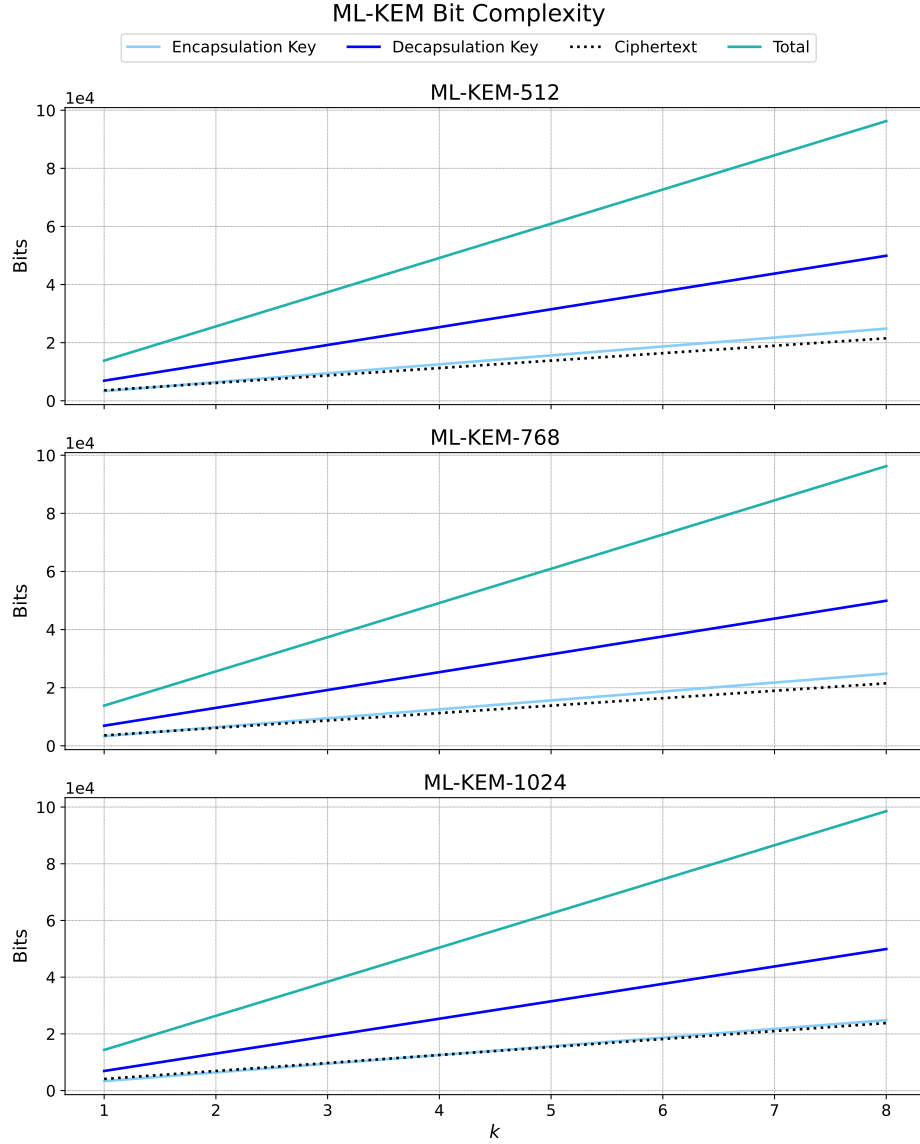


**Figure 6.8:** The bit complexity as a function of  $n$  for each component, with other parameters fixed according to Table 3.1. The  $n$  value corresponding to the three security categories is indicated on the  $x$ -axis with a line at 256.

The bit complexities increase linearly with  $n$  across all components. The private decapsulation key consistently shows the highest bit complexity and steepest growth due to its larger scaling factor. The public encapsulation key and ciphertext complexities are nearly identical, with their values overlapping throughout the range of  $n$ . The total bit complexity combines these components, resulting in the largest overall growth. Notably, the total bit complexity approximately doubles when moving from the lowest to the highest security category.

In Figure 6.9, the bit complexities in ML-KEM are shown as functions of  $k$ .

### 6.3. Bit Complexity Comparison

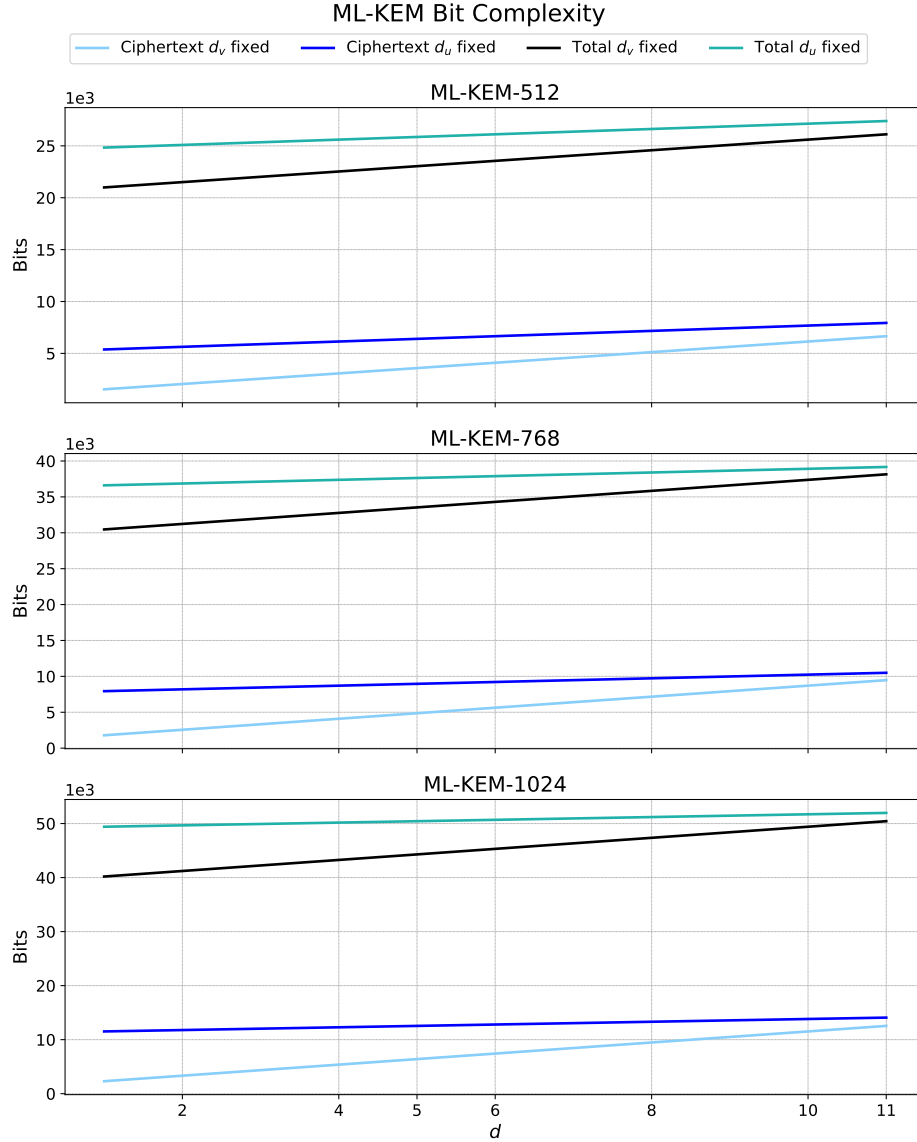


**Figure 6.9:** The bit complexity as a function of  $k$  for each component, with other parameters fixed according to Table 3.1. The  $k$  values corresponding to the three security categories are indicated along the  $x$ -axis by a line at 2, 3, and 4.

The bit complexities increase linearly with  $k$  for all components. Similar to the behaviour with  $n$ , the private decapsulation key exhibits the highest bit complexity and steepest growth due to its larger scaling factor. The public encapsulation key and ciphertext complexities remain closely aligned, with overlapping values across the range of  $k$ . The total bit complexity shows the greatest linear increase. Unlike the case with  $n$ , the bit complexity shows minimal variation across parameter sets as  $k$  changes, indicating that  $k$  has a limited influence on the overall bit complexity.

In Figure 6.10, the bit complexities in ML-KEM are shown as functions of  $d_u$  and  $d_v$ .

### 6.3. Bit Complexity Comparison



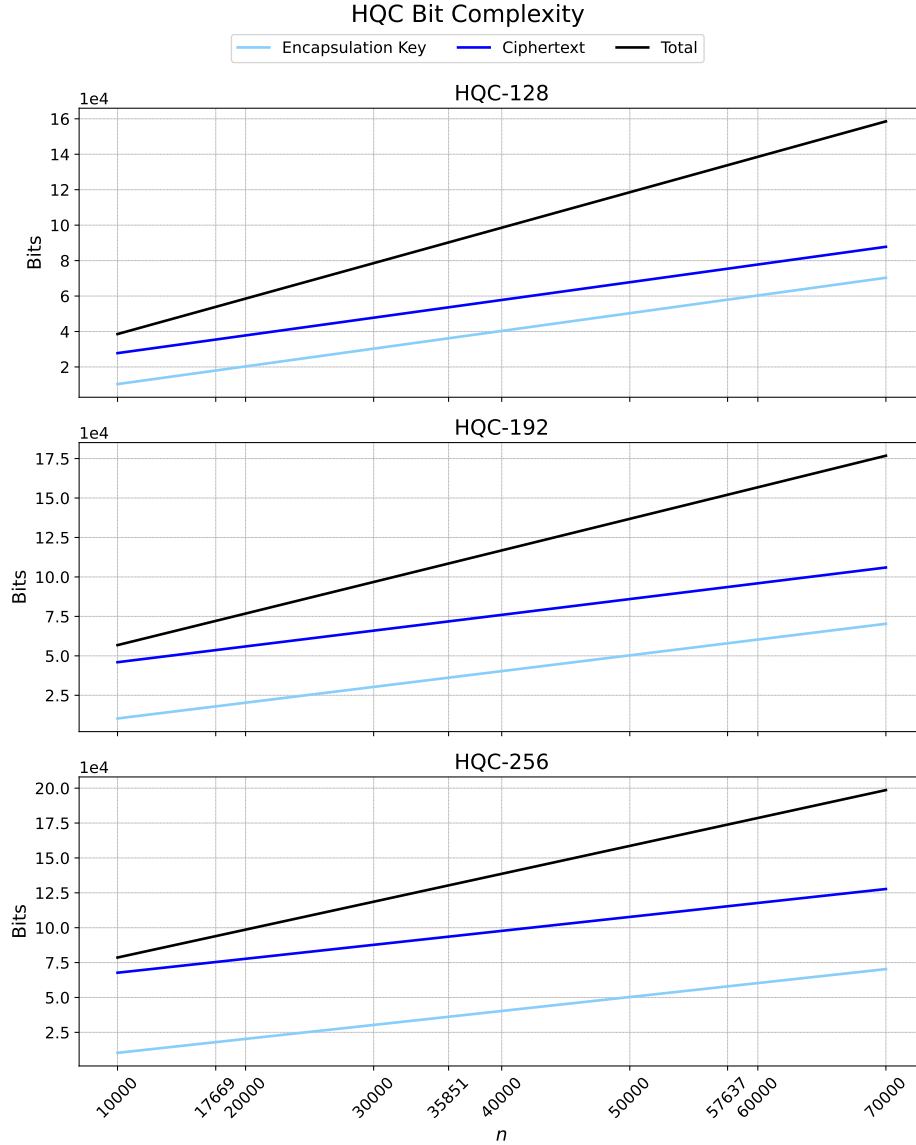
**Figure 6.10:** The bit complexity as a function of  $d_u$  and  $d_v$  for each component, with other parameters fixed according to Table 3.1. The  $d_u$  and  $d_v$  values corresponding to the three security categories are indicated along the  $x$ -axis by a line at 10 and 11 for  $d_u$ , and at 4 and 5 for  $d_v$ .

The bit complexities vary linearly with both  $d_u$  and  $d_v$ , but their influence is limited to the ciphertext and total bit complexity components. Bit complexity increases more noticeably with  $d_u$  than with  $d_v$ , reflecting the larger coefficients associated with  $d_u$  in the ciphertext expression. The ciphertext bit complexity exhibits some growth between the parameter sets. However, the most significant increase is observed in the total bit complexity. This larger overall growth is likely influenced more by fixed parameters within the security categories than by variations in  $d_u$  and  $d_v$  alone.

For HQC, the bit complexities depend on the parameters  $n$ ,  $k_1k_2$ , and  $n_1n_2$ . The public encapsulation key complexity scales linearly with  $n$ , while the private decapsulation key

### 6.3. Bit Complexity Comparison

complexity depends on  $k_1k_2$ . The ciphertext complexity increases with both  $n$  and  $n_1n_2$ . The total bit complexity aggregates these terms, reflecting linear growth with respect to all three parameters. Each complexity measure is plotted as a function of one parameter, holding the others fixed according to Table 5.3. The plots are organised into three figures, one per variable, each containing subplots for the three parameter sets. In Figure 6.11, the bit complexities are shown as functions of  $n$ .



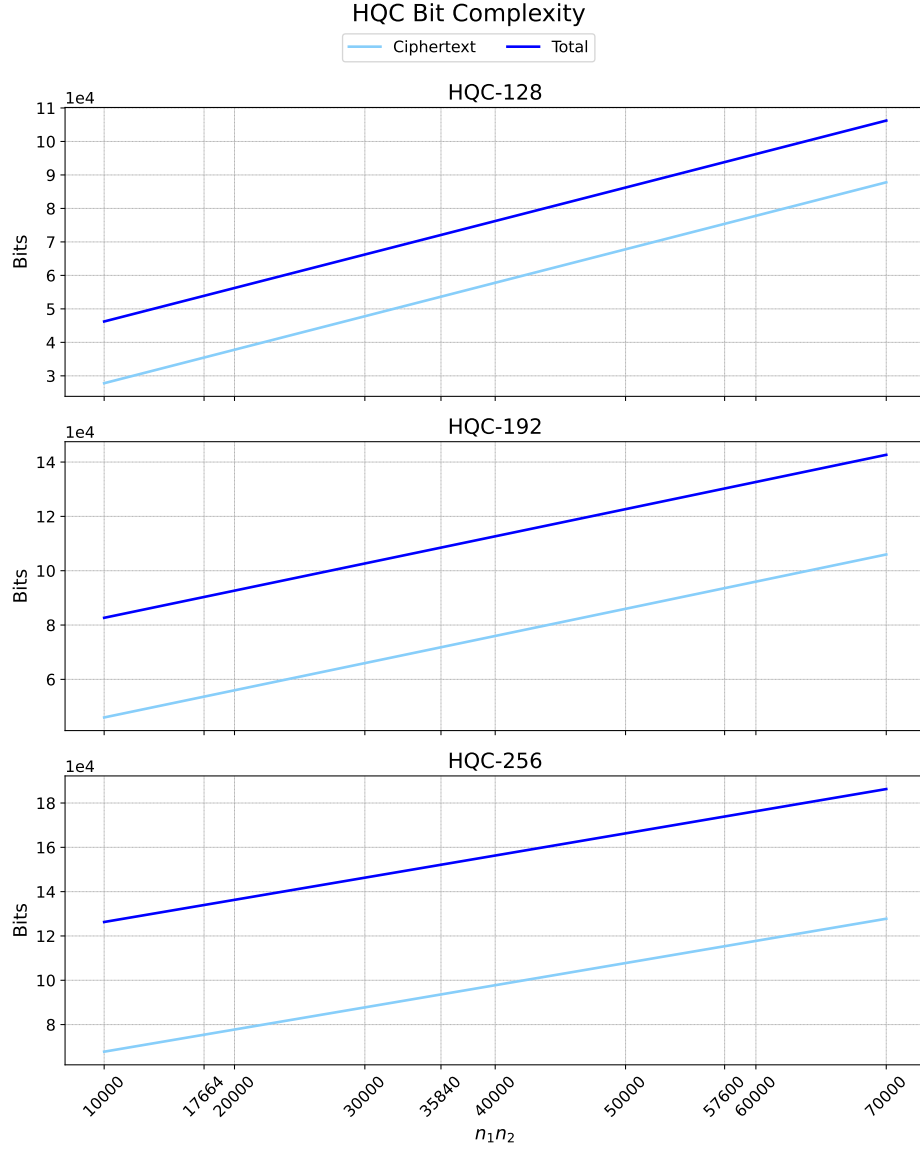
**Figure 6.11:** The bit complexity as a function of  $n$  for each component, with other parameters fixed according to Table 5.3. The  $n$  values corresponding to the three security categories are indicated along the  $x$ -axis by a line at 17669, 35851 and 57637.

The bit complexities in HQC increase linearly with  $n$ , as expected from their parameter dependencies. The public encapsulation key and ciphertext complexities both grow with  $n$ , resulting in parallel trends across the parameter sets. The total bit complexity shows

### 6.3. Bit Complexity Comparison

the steepest increase, reflecting the cumulative effect of the public encapsulation key and ciphertext components. Differences between parameter sets are visible but moderate.

In Figure 6.12, the bit complexities are shown as functions of  $n_1n_2$ .

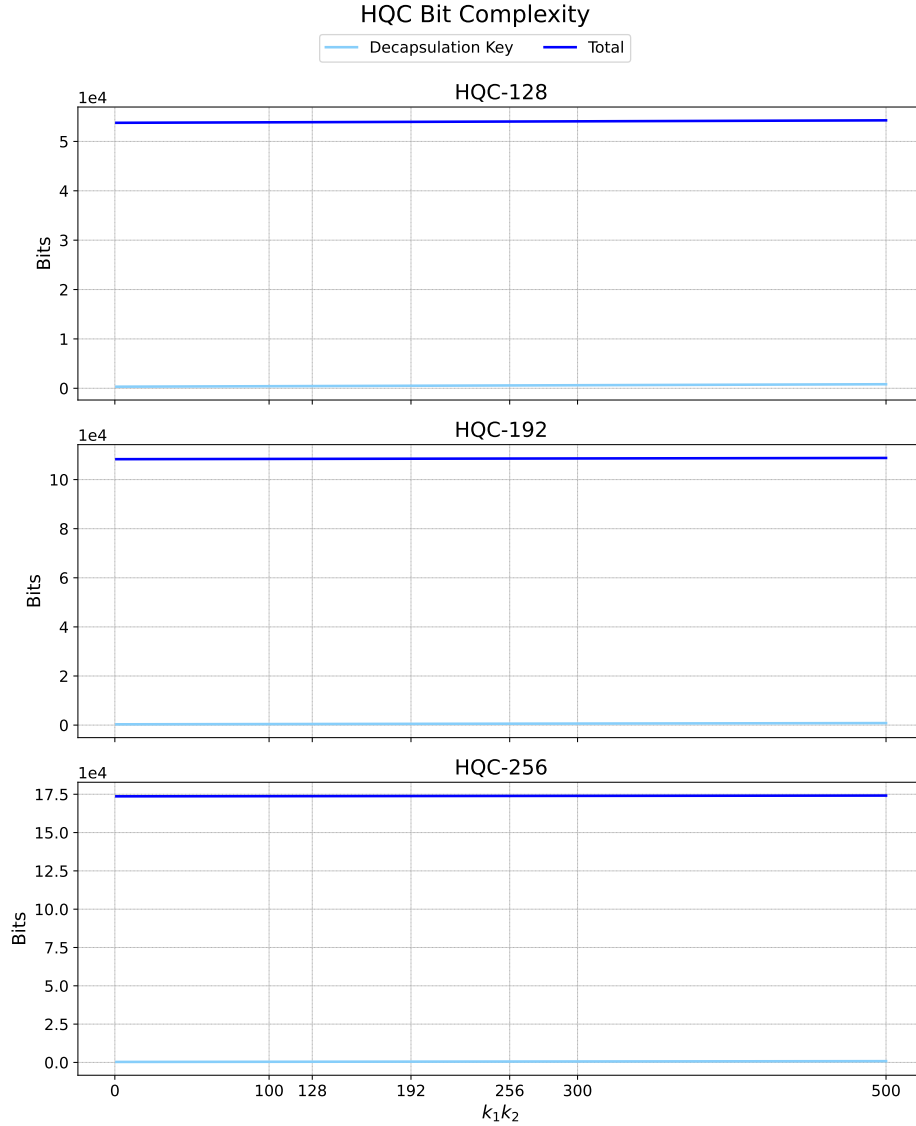


**Figure 6.12:** The bit complexity as a function of  $n_1n_2$  for each component, with other parameters fixed according to Table 5.3. The  $n_1n_2$  values corresponding to the three security categories are indicated along the  $x$ -axis by a line at 17664, 35840 and 57600.

The bit complexities grow linearly with  $n_1n_2$ , which directly influences the ciphertext and total bit complexity. The ciphertext complexity increases proportionally with  $n_1n_2$ , driving the corresponding increase in total bit complexity. The slope of the total bit complexity is therefore dominated by the ciphertext component. Variation across parameter sets is evident.

### 6.3. Bit Complexity Comparison

In Figure 6.13, the bit complexities are shown as functions of  $k_1k_2$ .



**Figure 6.13:** The bit complexity as a function of  $k_1k_2$  for each component, with other parameters fixed according to Table 5.3. The  $k_1k_2$  values corresponding to the three security categories are indicated along the  $x$ -axis by a line at 128, 192 and 256.

The bit complexities increase linearly with  $k_1k_2$ , affecting only the private decapsulation key and total bit complexity. While the private decapsulation key complexity increases with  $k_1k_2$ , the total bit complexity lies significantly above it across all parameter sets. This large gap reflects the contribution of other fixed terms, particularly the public encapsulation key and ciphertext, which dominate the total bit complexity. The result is a steep offset between the total bit and private decapsulation key complexities that persists regardless of the value of  $k_1k_2$ .

Comparing ML-KEM and HQC, distinct patterns emerge in how bit complexity scales with

### 6.3. Bit Complexity Comparison

each scheme’s parameters. ML-KEM exhibits more tightly coupled growth across its components due to shared dependencies on  $n$  and  $k$ , leading to consistent relative differences between public encapsulation key, private decapsulation key, and ciphertext complexities. In contrast, HQC’s total bit complexity is more heavily influenced by the ciphertext component, particularly through  $n_1n_2$ , with less variation in key sizes across parameters. Notably, HQC’s private decapsulation key shows a weaker contribution to the overall bit complexity compared to ML-KEM, while ML-KEM’s public encapsulation and ciphertext complexities are closely aligned. Overall, ML-KEM displays a more uniform and proportionate increase in bit complexity across its components, whereas HQC’s total bit complexity is dominated by fewer, larger contributors.

With respect to transmission, the ciphertext bit complexity serves as the primary factor. In HQC, the ciphertext includes large contributions from both  $n$  and  $n_1n_2$ , resulting in a greater transmission size compared to ML-KEM. ML-KEM’s ciphertexts are more compact and grow more gradually as the parameters increase. As a result, the transmission cost, measured by the size of the ciphertext, is consistently lower in ML-KEM across all parameter sets.

### Time Bit Complexity Comparison

The time bit complexities for ML-KEM and HQC are presented in Table 6.4. Throughout this subsection, all mentions of bit complexity refer specifically to time bit complexity. While these values are not visualised in plots, they are computed to provide a concrete understanding of the computational demands of each scheme.

Algorithm	Security Category	ML-KEM	HQC
Key Generation	1	148 736 bits	267 487 bits
	3	315 136 bits	579 026 bits
	5	542 976 bits	970 126 bits
Encapsulation	1	241 920 bits	2 760 782 bits
	3	437 504 bits	7 967 730 bits
	5	694 528 bits	16 570 625 bits
Decapsulation	1	238 336 bits	315 311 653 bits
	3	332 544 bits	1 293 629 935 bits
	5	432 896 bits	3 336 268 653 bits

**Table 6.4:** Time bit complexity comparison of ML-KEM and HQC for key generation, encapsulation, and decapsulation across NIST security categories.

### 6.3. Bit Complexity Comparison

For key generation, both schemes show a steady, linear increase in time bit complexity as the security level rises, with HQC consistently requiring approximately 1.8 times more bits than ML-KEM. Encapsulation further amplifies the difference between the two schemes. Even at the lowest security level, HQC requires over 11 times the number of bits that ML-KEM does. This difference becomes even more pronounced at higher security levels, where HQC requires nearly 24 times more bits than ML-KEM for security category 5. The most significant difference appears in decapsulation. While ML-KEM scales moderately with the security category, HQC's time bit complexity grows sharply.

## 7 Discussion

In this master's thesis, the purpose was to investigate how time, space, and bit complexities of ML-KEM and HQC scale with their input parameters. Furthermore, it assesses what those scaling behaviours imply for the deployment of each scheme on devices with limited computational resources, memory, and bandwidth. Therefore, a complexity analysis was conducted, followed by a comparative analysis of the results.

### Time Complexity Analysis

The time complexity analysis revealed a significant difference between the two schemes. For ML-KEM, the key generation and encapsulation scale with  $\mathcal{O}(k^2 n \log_2(n))$  on the order of  $10^4$ , and the decapsulation scale with  $\mathcal{O}(kn \log_2(n))$  on the order of  $10^3$ . In contrast, for HQC, the key generation and encapsulation scale with  $\mathcal{O}(n \log_2(n))$ , while the decapsulation scale with  $\mathcal{O}(n_1^2 n_2^2)$ . The quadratic term leads to a steep increase in time complexity, resulting in a time complexity to the order of  $10^9$ . HQC's lowest parameter set then demands more computational cost than ML-KEM's highest.

When considering devices with limited processing capabilities, HQC's computationally heavy decoding step, due to the quadratic term, may pose challenges for deployment. In contrast, ML-KEM's lower increase in time complexity makes it more suitable for such constrained devices. Therefore, the time complexity analysis indicates that ML-KEM is the computationally preferable choice, whereas HQC may better suit devices that can accommodate higher computational demands.

### Space Complexity Analysis

The space complexity analysis revealed a similar difference between the two schemes. For ML-KEM, all three algorithms as well as the total space complexity have the same space complexity of  $\mathcal{O}(kn)$ , which scales linearly with both  $k$  and  $n$ . Across all three parameter sets, the space complexity is on the order of  $10^2$ . In contrast, for HQC, the key generation scale with  $\mathcal{O}(n)$  on the order of  $10^4$ , and the encapsulation, decapsulation, and total space complexity scale with  $\mathcal{O}(k_1 k_2 n_1 n_2)$  on the order of  $10^6$ . Therefore, the memory consumption is two to four orders of magnitude greater than that of ML-KEM. This stems from HQC's inherently larger parameter sizes, which result in higher memory consumption even at its lowest security category.

When considering devices with limited RAM and memory capacity, ML-KEM's lower space complexity offers an advantage compared to HQC.

### Bit Complexity Analysis

The key point in space bit complexity lies in the size of the ciphertext, as this has an impact on transmission efficiency and communication costs. For ML-KEM, the ciphertext

size increases moderately with the input parameters, remaining relatively compact even at the highest security category. In contrast, HQC's ciphertext is mainly dominated by large terms for all security categories. As a result, even at its lowest security category, HQC produces ciphertexts that exceed those of ML-KEM at its highest. This leads to a higher space bit complexity, which directly translates to increased communication costs.

In the context of bandwidth-limited devices, the impact of the ciphertext size is an important factor. ML-KEM's smaller and more efficient ciphertexts offer a significant advantage, enabling faster transmission and lower communication costs. In contrast, HQC's larger ciphertexts may pose a challenge in scenarios where communication efficiency is critical.

The difference between ML-KEM and HQC is further reflected in the time bit complexity. While both schemes show a steady increase in key generation, HQC consistently requires more bits. Specifically, in decapsulation HQC's time bit complexity increases steeply and far exceeds that of ML-KEM.

## Summary

The comparative complexity analysis of ML-KEM and HQC across time, space, and bit provides a theoretical foundation for evaluating their suitability in resource-constrained devices. The findings indicate that ML-KEM is more suitable for devices with limited computational power, memory, and bandwidth. Furthermore, ML-KEM's smaller ciphertexts and keys offer an advantage in resource-constrained devices by reducing bit complexity and communication costs. However, it is important to note that this analysis is based on asymptotic and theoretical complexity estimates and has not been validated through practical implementation. Therefore, one could, for instance, implement both schemes to experimentally evaluate their time, space, and bit complexities in order to confirm and demonstrate the theoretical findings.

## 8 Conclusion

The advancement of quantum computing poses significant challenges to classical cryptographic schemes, necessitating the development of quantum-resistant KEMs. This master’s thesis presented a comparative analysis of two KEMs, ML-KEM and HQC, which are standardised or in the process of standardisation by NIST. This analysis aimed to answer the following problem statement:

*How do the time, space, and bit complexities of lattice- and code-based post-quantum key encapsulation mechanisms scale with input parameters, and what are the implications for their use in resource-constrained devices?*

First, the foundational lattice theory and security assumptions underpinning ML-KEM were examined, followed by a detailed description of the ML-KEM scheme. Subsequently, the relevant coding theory and security assumptions for HQC were reviewed, followed by a comprehensive presentation of the HQC scheme. The asymptotic time and space complexities, along with the bit complexities of both schemes, were derived. Lastly, these complexity measures facilitated a comparative analysis, enabling an evaluation of the trade-offs between security categories and parameter choices inherent to each scheme.

The comparative analysis reveals that ML-KEM exhibits a slower growth rate across time, space, and bit complexities compared to HQC. In particular, HQC’s time complexity is dominated by the decapsulation process, which introduces quadratic terms that significantly increase computation cost at higher security categories. In terms of bit complexity, HQC produces significantly larger ciphertexts and requires substantially more bits across all operations compared to ML-KEM, thus posing challenges for devices operating over limited-bandwidth communication channels.

To conclude, ML-KEM demonstrates more predictable and moderate scaling across parameter sets, making it more suitable for implementation in resource-constrained devices. Its balanced performance in terms of computational efficiency and data size offers a practical advantage over HQC in such settings.

# Bibliography

- [1] Lily Chen et al. *Report on Post-Quantum Cryptography*. en. Tech. rep. NIST Internal or Interagency Report (NISTIR) 8105. National Institute of Standards and Technology, Apr. 2016. DOI: 10.6028/NIST.IR.8105. URL: <https://csrc.nist.gov/pubs/ir/8105/final> (visited on 02/19/2025).
- [2] Sieglinde M. -L. Pfaendler, Konstantin Konson, and Franziska Greinert. “Advancements in Quantum Computing—Viewpoint: Building Adoption and Competency in Industry”. en. In: *Datenbank-Spektrum* 24.1 (Mar. 2024), pp. 5–20. ISSN: 1610-1995. DOI: 10.1007/s13222-024-00467-4. URL: <https://doi.org/10.1007/s13222-024-00467-4> (visited on 02/14/2025).
- [3] Michele Mosca. “Cybersecurity in an Era with Quantum Computers: Will We Be Ready?” In: *IEEE Security & Privacy* 16.5 (Sept. 2018). Conference Name: IEEE Security & Privacy, pp. 38–41. ISSN: 1558-4046. DOI: 10.1109/MSP.2018.3761723. URL: <https://ieeexplore.ieee.org/document/8490169> (visited on 02/18/2025).
- [4] Gorjan Alagic et al. *Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process*. en. Tech. rep. NIST Internal or Interagency Report (NISTIR) 8240. National Institute of Standards and Technology, Jan. 2019. DOI: 10.6028/NIST.IR.8240. URL: <https://csrc.nist.gov/pubs/ir/8240/final> (visited on 02/19/2025).
- [5] National Institute of Standards and Technology. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. en. Tech. rep. Federal Information Processing Standard (FIPS) 203. U.S. Department of Commerce, Aug. 2024. DOI: 10.6028/NIST.FIPS.203. URL: <https://csrc.nist.gov/pubs/fips/203/final> (visited on 02/13/2025).
- [6] National Institute of Standards and Technology. *Module-Lattice-Based Digital Signature Standard*. en. Tech. rep. Federal Information Processing Standard (FIPS) 204. U.S. Department of Commerce, Aug. 2024. DOI: 10.6028/NIST.FIPS.204. URL: <https://csrc.nist.gov/pubs/fips/204/final> (visited on 02/19/2025).
- [7] National Institute of Standards and Technology. *Stateless Hash-Based Digital Signature Standard*. en. Tech. rep. Federal Information Processing Standard (FIPS) 205. U.S. Department of Commerce, Aug. 2024. DOI: 10.6028/NIST.FIPS.205. URL: <https://csrc.nist.gov/pubs/fips/205/final> (visited on 02/19/2025).
- [8] Gorjan Alagic. *Status Report on the Fourth Round of the NIST Post-Quantum Cryptography Standardization Process*. en. Tech. rep. NIST IR 8545. Gaithersburg, MD: National Institute of Standards and Technology, 2025, NIST IR 8545. DOI: 10.6028/NIST.IR.8545. URL: <https://nvlpubs.nist.gov/nistpubs/ir/2025/NIST.IR.8545.pdf> (visited on 04/08/2025).
- [9] Khwaja Mansoor et al. “Securing the future: exploring post-quantum cryptography for authentication and user privacy in IoT devices”. en. In: *Cluster Computing* 28.2 (Apr. 2025). Company: Springer Distributor: Springer Institution: Springer Label:

## Bibliography

- Springer Number: 2 Publisher: Springer US, pp. 1–44. ISSN: 1573-7543. DOI: 10.1007/s10586-024-04799-4. URL: <https://link.springer.com/article/10.1007/s10586-024-04799-4> (visited on 02/19/2025).
- [10] Douglas Robert Stinson and Maura Paterson. *Cryptography: Theory and Practice*. English. 4th edition. Boca Raton: Chapman and Hall/CRC, Sept. 2018. ISBN: 978-1-138-19701-5.
- [11] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. English. 1st edition. Boca Raton, Fla.: CRC Press, Dec. 1996. ISBN: 978-0-8493-8523-0.
- [12] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *An Introduction to Mathematical Cryptography*. en. Undergraduate Texts in Mathematics. New York, NY: Springer, 2014. ISBN: 978-1-4939-1710-5. DOI: 10.1007/978-1-4939-1711-2. URL: <https://link.springer.com/10.1007/978-1-4939-1711-2> (visited on 02/26/2025).
- [13] Steven D. Galbraith. *Mathematics of Public Key Cryptography*. English. Cambridge ; New York: Cambridge University Press, Mar. 2012. ISBN: 978-1-107-01392-6.
- [14] Colin P. Williams. *Explorations in Quantum Computing*. Texts in Computer Science. London: Springer, 2011. ISBN: 978-1-84628-886-9. DOI: 10.1007/978-1-84628-887-6. URL: <http://link.springer.com/10.1007/978-1-84628-887-6> (visited on 02/20/2025).
- [15] Jingwen Suo et al. “Quantum algorithms for typical hard problems: a perspective of cryptanalysis”. en. In: *Quantum Information Processing* 19.6 (Apr. 2020), p. 178. ISSN: 1573-1332. DOI: 10.1007/s11128-020-02673-x. URL: <https://doi.org/10.1007/s11128-020-02673-x> (visited on 02/21/2025).
- [16] P.W. Shor. “Algorithms for quantum computation: discrete logarithms and factoring”. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. Nov. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700. URL: <https://ieeexplore.ieee.org/document/365700> (visited on 02/19/2025).
- [17] Lov K. Grover. “A fast quantum mechanical algorithm for database search”. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing*. STOC '96. New York, NY, USA: Association for Computing Machinery, July 1996, pp. 212–219. ISBN: 978-0-89791-785-8. DOI: 10.1145/237814.237866. URL: <https://dl.acm.org/doi/10.1145/237814.237866> (visited on 02/24/2025).
- [18] Dustin Moody et al. *Transition to Post-Quantum Cryptography Standards*. en. Tech. rep. NIST Internal or Interagency Report (NISTIR) 8547 (Draft). National Institute of Standards and Technology, Nov. 2024. DOI: 10.6028/NIST.IR.8547.ipd. URL: <https://csrc.nist.gov/pubs/ir/8547/ipd> (visited on 02/24/2025).
- [19] Gorjan Alagic et al. *Status report on the third round of the NIST Post-Quantum Cryptography Standardization process*. en. Tech. rep. NIST IR 8413. Gaithersburg, MD: National Institute of Standards and Technology (U.S.), July 2022, NIST IR 8413. DOI: 10.6028/NIST.IR.8413. URL: <https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413.pdf> (visited on 02/18/2025).

- [20] Ritik Bavdekar et al. “Post Quantum Cryptography: A Review of Techniques, Challenges and Standardizations”. In: *2023 International Conference on Information Networking (ICOIN)*. ISSN: 1976-7684. Jan. 2023, pp. 146–151. DOI: 10.1109/ICOIN56518.2023.10048976. URL: <https://ieeexplore.ieee.org/document/10048976/?arnumber=10048976> (visited on 03/03/2025).
- [21] Apostolos P. Fournaris et al. “Running Longer To Slim Down: Post-Quantum Cryptography on Memory-Constrained Devices”. en. In: *2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*. Berlin, Germany: IEEE, July 2023, pp. 1–6. ISBN: 979-8-3503-4647-3. DOI: 10.1109/COINS57856.2023.10189268. URL: <https://ieeexplore.ieee.org/document/10189268/> (visited on 03/05/2025).
- [22] Kerry A McKay et al. *Report on lightweight cryptography*. en. Tech. rep. NIST IR 8114. Gaithersburg, MD: National Institute of Standards and Technology, Mar. 2017, NIST IR 8114. DOI: 10.6028/NIST.IR.8114. URL: <https://nvlpubs.nist.gov/nistpubs/ir/2017/NIST.IR.8114.pdf> (visited on 03/05/2025).
- [23] Derek Atkins. “Requirements for Post-Quantum Cryptography on Embedded Devices in the IoT”. en. In: (). URL: <https://csrc.nist.gov/presentations/2021/requirements-for-post-quantum-cryptography-on-embe>.
- [24] David Dummit. *Abstract Algebra*. English. Englewood Cliffs, N.J: Prentice Hall College Div, Jan. 1990. ISBN: 978-0-13-004771-7.
- [25] Christof Paar, Jan Pelzl, and Tim Güneysu. *Understanding Cryptography: From Established Symmetric and Asymmetric Ciphers to Post-Quantum Algorithms*. en. Berlin, Heidelberg: Springer, 2024. ISBN: 978-3-662-69006-2. DOI: 10.1007/978-3-662-69007-9. URL: <https://link.springer.com/10.1007/978-3-662-69007-9> (visited on 02/27/2025).
- [26] 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. New York, NY, USA: Association for Computing Machinery, 2005, pp. 84–93. ISBN: 978-1-58113-960-0. DOI: 10.1145/1060590.1060603. URL: <https://doi.org/10.1145/1060590.1060603> (visited on 05/09/2025).
- [27] Qiang Tang and Vanessa Teague, eds. *Public-Key Cryptography – PKC 2024: 27th IACR International Conference on Practice and Theory of Public-Key Cryptography, Sydney, NSW, Australia, April 15–17, 2024, Proceedings, Part II*. en. Vol. 14602. Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2024. ISBN: 978-3-031-57721-5. DOI: 10.1007/978-3-031-57722-2. URL: <https://link.springer.com/10.1007/978-3-031-57722-2> (visited on 03/31/2025).
- [28] Roberto Avanzi et al. “Algorithm Specifications And Supporting Documentation”. en. In: 3.02 (Aug. 2021). URL: <https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf> (visited on 03/19/2025).

## Bibliography

- [29] National Institute of Standards and Technology. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. en. Tech. rep. Federal Information Processing Standard (FIPS) 202. U.S. Department of Commerce, Aug. 2015. DOI: 10.6028/NIST.FIPS.202. URL: <https://csrc.nist.gov/pubs/fips/202/final> (visited on 05/12/2025).
- [30] Ardianto Satriawan, Rella Mareta, and Hanho Lee. *A Complete Beginner Guide to the Number Theoretic Transform (NTT)*. Publication info: Published elsewhere. Minor revision. IEEE Access. 2024. URL: <https://eprint.iacr.org/2024/585> (visited on 03/25/2025).
- [31] Melchor, Carlos Aguilar et al. “Hamming Quasi-Cyclic (HQC) Fourth round version Updated version 19/02/2025”. en. In: *Hamming Quasi-Cyclic (HQC)* (Feb. 2025), p. 52. URL: [https://pqc-hqc.org/doc/hqc-specification\\_2025-02-19.pdf](https://pqc-hqc.org/doc/hqc-specification_2025-02-19.pdf).
- [32] W. Cary Huffman and Vera Pless. *Fundamentals of Error-Correcting Codes*. Cambridge: Cambridge University Press, 2003. ISBN: 978-0-521-13170-4. DOI: 10.1017/CB09780511807077. URL: <https://www.cambridge.org/core/books/fundamentals-of-errorcorrecting-codes/BF3AFDFB539C3C023BBD9DCBA4CDA761> (visited on 04/23/2025).
- [33] Jørn Justesen and Tom Høholdt. *A Course in Error-correcting Codes*. en. Google-Books-ID: y1a\_AESctqQC. European Mathematical Society, 2004. ISBN: 978-3-03719-001-2.
- [34] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. Chantilly, NETHERLANDS, THE: Elsevier Science & Technology, 1977. ISBN: 978-0-08-095423-3. URL: <http://ebookcentral.proquest.com/lib/aalborguniv-ebooks/detail.action?docID=648815> (visited on 04/23/2025).
- [35] National Institute of Standards and Technology (US). *SHA-3 standard : permutation-based hash and extendable-output functions*. en. Tech. rep. error: 202. Washington, D.C.: National Institute of Standards and Technology (U.S.), 2015, error: 202. DOI: 10.6028/NIST.FIPS.202. URL: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf> (visited on 04/15/2025).
- [36] Kenneth Rosen. *Discrete Mathematics and Its Applications Seventh Edition*. English. 7th edition. New York, NY: McGraw Hill, June 2011. ISBN: 978-0-07-338309-5.
- [37] Michael Sipser. *Introduction to the Theory of Computation*. English. 3rd edition. Australia Brazil Japan Korea Mexiko Singapore Spain United Kingdom United States: Cengage Learning, June 2012. ISBN: 978-1-133-18779-0.
- [38] Carlos Aguilar et al. *Efficient Encryption from Random Quasi-Cyclic Codes*. Published: Cryptology ePrint Archive, Report 2016/1194. 2016.
- [39] Hongyi Pan, Diaa Dabawi, and Ahmet Enis Cetin. *Fast Walsh-Hadamard Transform and Smooth-Thresholding Based Binary Layers in Deep Neural Networks*. arXiv:2104.07085 [cs]. Oct. 2021. DOI: 10.48550/arXiv.2104.07085. URL: <http://arxiv.org/abs/2104.07085> (visited on 05/14/2025).

# A Complexity Analysis Foundations

The computational complexity of an algorithm refers to the resources required to solve a problem as a function of the input size [36, pp. 218–219]. Two common measures are time complexity, which quantifies the amount of computational time required, and space complexity, which describes the amount of memory consumed during execution [36, p. 219].

## A.1 Big- $\mathcal{O}$ Notation

Big- $\mathcal{O}$  notation is used in asymptotic complexity analysis to describe how the time or space requirements of an algorithm grow as the input size increases [36, p. 205]. This method gives an upper bound on the worst-case growth rate of a function [37, pp. 276–277].

### Definition A.1 (Big- $\mathcal{O}$ Notation)

Let  $f, g : \mathbb{Z} \rightarrow \mathbb{R}$ . The function  $f(x)$  is said to be big- $\mathcal{O}$  of  $g(x)$ , denoted by  $f(x) = \mathcal{O}(g(x))$ , if there exists constants  $C > 0$  and  $k \in \mathbb{R}$  such that

$$\forall x \in \mathbb{R}, |x| > k \Rightarrow |f(x)| \leq C|g(x)|. \quad [36, \text{p. 205}]$$

Note that the equality symbol in Definition A.1 does not represent a genuine equality. Instead, the notation indicates that an inequality holds between the functions  $f(x)$  and  $g(x)$  for sufficiently large values of  $x$  [36, p. 207].

Many algorithms consist of multiple subproblems. The overall number of steps required is determined by the combined cost of these components.

### Theorem A.2 (Sum Rule)

Let  $f_1(x) = \mathcal{O}(g_1(x))$  and  $f_2(x) = \mathcal{O}(g_2(x))$ . Then

$$f_1(x) + f_2(x) = \mathcal{O}(\max(|g_1(x)|, |g_2(x)|)). \quad [36, \text{p. 213}]$$

### Proof

Let  $f_1(x) = \mathcal{O}(g_1(x))$  and  $f_2(x) = \mathcal{O}(g_2(x))$ . By Definition A.1, there exists positive constants  $C_1, C_2$  and thresholds  $k_1, k_2$  such that:

$$\begin{aligned} |f_1(x)| &\leq C_1|g_1(x)| \quad \text{for all } x > k_1, \\ |f_2(x)| &\leq C_2|g_2(x)| \quad \text{for all } x > k_2. \end{aligned}$$

To estimate the growth of the sum  $f_1(x) + f_2(x)$ , apply the triangle inequality:

$$|f_1(x) + f_2(x)| \leq |f_1(x)| + |f_2(x)|.$$

### A.1. Big- $\mathcal{O}$ Notation

When  $x > \max(k_1, k_2)$ , both bounds apply, so

$$|f_1(x)| + |f_2(x)| \leq C_1|g_1(x)| + C_2|g_2(x)|.$$

Let  $g(x) = \max(|g_1(x)|, |g_2(x)|)$ . Then

$$C_1|g_1(x)| + C_2|g_2(x)| \leq (C_1 + C_2)g(x) = Cg(x),$$

where  $C = C_1 + C_2$ . Therefore,

$$|f_1(x) + f_2(x)| \leq C|g(x)| \quad \text{for all } x > k,$$

where  $k = \max(k_1, k_2)$ . This establishes that

$$f_1(x) + f_2(x) = \mathcal{O}(g(x)),$$

where  $g(x) = \max(|g_1(x)|, |g_2(x)|)$ . ■

The following result is a direct consequence of the sum rule and applies in the common case where both functions share the same big- $\mathcal{O}$  bound.

#### Corollary A.3

Let  $f_1(x) = \mathcal{O}(g(x))$  and  $f_2(x) = \mathcal{O}(g(x))$ . Then

$$f_1(x) + f_2(x) = \mathcal{O}(g(x)). \quad [36, \text{p. 213}]$$

#### Proof

Suppose  $f_1(x) = \mathcal{O}(g(x))$  and  $f_2(x) = \mathcal{O}(g(x))$ . Applying Theorem A.2 it follows that

$$f_1(x) + f_2(x) = \mathcal{O}(\max(|g(x)|, |g(x)|)) = \mathcal{O}(g(x)). \quad \blacksquare$$

The next result addresses the asymptotic growth of the product of two functions.

#### Theorem A.4 (Product Rule)

Let  $f_1(x) = \mathcal{O}(g_1(x))$  and  $f_2(x) = \mathcal{O}(g_2(x))$ . Then

$$f_1(x)f_2(x) = \mathcal{O}(g_1(x)g_2(x)). \quad [36, \text{p. 213}]$$

#### Proof

Assume that  $f_1(x) = \mathcal{O}(g_1(x))$  and  $f_2(x) = \mathcal{O}(g_2(x))$ . Let  $k = \max(k_1, k_2)$  and  $C = C_1C_2$ .

## A.2. Divide-and-Conquer Algorithms

Then for all  $x > k$ :

$$\begin{aligned} |f_1(x)f_2(x)| &= |f_1(x)||f_2(x)| \\ &\leq C_1|g_1(x)|C_2|g_2(x)| \\ &= C_1C_2|g_1(x)g_2(x)| \\ &= C|g_1(x)g_2(x)|. \end{aligned}$$

This satisfies Definition A.1, so

$$f_1(x)f_2(x) = \mathcal{O}(g_1(x)g_2(x)),$$

which concludes the proof. ■

Both ML-KEM and HQC rely on divide-and-conquer algorithms, a key concept that plays a crucial role in their respective complexity analyses.

## A.2 Divide-and-Conquer Algorithms

Divide-and-conquer algorithms solve a problem of size  $n$  by dividing it into  $a$  subproblems of size  $\frac{n}{b}$ , assuming for simplicity that  $n$  is divisible by  $b$ . The solutions to these subproblems are then combined using  $g(n)$  operations. If  $f(n)$  denotes the total number of operations required, the process can be described by the recurrence relation

$$f(n) = af\left(\frac{n}{b}\right) + g(n),$$

known as a divide-and-conquer recurrence [36, p. 527]. Solving such a recurrence yields an asymptotic bound on the algorithm's computational complexity, describing how the cost scales with input size [36, p. 528].

### Theorem A.5 (Master Theorem)

Let  $f(n)$  be an increasing function satisfying the recurrence

$$f(n) = af\left(\frac{n}{b}\right) + cn^d \tag{A.1}$$

for  $n = b^k$ , where  $k$  is a positive integer,  $a \geq 1$ ,  $b$  is an integer greater than 1, and  $c, d \in \mathbb{R}$  satisfy  $c > 0$  and  $d \geq 0$ . Then

$$f(n) = \begin{cases} \mathcal{O}(n^d) & \text{if } a < b^d, \\ \mathcal{O}(n^d \log(n)) & \text{if } a = b^d, \\ \mathcal{O}(n^{\log_b(a)}) & \text{if } a > b^d. \end{cases}$$

[36, p. 532]

## A.2. Divide-and-Conquer Algorithms

### Proof

Let  $k$  be a positive integer, and let  $a \geq 1$ ,  $b > 1$ ,  $c > 0$ ,  $d \geq 0$  be constants. Consider an increasing function  $f(n)$  that satisfies the recurrence in (A.1) for inputs of the form  $n = b^k$ .

To solve the recurrence, it is expanded recursively. First, the recurrence is substituted into itself. Next, substitute  $f\left(\frac{n}{b}\right)$  using the same recurrence relation

$$f\left(\frac{n}{b}\right) = af\left(\frac{n}{b^2}\right) + c\left(\frac{n}{b}\right)^d. \quad (\text{A.2})$$

Inserting (A.2) into (A.1) yields

$$f(n) = a\left(af\left(\frac{n}{b^2}\right) + c\left(\frac{n}{b}\right)^d\right) + cn^d.$$

The process continues recursively, and after  $k$  steps, the following expression is obtained

$$f(n) = a^k f\left(\frac{n}{b^k}\right) + c \sum_{i=0}^{k-1} a^i \left(\frac{n}{b^i}\right)^d.$$

At this point,  $f\left(\frac{n}{b^k}\right)$  corresponds to the smallest subproblem, which is equal to  $f(1)$ . The expression can be rewritten as

$$f(n) = a^k f(1) + cn^d \sum_{i=0}^{k-1} \left(\frac{a}{b^d}\right)^i. \quad (\text{A.3})$$

To analyse the behaviour of the sum in (A.3), introduce the ratio

$$r = \frac{a}{b^d}, \quad (\text{A.4})$$

so that the summation becomes a geometric series

$$\sum_{i=0}^{k-1} \left(\frac{a}{b^d}\right)^i = \sum_{i=0}^{k-1} r^i.$$

The asymptotic behaviour of  $f(n)$  now depends on the value of the ratio in (A.4). The remainder of the proof considers three distinct cases, determined by whether  $r < 1$ ,  $r = 1$ , or  $r > 1$ , corresponding to  $a < b^d$ ,  $a = b^d$ , and  $a > b^d$ , respectively.

**Case 1:** Suppose  $r < 1$ . Then, by the formula for the sum of a convergent infinite geometric series,

$$\sum_{i=0}^{k-1} r^i \leq \sum_{i=0}^{\infty} r^i = \frac{1}{1-r}.$$

Thus, the recurrence in (A.3) simplifies to

$$f(n) = \mathcal{O}\left(n^{\log_b(a)}\right) + \mathcal{O}\left(n^d\right).$$

## A.2. Divide-and-Conquer Algorithms

Since  $a < b^d \Rightarrow \log_b(a) < d$ , it follows that  $n^{\log_b(a)}$  grows asymptotically no faster than  $n^d$ . Therefore, the dominant term in the sum is  $\mathcal{O}(n^d)$ , and thus  $f(n) = \mathcal{O}(n^d)$ , by Theorem A.2.

**Case 2:** Suppose  $r = 1$ . Then the sum

$$\sum_{i=0}^{k-1} r^i = \sum_{i=0}^{k-1} 1 = k = \log_b(n).$$

Thus, the recurrence becomes

$$f(n) = \mathcal{O}\left(n^{\log_b(a)}\right) + \mathcal{O}\left(n^d \log_b(n)\right).$$

Since  $r = 1$ , it follows that  $a = b^d$ , which implies  $\log_b(a) = d$ . Therefore, the two exponents are equal, and  $n^{\log_b(a)} = n^d$ . Thus, by Theorem A.2, the overall complexity simplifies to  $f(n) = \mathcal{O}(n^d \log(n))$ , where the change from  $\log_b(n)$  to  $\log(n)$  absorbs a constant factor, as logarithms with different bases differ by a constant multiple.

**Case 3:** Suppose  $r > 1$ . Then the sum  $\sum_{i=0}^{k-1} r^i$  is a finite geometric series with ratio  $r > 1$ , and satisfies

$$\sum_{i=0}^{k-1} r^i = \frac{r^k - 1}{r - 1}.$$

Since  $r - 1$  is a positive constant and  $r^k$  dominates  $-1$  as  $k$  increases, the sum simplifies to

$$\sum_{i=0}^{k-1} r^i = \mathcal{O}(r^k).$$

Thus, the recurrence becomes

$$f(n) = \mathcal{O}\left(n^{\log_b(a)}\right) + \mathcal{O}\left(n^d r^k\right).$$

Since  $r = \frac{a}{b^d}$ , it follows that

$$r^k = \left(\frac{a}{b^d}\right)^k = a^k b^{-dk} = a^{\log_b(n)} b^{-d \log_b(n)} = n^{\log_b(a)} n^{-d} = n^{\log_b(a) - d}.$$

Substituting back, the recurrence simplifies to

$$f(n) = \mathcal{O}\left(n^{\log_b(a)}\right) + \mathcal{O}\left(n^d n^{\log_b(a) - d}\right) = \mathcal{O}\left(n^{\log_b(a)}\right) + \mathcal{O}\left(n^{\log_b(a)}\right).$$

Since both terms are  $\mathcal{O}(n^{\log_b(a)})$ , the overall complexity is  $f(n) = \mathcal{O}(n^{\log_b(a)})$ , by Corollary A.3.

Walking through all three cases concludes the proof of the Master Theorem. ■

# B ML-KEM Complexity Analysis

This chapter presents a detailed analysis of the complexity of ML-KEM, as introduced in Chapter 3.

## B.1 Time Complexity Analysis of ML-KEM

The asymptotic time complexity of ML-KEM is influenced by the parameters  $n$  and  $k$ , where  $n - 1$  denotes the degree of the polynomials and  $k$  is the number of polynomials in the matrix and vectors used in the scheme.

### Key Generation

In the key generation algorithm, generating the matrix  $\mathbf{A} \in R_q^{k \times k}$  involves producing all  $k^2 n$  coefficients. By Theorem A.4, the time complexity of this process is

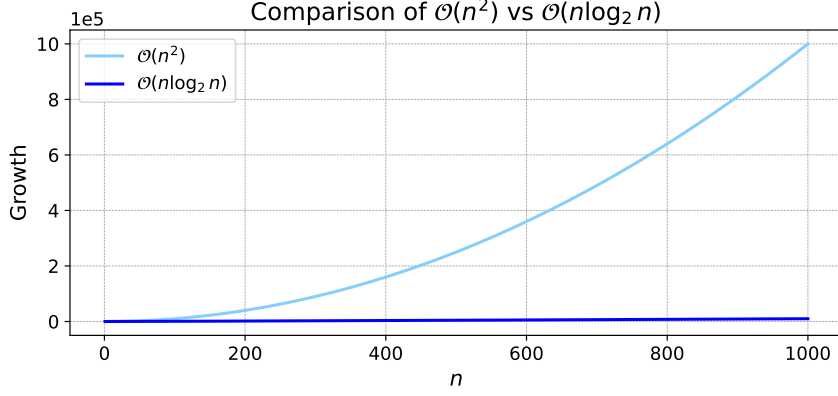
$$\mathcal{O}(k^2) \mathcal{O}(n) = \mathcal{O}(k^2 n). \quad (\text{B.1})$$

Next, the secret vector  $\mathbf{s} \in R_q^k$  and the error vector  $\mathbf{e} \in R_q^k$  are each sampled independently from the central binomial distribution, see Section 3.1. The total time complexity for generating both vectors is

$$\mathcal{O}(kn) + \mathcal{O}(kn) = \mathcal{O}(kn), \quad (\text{B.2})$$

by Corollary A.3. The computation of  $\mathbf{t} \in R_q^k$  in (3.7) is carried out in the NTT domain. The NTT follows a divide-and-conquer approach, as described in detail in Section 3.5. Using Theorem A.5, the recurrence  $f(n) = 2f(\frac{n}{2}) + \mathcal{O}(n)$  falls under case 2, where  $a = b^d$ . According to case 2, the solution to this recurrence is  $f(n) = \mathcal{O}(n \log_2(n))$ . Once the data is in the NTT domain, the multiplication of polynomials can be performed in constant time relative to the size of the polynomial. In contrast, classical polynomial multiplication takes  $\mathcal{O}(n^2)$  operations. Thus, the NTT-based approach provides better asymptotic performance for polynomial multiplication, see Figure B.1.

### B.1. Time Complexity Analysis of ML-KEM



**Figure B.1:** The figure compares the time complexity of multiplication in the NTT and classic domains. In the NTT domain, the complexity is  $\mathcal{O}(n \log_2(n))$ , while in the classic domain, it is  $\mathcal{O}(n^2)$ .

The NTT is applied element-wise to the  $k$  polynomials in  $\mathbf{s}$  and  $\mathbf{e}$ , see Table 3.4. Hence, the total cost of transforming these vectors is  $\mathcal{O}(kn \log_2(n))$ . Similarly, the matrix  $\mathbf{A}$  requires  $\mathcal{O}(k^2n \log_2(n))$  time. Combining these, the overall time complexity for all NTT operations in the key generation algorithm is

$$\mathcal{O}(kn \log_2(n)) + \mathcal{O}(kn \log_2(n)) + \mathcal{O}(k^2n \log_2(n)) = \mathcal{O}(k^2n \log_2(n)),$$

since the last term dominates for  $k > 1$ , as formalised in Theorem A.2.

After the NTT transformations, the matrix-vector multiplication in (3.7) is performed in the NTT domain, as specified in Table 3.4. Since there are  $k^2n$  such multiplications, the total time complexity for this operation is  $\mathcal{O}(k^2n)$ , as established by Theorem A.4. The addition of the error vector  $\hat{\mathbf{e}}$  adds a complexity of  $\mathcal{O}(kn)$ , but it does not affect the overall complexity, which remains  $\mathcal{O}(k^2n)$  by Theorem A.2.

Finally, constructing the decapsulation key in (3.13) involves generating a fixed-size random value  $z \in \{0,1\}^{256}$ , which requires constant time  $\mathcal{O}(1)$ , and computing the hash  $\mathcal{H}(ek)$ , which has complexity  $\mathcal{O}(kn)$ . Therefore, the overall time complexity for decapsulation key construction is  $\mathcal{O}(kn)$ .

The total time complexity for key generation is the sum of the individual complexities:

$$\mathcal{O}(k^2n) + \mathcal{O}(kn) + \mathcal{O}(k^2n \log_2(n)) + \mathcal{O}(k^2n) + \mathcal{O}(kn) = \mathcal{O}(k^2n \log_2(n)),$$

where the  $\mathcal{O}(k^2n \log_2(n))$  term dominates for  $n > 1$ , as confirmed by Theorem A.2.

### Encapsulation

In the first step of encapsulation, a random message representative  $m \in \{0,1\}^n$  is sampled, which requires  $\mathcal{O}(n)$  time. The shared secret key  $K \in \{0,1\}^{256}$  and encryption randomness  $r \in \{0,1\}^{256}$  are then derived according to (3.14). Computing  $\mathcal{H}(ek)$  has a complexity of  $\mathcal{O}(kn)$ . The hash function  $\mathcal{G}$  is applied to the message  $m$  and  $\mathcal{H}(ek)$ , which has a fixed

### B.1. Time Complexity Analysis of ML-KEM

size, so this step contributes an additional  $\mathcal{O}(n)$  time. The total time complexity of this preprocessing step is

$$\mathcal{O}(kn) + \mathcal{O}(n) = \mathcal{O}(kn).$$

In the subsequent encryption step, the matrix  $\mathbf{A}$  is regenerated from the public seed  $\rho \in \{0, 1\}^{256}$ , with the same time complexity as in key generation, see (B.1). The matrix is then transformed into the NTT domain as  $\hat{\mathbf{A}}$ , which requires a time complexity of  $\mathcal{O}(k^2 n \log_2(n))$ . The vectors  $\mathbf{r} \in R_q^k$  and  $\mathbf{e}_1 \in R_q^k$ , and polynomial  $e_2 \in R_q$  are sampled analogously to the key generation algorithm, with identical overall time complexity as given in (B.2). Next, the NTT is applied to the vector  $\mathbf{r}$  resulting in a time complexity of  $\mathcal{O}(kn \log_2(n))$ .

The first ciphertext component  $\mathbf{u} \in R_q^k$  is computed in (3.8). The matrix-vector multiplication is carried out in the NTT domain, as specified in Table 3.4. This computation involves  $k^2$  pointwise polynomial multiplications, followed by an inverse NTT on each resulting polynomial to return to the coefficient domain. Finally, the error vector  $\mathbf{e}_1$  is added, completing the construction of  $\mathbf{u}$ . The time complexity of these operations is

$$\mathcal{O}(k^2 n) + \mathcal{O}(kn \log_2(n)) + \mathcal{O}(kn) = \mathcal{O}(kn \log_2(n)). \quad (\text{B.3})$$

The dominant term depends on the relationship between  $k$  and  $\log_2(n)$ . Under the typical parameter choices in ML-KEM, where  $k < \log_2(n)$  in every case, the second term  $\mathcal{O}(kn \log_2(n))$  dominates.

The computation of the second ciphertext component  $v \in R_q$  in (3.9) also takes place in the NTT domain, as shown in Table 3.4. This step involves  $k$  polynomial multiplications, an inverse NTT, and an addition with the terms  $e_2 + \lceil \frac{q}{2} \rceil$ . The associated time complexity is

$$\mathcal{O}(kn) + \mathcal{O}(n \log_2(n)) + \mathcal{O}(n) = \mathcal{O}(n \log_2(n)).$$

Similar to the previous case, the dominant term depends on the relationship between  $k$  and  $\log_2(n)$ . Under the typical parameter choices in ML-KEM, where  $k < \log_2(n)$  in every case, the second term  $\mathcal{O}(n \log_2(n))$  dominates.

Finally, the ciphertext components  $\mathbf{u}$  and  $v$  are compressed by (3.1), which operates linearly over the number of coefficients, contributing a complexity of  $\mathcal{O}(kn)$ . The total time complexity for the encapsulation algorithm is then:

$$\begin{aligned} &\mathcal{O}(kn) + \mathcal{O}(k^2 n \log_2(n)) + \mathcal{O}(kn) + \mathcal{O}(kn \log_2(n)) + \mathcal{O}(kn \log_2(n)) \\ &+ \mathcal{O}(n \log_2(n)) + \mathcal{O}(kn) = \mathcal{O}(k^2 n \log_2(n)), \end{aligned}$$

where the second term dominates for  $k > 1$ , as confirmed by Theorem A.2.

### Decapsulation

In the decapsulation algorithm, the ciphertext components  $c_1$  and  $c_2$  are decompressed to recover  $\bar{\mathbf{u}} \in R_q^k$  and  $\bar{v} \in R_q$ , as shown in (3.10) and (3.11). The decompression function

## B.2. Space Complexity Analysis of ML-KEM

operates pointwise on the coefficients of each polynomial. The total time complexity of decompressing  $c_1$  and  $c_2$  is

$$\mathcal{O}(kn) + \mathcal{O}(n) = \mathcal{O}(kn).$$

Next, each polynomial in  $\bar{\mathbf{u}}$  is subsequently transformed into the NTT domain. Applying the NTT to a vector of  $k$  polynomials, each of degree  $n - 1$ , has a time complexity of  $\mathcal{O}(kn \log_2(n))$ . Following this, the inner product between the private decryption key  $\hat{\mathbf{s}}$  and the transformed ciphertext component  $\hat{\mathbf{u}}$  in (3.12) is computed. This entails  $kn$  pointwise multiplications and  $n$  additions to sum the  $k$  products. The total time complexity of this operation is therefore  $\mathcal{O}(kn)$ . The result of the inner product is a single polynomial, which is then transformed back from the NTT domain using the inverse NTT. Since the inverse NTT operates on one polynomial, the time complexity is  $\mathcal{O}(n \log_2(n))$ .

The next step is message recovery as defined in (3.12), where the polynomial resulting from the inner product is subtracted from  $\bar{v}$  and then compressed. Both the subtraction and compression operate coefficient-wise over polynomials of degree  $n - 1$ , yielding  $\mathcal{O}(n)$  time complexity. After obtaining  $m' \in \{0, 1\}^n$ , the hash function  $\mathcal{G}$  from (3.4) computes the shared secret key candidate  $K' \in \{0, 1\}^{256}$  and randomness  $r' \in \{0, 1\}^{256}$  in  $\mathcal{O}(kn)$  time, due to the hashed input size. The implicit rejection value  $z$  from the private decapsulation key  $dk$  is used to derive the alternative key  $\bar{K} \in \{0, 1\}^{256}$  as defined in (3.15). Since  $c$  has total size proportional to  $\mathcal{O}(kn)$ , the hash function  $\mathcal{J}$  defined in (3.6), runs in  $\mathcal{O}(kn)$  time.

Using the public encryption key  $ek$  and the derived randomness  $r'$ , the message  $m'$  is re-encrypted to produce a ciphertext  $c'$ . As detailed in the time complexity analysis of key encapsulation, the dominant cost in the encryption process arises from the NTT operations, contributing to the same time complexity as shown in (B.3). Finally, the received ciphertext  $c$  is compared to the recomputed ciphertext  $c'$  to verify correctness. This is a component-wise comparison of polynomials and thus takes  $\mathcal{O}(kn)$ . Summing all steps, the total time complexity of the decapsulation algorithm is  $\mathcal{O}(kn \log_2(n))$ .

## B.2 Space Complexity Analysis of ML-KEM

In ML-KEM, the parameters influencing the asymptotic space complexity are  $k$  and  $n$ . Since the time complexity was already examined in detail, demonstrating the use of big- $\mathcal{O}$  notation and its underlying principles, this section adopts a more concise approach.

Although NTTs are central to ML-KEM, they are typically performed in-place and do not contribute additional asymptotic memory. As such, they are not considered in the space complexities presented here.

### Key Generation

Table B.1 summarises the space complexities for key components in the algorithm.

## B.2. Space Complexity Analysis of ML-KEM

Component	Space Complexity
Matrix $\mathbf{A}$	$\mathcal{O}(n)$
Private decryption key $\mathbf{s}$	$\mathcal{O}(kn)$
Error vector $\mathbf{e}$	$\mathcal{O}(kn)$
Result $\mathbf{t}$	$\mathcal{O}(kn)$
Encapsulation key $ek$	$\mathcal{O}(kn)$
Decapsulation key $dk$	$\mathcal{O}(kn)$

**Table B.1:** Space complexity of the key generation algorithm in ML-KEM.

The matrix  $\mathbf{A}$  is not stored in full, but instead individual polynomials are deterministically reconstructed as needed during computation, thus, requiring a space complexity of  $\mathcal{O}(n)$  [5]. The overall space complexity for key generation is dominated by the storage of  $\mathbf{s}$ ,  $\mathbf{e}$ , and  $\mathbf{t}$ , leading to a total space complexity of  $\mathcal{O}(kn)$ .

## Encapsulation

A summary of the space complexities for key components is shown in Table B.2.

Component	Space Complexity
Message $m$	$\mathcal{O}(n)$
Encapsulation key $ek$	$\mathcal{O}(kn)$
Matrix $\mathbf{A}$	$\mathcal{O}(n)$
Random vector $\mathbf{r}$	$\mathcal{O}(kn)$
Error vector $\mathbf{e}_1$	$\mathcal{O}(kn)$
Error polynomial $e_2$	$\mathcal{O}(n)$
Ciphertext component $\mathbf{u}$	$\mathcal{O}(kn)$
Ciphertext component $v$	$\mathcal{O}(n)$
Ciphertext $c$	$\mathcal{O}(kn)$

**Table B.2:** Space complexity of the encapsulation algorithm in ML-KEM.

The ciphertext components  $\mathbf{u}$  and  $v$  occupy  $\mathcal{O}(kn)$  and  $\mathcal{O}(n)$  space, respectively. Their compressed counterparts,  $c_1$  and  $c_2$ , retain the same asymptotic space requirements, resulting

### B.3. Bit Complexity Analysis of ML-KEM

in an overall ciphertext space complexity of  $\mathcal{O}(kn)$  for  $c$ . This is because compression only reduces the bit length of each coefficient by a constant factor, without changing the number of coefficients. As asymptotic analysis abstracts away constant factors, the overall space complexity remains unchanged. Thus, the total space complexity is dominated by the storage of vectors, leading to an overall complexity of  $\mathcal{O}(kn)$ .

### Decapsulation

Table B.3 summarises the space usage of the relevant components.

Component	Space Complexity
Decapsulation key $dk$	$\mathcal{O}(kn)$
Private decryption key $\mathbf{s}$	$\mathcal{O}(kn)$
Decompressed $\bar{\mathbf{u}}$	$\mathcal{O}(kn)$
Decompressed $\bar{v}$	$\mathcal{O}(n)$
Recovered message $m'$	$\mathcal{O}(n)$
Encapsulation key $ek$	$\mathcal{O}(kn)$
K-PKE encryption	$\mathcal{O}(kn)$
Ciphertext $c$	$\mathcal{O}(kn)$
New ciphertext $c'$	$\mathcal{O}(kn)$

**Table B.3:** Space complexity of the decapsulation algorithm in ML-KEM.

The dominant terms arise from the storage of vectors and the K-PKE algorithm, resulting in total space complexity  $\mathcal{O}(kn)$ .

## B.3 Bit Complexity Analysis of ML-KEM

This section provides a bit-level complexity analysis, offering a more precise view than asymptotic analyses. It quantifies both the number of bits needed to perform operations and to store data. The bit complexities are expressed in terms of the algorithm parameters. The modulus  $q = 3329$  determines the bit-width required to represent each coefficient in the ring  $R_q$ . The number of bits needed to represent an element modulo  $q$  is calculated as

$$\lfloor \log_2(3329) \rfloor + 1 = 12 \text{ bits.}$$

Thus, each coefficient in  $R_q$  is stored using 12 bits.

### Time Bit Complexity Analysis

The time bit complexity analysis expands on the asymptotic results from Section B.1, providing concrete estimates of bit-level operations per step. While it excludes practical factors such as hardware details and coding optimisations, it offers a consistent basis for comparing ML-KEM and HQC computational costs. Table B.4 details the time bit complexity for the main operations in the ML-KEM key generation algorithm. Similarly, Table B.5 and Table B.6 present the time bit complexities for the encapsulation and decapsulation algorithms. The terms  $\alpha_{\eta_1}$  and  $\alpha_{\eta_2}$  denote the number of bits required to encode coefficients sampled from  $B_{\eta_1}$  and  $B_{\eta_2}$ , respectively. They are computed as

$$\lfloor \log_2(\eta_1) \rfloor + 1 = \alpha_{\eta_1} \quad \text{and} \quad \lfloor \log_2(\eta_2) \rfloor + 1 = \alpha_{\eta_2}.$$

Operation	Time Bit Complexity
Generate matrix $\mathbf{A}$	$12k^2n$ bits
Sample $\mathbf{s}, \mathbf{e}$ from $B_\eta$	$kn(\alpha_{\eta_1} + \alpha_{\eta_2})$ bits
NTT of $\mathbf{A}$	$12k^2n \log_2(n)$ bits
NTT of $\mathbf{s}, \mathbf{e}$	$kn \log_2(n)(\alpha_{\eta_1} + \alpha_{\eta_2})$ bits
Matrix-vector multiplication + addition	$12k^2n + kn\alpha_{\eta_2}$ bits
Hash $\mathcal{H}(ek)$ and rejection $z$	$12kn + 256$ bits

**Table B.4:** Time bit complexity of the ML-KEM key generation algorithm.

### B.3. Bit Complexity Analysis of ML-KEM

Operation	Time Bit Complexity
Sample message $m$	$n$ bits
Shared secret key $K$ and randomness $r$	512 bits
Hashes $\mathcal{H}(ek)$ and $\mathcal{G}(m, \mathcal{H}(ek))$	$12(n + kn)$ bits
Generate matrix $\mathbf{A}$	$12k^2n$ bits
NTT of $\mathbf{A}$	$12k^2n \log_2(n)$ bits
Sample $\mathbf{r}, \mathbf{e}_1, \mathbf{e}_2$	$kn\alpha_{\eta_1} + kn\alpha_{\eta_2} + n\alpha_{\eta_2}$ bits
NTT of $\mathbf{r}$	$kn \log_2(n)\alpha_{\eta_1}$ bits
Calculate $\mathbf{u}$	$12k^2n + 12kn \log_2(n) + 12kn$ bits
Calculate $v$	$12kn + 12n \log_2(n) + 12n$ bits
Compress $\mathbf{u}$ and $v$	$12kn + 12n$ bits

**Table B.5:** Time bit complexity of the ML-KEM encapsulation algorithm.

Operation	Time Bit Complexity
Decompress $\mathbf{u}$ and $v$	$12(kn + n)$ bits
NTT on $\mathbf{u}$	$12kn \log_2(n)$ bits
Inner product of $\mathbf{s}$ and $\mathbf{u}$ plus inverse NTT	$12kn + 12n \log_2(n)$ bits
Subtract $v$ and compression	$24n$ bits
Hashes $\mathcal{H}(ek)$ and $\mathcal{G}(m, \mathcal{H}(ek))$	$12(n + kn)$ bits
Hash $\mathcal{J}(z, c)$	$12kn + 256$ bits
Re-encrypt $m'$	Same as the last five steps in Table B.5
Comparison of $c$ and $c'$	$12kn$ bits

**Table B.6:** Time bit complexity of the ML-KEM decapsulation algorithm.

### Space Bit Complexity Analysis

The space bit complexity analysis is not subdivided into key generation, encapsulation, and decapsulation, as in the previous sections. Instead, it builds directly on the space complexities already presented in Table B.1, Table B.2 and Table B.3, by converting those results into bit-level quantities. For instance, each entry in the matrix  $\mathbf{A}$ , which has a space complexity

### B.3. Bit Complexity Analysis of ML-KEM

of  $\mathcal{O}(n)$ , consists of elements in  $R_q$ , where each coefficient requires 12 bits for representation. Thus, the space bit complexity for each entry in  $\mathbf{A}$  is  $12n$  bits. The same reasoning applies to all vectors with space complexity  $\mathcal{O}(kn)$ , including  $\mathbf{s}$ ,  $\mathbf{e}$ ,  $\mathbf{t}$ ,  $\mathbf{r}$ ,  $\mathbf{e}_1$  and  $\mathbf{u}$ , each of which contributes a space bit complexity of

$$12kn \text{ bits.} \quad (\text{B.4})$$

The most relevant elements in the space bit complexity analysis, however, are the sizes of the public encapsulation key, the private decapsulation key, and the ciphertext. The public encapsulation key is defined as  $ek = (\rho, \mathbf{t})$ , where  $\rho$  is a 256-bit seed, and  $\mathbf{t}$  contains  $k$  polynomials in  $R_q$ , each with  $n$  coefficients of 12 bits. This leads to a total space bit complexity of

$$12kn + 256 \text{ bits.} \quad (\text{B.5})$$

The private decapsulation key  $dk$ , as defined in (3.13), consists of  $\mathbf{s}$ , which requires  $12kn$  bits as shown in (B.4), the private encapsulation key  $ek$ , which adds  $12kn + 256$  bits as given in (B.5), the hash function  $\mathcal{H}(ek)$  of 256 bits, and the 256-bit rejection value  $z$ . Summing these components yields a total space bit complexity for the private decapsulation key of

$$12kn + 12kn + 256 + 256 + 256 = 24kn + 768 \text{ bits.} \quad (\text{B.6})$$

The ciphertext  $c$  consists of two compressed components. The first ciphertext component compresses the vector  $\mathbf{u}$ , which contains  $k$  polynomials, each with  $n$  coefficients. Each coefficient is compressed to  $d_u$  bits, resulting in a total bit size of  $knd_u$ . The second ciphertext component compresses the polynomial  $v$ , which has  $n$  coefficients, each represented using  $d_v$  bits, giving a total of  $nd_v$  bits. Thus, the total space bit complexity of the ciphertext is

$$knd_u + nd_v \text{ bits.} \quad (\text{B.7})$$

The shared secret key has a fixed bit length of 256 bits, as defined in [5]. This concludes the complexity analysis of the ML-KEM algorithm. By substituting the concrete parameter values for each ML-KEM parameter set into the formulas (B.5), (B.6), and (B.7), the resulting bit sizes correspond to the byte values presented in Table 3.2.

# C HQC Complexity Analysis

This chapter presents a detailed analysis of the complexity of HQC, as introduced in Chapter 5.

## C.1 Time Complexity Analysis of HQC

The time complexity of HQC is influenced by the parameters  $n$ ,  $n_1$ ,  $n_2$ ,  $k_1$ , and  $k_2$ , where  $n$  is the length of the ambient space in which arithmetic operations are performed,  $n_1 n_2$  is the code length of the concatenated RMRS code, and  $k_1 k_2$  is the dimension of the message and seed vector as well as the dimension of the concatenated RMRS code.

### Key Generation

In the key generation algorithm,  $\mathbf{h} \in R_2$  is sampled uniformly at random from  $R_2$ . As  $\mathbf{h}$  is represented as a vector of length  $n$  corresponding to the coefficients of a polynomial of degree at most  $n - 1$ , generating  $\mathbf{h}$  requires producing  $n$  coefficients. Hence, this results in a time complexity of  $\mathcal{O}(n)$ .

Next, the two secret vectors  $(\mathbf{x}, \mathbf{y}) \in R_w \times R_w$  are sampled uniformly at random from  $R_w$ . Each secret vector consists of the coefficients of a sparse polynomial of degree at most  $n - 1$  with exactly  $w$  non-zero coefficients. Sampling such vectors involves selecting  $w$  distinct positions from  $\{0, 1, \dots, n - 1\}$  and setting those positions to 1. Thus, the time complexity of a sparse vector is proportional to the number of its non-zero elements. Therefore, this process requires a time complexity of  $\mathcal{O}(w)$ . However, according to [38], sparse vectors with Hamming weight  $w$  typically have a time complexity of  $\mathcal{O}(\sqrt{n})$  to balance security and efficiency. Therefore, the total time complexity for sampling both vectors is

$$\mathcal{O}(\sqrt{n}) + \mathcal{O}(\sqrt{n}) = \mathcal{O}(\sqrt{n}), \quad (\text{C.1})$$

by Corollary A.3. The next step is to compute  $\mathbf{s}$ , as defined in (5.3). This involves a multiplication between  $\mathbf{h}$  and  $\mathbf{y}$ , followed by an addition with  $\mathbf{x}$ . The time complexity for the multiplication depends on the method used. Using classical polynomial multiplication, the time complexity is

$$\mathcal{O}(n)\mathcal{O}(\sqrt{n}) = \mathcal{O}(n\sqrt{n}),$$

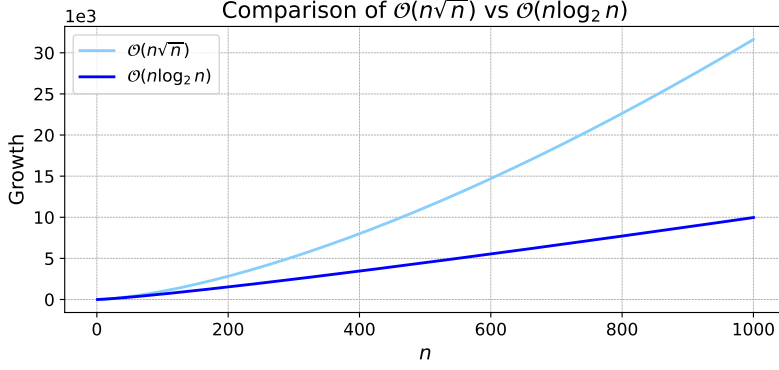
by Theorem A.4. The subsequent addition with  $\mathbf{x}$  is performed component-wise and has a linear-time complexity of  $\mathcal{O}(n)$ . Thus, the total time complexity to compute  $\mathbf{s}$  is

$$\mathcal{O}(n) + \mathcal{O}(n\sqrt{n}) = \mathcal{O}(n\sqrt{n}),$$

by Theorem A.2. Alternatively, the multiplication can be implemented using fast Fourier transform (FFT), which transforms the input vectors into the frequency domain to enable

### C.1. Time Complexity Analysis of HQC

pointwise multiplication. Similar to the NTT, the FFT follows a divide-and-conquer approach. Using Theorem A.5, the recurrence  $f(n) = 2f(\frac{n}{2}) + \mathcal{O}(n)$  falls under case 2, where  $a = b^d$ . According to case 2, the solution to this recurrence is  $f(n) = \mathcal{O}(n \log_2(n))$ . Compared to the classical polynomial multiplication with complexity  $\mathcal{O}(n\sqrt{n})$ , the FFT-based method achieves better asymptotic performance, as illustrated in Figure C.1.



**Figure C.1:** Comparison of the time complexity of multiplication  $\mathcal{O}(n\sqrt{n})$  vs.  $\mathcal{O}(n \log_2(n))$ .

Using the FFT, the overall time complexity of computing  $\mathbf{s}$  becomes

$$\mathcal{O}(\sqrt{n}) + \mathcal{O}(n \log_2(n)) = \mathcal{O}(n \log_2(n)), \quad (\text{C.2})$$

by Theorem A.2, as the last term dominates for large  $n$ .

Finally, to construct the decapsulation key, the vector  $\boldsymbol{\sigma} \in \mathbb{F}_2^{k_1 k_2}$  is sampled uniformly at random. Generating a vector of length  $k_1 k_2$  requires a time complexity of  $\mathcal{O}(k_1 k_2)$ .

The total time complexity for key generation is obtained by summing the individual complexities:

$$\mathcal{O}(n) + \mathcal{O}(\sqrt{n}) + \mathcal{O}(n \log_2(n)) + \mathcal{O}(k_1 k_2) = \mathcal{O}(n \log_2(n)),$$

where the third term dominates asymptotically under the assumption that  $k_1 k_2 \ll n$ , as established by Theorem A.2. This assumption holds for typical parameter choices in HQC.

### Encapsulation

In the encapsulation algorithm, a message  $\mathbf{m} \in \mathbb{F}_2^{k_1 k_2}$  is selected uniformly at random. As the message consist of  $k_1 k_2$  elements, a time complexity of  $\mathcal{O}(k_1 k_2)$  is required. A salt value is also sampled uniformly at random,  $\text{salt} \in \mathbb{F}_2^{128}$ , which results in a constant-time complexity of  $\mathcal{O}(1)$ . The randomness  $\theta$  is computed, as defined in (5.8). Here, the hash function defined in (5.1) is used, which processes input of size  $k_1 k_2 + 32 + 128$  bits and outputs a fixed length of 512 bits. As the time complexity of the firstBytes( $\cdot$ ) function and salt value is constant, the time complexity of computing  $\theta$  is  $\mathcal{O}(k_1 k_2)$ .

### C.1. Time Complexity Analysis of HQC

Next, the error vector  $\mathbf{e} \in R_{w_e}$  with Hamming weight  $w_e$  is sampled uniformly at random and the secret vectors  $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2) \in R_{w_r} \times R_{w_r}$ , consisting of two sparse vectors with Hamming weight  $w_r$  each, are sampled uniformly at random. Sampling these sparse vectors involves the same process as in the key generation algorithm, resulting in the same total time complexity as given in (C.1).

The first ciphertext component,  $\mathbf{u}$ , as defined in (5.4), includes a multiplication between  $\mathbf{h}$  and  $\mathbf{r}_2$ . As seen in the time complexity analysis of the key generation algorithm, using the FFT for the multiplication is more efficient than classical polynomial multiplication. The addition of  $\mathbf{r}_1$ , which involves at most  $w_r$  non-zero positions, requires a time complexity of  $\mathcal{O}(\sqrt{n})$ . Therefore, the total time complexity of  $\mathbf{u}$  is identical to the time complexity in (C.2).

To compute the second ciphertext component  $\mathbf{v}$ , as defined in (5.5), the matrix-vector multiplication  $\mathbf{m}\mathbf{G}$  is first computed. The generator matrix  $\mathbf{G} \in \mathbb{F}_2^{k_1 k_2 \times n_1 n_2}$  is generated for a concatenated RMRS  $[n_1 n_2, k_1 k_2]$  code. The generator matrix enables encoding of the message  $\mathbf{m}$  into a codeword, a process which involves multiplying a  $1 \times k_1 k_2$  vector by a  $k_1 k_2 \times n_1 n_2$  matrix, which requires a time complexity of

$$\mathcal{O}(1)\mathcal{O}(k_1 k_2)\mathcal{O}(n_1 n_2) = \mathcal{O}(k_1 k_2 n_1 n_2),$$

by Theorem A.4. Next, the multiplication between  $\mathbf{s}$  and  $\mathbf{r}_2$  with the addition of  $\mathbf{e}$  is equivalent to the process in the key generation algorithm, leading to the identical time complexity as given in (C.2). The truncation operation reduces the output with  $\ell$  bits and operates in constant time, thus contributing  $\mathcal{O}(1)$  to the time complexity. Therefore, the overall time complexity of  $\mathbf{v}$  is

$$\mathcal{O}(k_1 k_2 n_1 n_2) + \mathcal{O}(n \log_2(n)) + \mathcal{O}(\sqrt{n}) + \mathcal{O}(1) = \mathcal{O}(k_1 k_2 n_1 n_2) + \mathcal{O}(n \log_2(n)), \quad (\text{C.3})$$

as established by Theorem A.2. The dominant term depends on the relative sizes of  $k_1 k_2 n_1 n_2$  and  $n \log_2(n)$ . Under the typical parameter choices in HQC where  $k_1 k_2 \ll n_1 n_2 < n$ , the total time complexity of  $\mathbf{v}$  is dominated by  $\mathcal{O}(n \log_2(n))$ .

The total time complexity for the encapsulation algorithm is obtained by summing the dominant terms of each step:

$$\mathcal{O}(k_1 k_2) + \mathcal{O}(k_1 k_2) + \mathcal{O}(\sqrt{n}) + \mathcal{O}(n \log_2(n)) + \mathcal{O}(n \log_2(n)) = \mathcal{O}(n \log_2(n)),$$

where the  $\mathcal{O}(n \log_2(n))$  term dominates asymptotically under the assumption that  $k_1 k_2 \ll n$ , as established by Theorem A.2.

### Decapsulation

The decapsulation algorithm begins by decrypting the received ciphertext  $\mathbf{c}$  using the private decapsulation key  $sk$ . This includes the multiplication between  $\mathbf{u}$  and  $\mathbf{y}$  which is then subtracted from  $\mathbf{v}$ , as defined in (5.6). As explained in the key generation, using the FFT method is more efficient than classical polynomial multiplication. The subtraction from  $\mathbf{v}$

## C.2. Space Complexity Analysis of HQC

operates coefficient-wise and requires a time complexity of  $\mathcal{O}(n)$ . Therefore, the overall time complexity of computing the decoding input is identical to (C.2).

Decoding that input with a decoding algorithm for the concatenated RMRS code, the RM code is first decoded, then the RS code. The time complexity of decoding an RM code using the decoding algorithm as explained in [31, p. 26] requires  $\mathcal{O}(n_1 n_2 \log_2(n_1 n_2))$  [39]. The time complexity of decoding an RS code using the decoding algorithm as explained in [31, pp. 24–25] requires  $\mathcal{O}(n_1^2 n_2^2)$  [11, p. 201]. Therefore, the overall time complexity of the decoding algorithm using concatenated RMRS codes is

$$\mathcal{O}(n_1 n_2 \log_2(n_1 n_2)) + \mathcal{O}(n_1^2 n_2^2) = \mathcal{O}(n_1^2 n_2^2),$$

as established by Theorem A.2. Subsequently, the randomness  $\theta'$  is computed, as defined in (5.9). The time complexity of this operation is identical to the one of  $\theta$  in the encapsulation algorithm. Next, the sparse vectors  $\mathbf{e}'$ ,  $\mathbf{r}'_1$ , and  $\mathbf{r}'_2$  of Hamming weights  $w_e$ ,  $w_r$ , and  $w_r$ , respectively, is sampled uniformly at random. Sampling these vectors involves the same process as in the key generation algorithm, resulting in the same overall time complexity as given in (C.1). Similarly as in encapsulation, the ciphertext components  $\mathbf{u}'$  and  $\mathbf{v}'$ , as defined in (5.10) and (5.11), are computed, with the total complexity identical to (C.2) and (C.3), respectively.

Then, the re-encrypted ciphertext  $\mathbf{c}'$  is compared to the received ciphertext  $\mathbf{c}$ . This requires checking  $n$  bits in each component, leading to a time complexity of  $\mathcal{O}(n)$ . Depending on the comparison, the shared secret key is computed using the hash function  $\mathcal{K}$ , defined in (5.2), applied to either  $\mathbf{m}'$  or  $\sigma$ . As  $\mathcal{K}$  processes inputs of size  $k_1 k_2$ , this step has complexity  $\mathcal{O}(k_1 k_2)$ .

Summing the dominant terms from each step, the total time complexity of the decapsulation algorithm is:

$$\begin{aligned} & \mathcal{O}(n \log_2(n)) + \mathcal{O}(n_1^2 n_2^2) + \mathcal{O}(k_1 k_2) + \mathcal{O}(\sqrt{n}) + \mathcal{O}(n \log_2(n)) \\ & + \mathcal{O}(n \log_2(n)) + \mathcal{O}(n) + \mathcal{O}(k_1 k_2) = \mathcal{O}(n_1^2 n_2^2), \end{aligned}$$

where the second term dominates asymptotically, as established by Theorem A.2. This holds under the assumptions that  $k_1 k_2 \ll n$  and  $n \ll n_1^2 n_2^2$ , which are satisfied for typical HQC parameter choices.

## C.2 Space Complexity Analysis of HQC

The space complexity of HQC is influenced by the parameters  $n$ ,  $n_1$ ,  $n_2$ ,  $k_1$ , and  $k_2$ . Since the time complexity has already been examined in detail, including the use of big- $\mathcal{O}$  notation and its underlying principles, this section adopts a more concise approach, similar to the space complexity analysis of ML-KEM, as presented in Section B.2.

### Key Generation

Table C.1 summarises the space complexities for key components in the algorithm.

## C.2. Space Complexity Analysis of HQC

Component	Space Complexity
Vector $\mathbf{h}$	$\mathcal{O}(n)$
Secret vectors $(\mathbf{x}, \mathbf{y})$	$\mathcal{O}(\sqrt{n})$
Vector $\mathbf{s}$	$\mathcal{O}(n)$
Seed $\sigma$	$\mathcal{O}(k_1 k_2)$
Public encapsulation key $pk$	$\mathcal{O}(n)$
Private decapsulation key $sk$	$\mathcal{O}(\sqrt{n})$

**Table C.1:** Space complexity of the key generation algorithm in HQC.

The sparse vectors  $(\mathbf{x}, \mathbf{y})$  are stored by listing the positions of their non-zero entries, thus requiring a space complexity of  $\mathcal{O}(\sqrt{n})$  each. The private decapsulation key  $sk$  is formed by  $(\mathbf{x}, \mathbf{y})$  and  $\sigma$ , where the latter consists of  $k_1 k_2$  bits, requiring a space complexity of  $\mathcal{O}(k_1 k_2)$ . Thus, under the assumption that  $k_1 k_2 < \sqrt{n}$ , which holds for the typical HQC parameter choices, the private decapsulation key has a space complexity of  $\mathcal{O}(\sqrt{n})$ . The overall space complexity for key generation is dominated by the storage of  $\mathbf{h}$  and  $\mathbf{s}$ , leading to a total space complexity of  $\mathcal{O}(n)$ .

## Encapsulation

A summary of the space complexities for key components is shown in Table C.2.

Component	Space Complexity
Message $\mathbf{m}$	$\mathcal{O}(k_1 k_2)$
Matrix $\mathbf{G}$	$\mathcal{O}(k_1 k_2 n_1 n_2)$
Encoded message $\mathbf{mG}$	$\mathcal{O}(n_1 n_2)$
Error vector $\mathbf{e}$	$\mathcal{O}(\sqrt{n})$
Random vectors $(\mathbf{r}_1, \mathbf{r}_2)$	$\mathcal{O}(\sqrt{n})$
Ciphertext component $\mathbf{u}$	$\mathcal{O}(n)$
Ciphertext component $\mathbf{v}$	$\mathcal{O}(n_1 n_2)$
Ciphertext $\mathbf{c}$	$\mathcal{O}(n)$

**Table C.2:** Space complexity of the encapsulation algorithm in HQC.

### C.3. Bit Complexity Analysis of HQC

The ciphertext components  $\mathbf{u}$  and  $\mathbf{v}$  occupy  $\mathcal{O}(n)$  and  $\mathcal{O}(n_1 n_2)$  space each, respectively. Therefore, the total ciphertext space complexity is  $\mathcal{O}(n)$ , under the assumption that  $n_1 n_1 < n$ . The overall space complexity is dominated by the storage of vectors, leading to a total space complexity of  $\mathcal{O}(k_1 k_2 n_1 n_2)$ .

#### Decapsulation

Table C.3 summarises the space complexities for key components in the algorithm.

Component	Space Complexity
Private decapsulation key $sk$	$\mathcal{O}(\sqrt{n})$
Ciphertext $\mathbf{c}$	$\mathcal{O}(n)$
Difference and product $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$	$\mathcal{O}(n)$
Decoded message $\mathbf{m}'$	$\mathcal{O}(n)$
Re-computing $\mathbf{e}', \mathbf{r}'_1, \mathbf{r}'_2$	$\mathcal{O}(\sqrt{n})$
Re-encryption $\mathbf{u}'$	$\mathcal{O}(n)$
Re-encryption $\mathbf{v}'$	$\mathcal{O}(n_1 n_2)$
Recomputed ciphertext $\mathbf{c}'$	$\mathcal{O}(n)$

**Table C.3:** Space complexity of the decapsulation algorithm in HQC.

The total space complexity is dominated by the ciphertext, resulting in a total space complexity of  $\mathcal{O}(n)$ .

### C.3 Bit Complexity Analysis of HQC

This section provides a bit-level complexity analysis, offering a more precise view than asymptotic analyses. It quantifies both the number of bits needed to perform operations and to store data. The bit complexities are expressed in terms of the algorithm parameters.

#### Time Bit Complexity Analysis

The time bit complexity analysis expands on the asymptotic results from Section C.1, providing concrete estimates of bit-level operations per step. While it excludes practical factors such as hardware details and coding optimisations, it offers a consistent basis for comparing ML-KEM and HQC computational costs. Table C.4 details the time bit complexity for the main operations in the HQC key generation algorithm. Similarly, Table C.5 and Table C.6 present the time bit complexities for the encapsulation and decapsulation algorithms.

### C.3. Bit Complexity Analysis of HQC

Operation	Time Bit Complexity
Sample $\mathbf{h}$	$n$ bits
Sample sparse vectors $(\mathbf{x}, \mathbf{y})$	$2\sqrt{n}$ bits
Compute $\mathbf{s}$	$\sqrt{n} + n \log_2(n)$ bits
Sample $\sigma$	$k_1 k_2$ bits

**Table C.4:** Time bit complexity of the HQC key generation algorithm.

Operation	Time Bit Complexity
Select message $\mathbf{m}$	$k_1 k_2$ bits
Salt value $\text{salt}$	128 bits
Randomness $\theta$	$k_1 k_2 + 160$ bits
Sample sparse vectors $\mathbf{e}, \mathbf{r}_1$ and $\mathbf{r}_2$	$3\sqrt{n}$ bits
Calculate $\mathbf{u}$	$\sqrt{n} + n \log_2(n)$ bits
Calculate $\mathbf{v}$	$k_1 k_2 n_1 n_2 + \sqrt{n} + n \log_2(n)$ bits

**Table C.5:** Time bit complexity of the HQC encapsulation algorithm.

Operation	Time Bit Complexity
Calculate $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$	$\sqrt{n} + n \log_2(n)$ bits
Decode RMRS code	$n_1 n_2 \log_2(n_1 n_2) + n_1^2 n_2^2$ bits
Randomness $\theta'$	$k_1 k_2 + 160$ bits
Sample sparse vectors $\mathbf{e}', \mathbf{r}'_1$ and $\mathbf{r}'_2$	$3\sqrt{n}$ bits
Calculate $\mathbf{u}'$	$\sqrt{n} + n \log_2(n)$ bits
Calculate $\mathbf{v}'$	$k_1 k_2 n_1 n_2 + \sqrt{n} + n \log_2(n)$ bits
Comparison of $\mathbf{c}$ and $\mathbf{c}'$	$2n$ bits
Hash $\mathcal{K}$	$k_1 k_2$ bits

**Table C.6:** Time bit complexity of the HQC decapsulation algorithm.

### Space Bit Complexity Analysis

The space bit complexity analysis is not subdivided into key generation, encapsulation, and decapsulation, as in the previous sections. Instead, it builds directly on the space complexities. The most relevant elements in the space bit complexity analysis are the sizes of the public encapsulation key, the private decapsulation key, and the ciphertext.

Both the public encapsulation key and the private decapsulation key involve the use of a seed, which is deterministically expanded using the SHAKE256 function. This seed is initialised from a 40-byte string, corresponding to 320 bits [31].

The public encapsulation key  $pk$ , as defined in (5.3), consists of the vector  $\mathbf{h}$ , which is generated from the seed, and the vector  $\mathbf{s}$ , which requires  $\mathcal{O}(n)$  bits of storage. Therefore, the total space bit complexity of the public encapsulation key is

$$320 + n \text{ bits.} \quad (\text{C.4})$$

The private decapsulation key  $sk$ , as defined in (5.7), consists of the sparse vectors  $(\mathbf{x}, \mathbf{y})$ , which are generated from the seed, and the rejection value  $\sigma$ , which is  $k_1 k_2$ -bit. Hence, the total space bit complexity of the private decapsulation key is

$$320 + k_1 k_2 \text{ bits.} \quad (\text{C.5})$$

The ciphertext  $\mathbf{c} = (\mathbf{u}, \mathbf{v}, \text{salt})$  consists of  $\mathbf{u}$ , which has the space complexity  $\mathcal{O}(n)$ , the vector  $\mathbf{v}$ , which has the space complexity  $\mathcal{O}(n_1 n_2)$ , and the salt value, which is a fixed 128-bit value. Therefore, the total space bit complexity of the ciphertext is

$$n + n_1 n_2 + 128 \text{ bits.} \quad (\text{C.6})$$

The shared secret key has a fixed bit length of 512 bits, as specified in the HQC specification [31]. This concludes the complexity analysis of HQC. By substituting the concrete parameter values for each HQC parameter set into the equations (C.5), (C.4), and (C.6), the resulting bit sizes correspond to the byte-sized values presented in Table 5.4.