EFFICIENT BATCH ALGORITHMS FOR THE POST-QUANTUM CRYSTALS DILITHIUM SIGNATURE SCHEME AND CRYSTALS KYBER ENCRYPTION SCHEME

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

NAZLI DENİZ TÜRE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
CRYPTOGRAPHY

SEPTEMBER 2024

Approval of the thesis:

EFFICIENT BATCH ALGORITHMS FOR THE POST-QUANTUM CRYSTALS DILITHIUM SIGNATURE SCHEME AND CRYSTALS KYBER ENCRYPTION SCHEME

submitted by **NAZLI DENİZ TÜRE** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Cryptography Department, Middle East Technical University** by,

Prof. Dr. Ayşe Sevtap Kestel
Dean, Graduate School of **Applied Mathematics** _____

Assoc. Prof. Dr. Oğuz Yayla
Head of Department, **Cryptography** _____

Assoc. Prof. Dr. Oğuz Yayla
Supervisor, **Department of Cryptography, METU** _____

Prof. Dr. Murat Cenk
Co-supervisor, **Department of Cryptography, METU** _____

Examining Committee Members:

Prof. Dr. Zülfükar Saygı
Department of Mathematics, TOBB ETU _____

Assoc. Prof. Dr. Oğuz Yayla
Department of Cryptography, METU _____

Assoc. Prof. Dr. Fatih Sulak
Department of Mathematics, Atılım University _____

Assoc. Prof. Dr. Ahmet Sınak
Department of MIS, Akdeniz University _____

Assist. Prof. Dr. Buket Özkaya
Department of Cryptography, METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: NAZLI DENİZ TÜRE

Signature :

ABSTRACT

EFFICIENT BATCH ALGORITHMS FOR THE POST-QUANTUM CRYSTALS DILITHIUM SIGNATURE SCHEME AND CRYSTALS KYBER ENCRYPTION SCHEME

Türe, Nazlı Deniz
Ph.D., Department of Cryptography
Supervisor : Assoc. Prof. Dr. Oğuz Yayla
Co-Supervisor : Prof. Dr. Murat Cenk

September 2024, 119 pages

Digital signatures ensure authenticity and data integrity and are widely utilized in various fields such as information technologies, finance, education, and law. They are crucial in securing servers against cyber attacks by authenticating connections between clients and servers. Additionally, encryption is used in many areas, such as secure communication, cloud, server and database security to ensure data confidentiality. Public-key cryptosystems are widely used as digital signature and encryption mechanisms. The security of public-key cryptosystems is based on difficult mathematical problems such as the integer factorization problem and the discrete logarithm problem. Security of the public-key cryptosystems is based on solving challenging mathematical problems like the discrete logarithm and integer factorization problems. P. Shor (1999) has shown that high-capacity quantum computers can be used to break cryptosystems whose security depends on solving the discrete logarithm problem and the integer factorization problem. The post-quantum standardization procedure has been organized by NIST (National Institute of Standards and Technology, USA) to replace cryptographic algorithms that will lose their security with the next-generation quantum-secure algorithms. Following this procedure, Crystals Dilithium, Falcon, and SPHINCS+ are chosen as the digital signature standards, and Crystals Kyber is chosen as the encryption/KEM standard.

This work focuses on efficient batch signature generation with Dilithium, batch verification of signatures from the same user using Crystals Dilithium (NIST's post-quantum digital signature standard), and batch encryption to a single user with Crystals Kyber (NIST's post-quantum encryption/KEM standard). One of the main operations of Dilithium and Kyber is the matrix-vector product with polynomial entries. So, the naive approach to generate/verify m signatures with Dilithium (or encrypt m messages with Kyber) where $m > 1$ is to perform m such multiplications. In this thesis, we propose to use efficient matrix multiplications of sizes greater than three to generate/verify m signatures with Dilithium and greater than two to encrypt m messages with Kyber. To this end, Dilithium signature generation, verification, and Kyber encryption algorithms are analyzed. In this study, individual operations for multiple messages or signatures are redesigned to allow for batch processing. For this purpose, the matrix-vector multiplications contained in these algorithms are converted into matrix-matrix multiplications, and the new batch Dilithium signature generation/verification and batch Kyber encryption algorithms are designed. Furthermore, it is shown that the efficient batch Dilithium signature generation algorithm can be used for the generation of a single signature. This is confirmed through probability and efficiency analyses. New batch algorithms have components that ensure the security of the original Dilithium and Kyber and allow batch signing/verification and encryption without causing security vulnerabilities. By transforming matrix-vector multiplications into matrix-matrix multiplications, efficient non-commutative or commutative matrix-matrix multiplication algorithms are allowed to be used in these systems. The efficiency achieved is independent of any specific architecture, device, or platform and is based on mathematical improvements. The designed batch algorithms provide the opportunity to select any batch size and apply the suitable matrix-matrix multiplication technique. In this context, probability computations are made for the usability of the Dilithium signature algorithm in both multiple and single signatures according to various batch sizes, and efficiency percentages are obtained based on the selected matrix-matrix multiplication techniques. Furthermore, improvement percentages provided by batch Kyber encryption and Dilithium verification are computed via matrix-matrix multiplication methods appropriate for changing batch sizes. To exemplify the theoretical calculations made, sample batch values are selected, and suitable matrix-matrix multiplication methods are determined. According to the dimensions of the matrices, multiplication formulas are derived for three security levels of Dilithium and Kyber. To test the calculations, the batch Dilithium signing, verification, and Kyber encryption algorithms designed within this study are implemented in the C programming language. The derived matrix-matrix multiplication formulas are also implemented in C to fit the infrastructure of Kyber and Dilithium and integrated into the batch algorithms. Efficiency comparisons are made between the new algorithms and reference algorithms. As a result, improvements up to 28.1%, 33.3%, and 31.5% in the arithmetic complexities are observed at three different security levels of Dilithium's signature, respectively. The batch Dilithium signature algorithm with an efficient matrix-multiplication method provides 34.22%, 17.40%,

and 10.15% improvements on CPU cycle counts for three security levels. The multiplication formulas used for batch Dilithium signature generation are also applied for batch Dilithium verification. At three different levels of security, improvements in the arithmetic complexity are observed of up to 28.13%, 33.33%, and 31.25%. Furthermore, 49.88%, 56.60%, and 61.08% improvements on CPU cycle counts for three security levels are achieved, respectively. As a result of implementing Kyber Batch Encryption with efficient multiplication algorithms, 12.50%, 22.22%, and 28.13% improvements on arithmetic complexity, as well as 22.34%, 24.07%, and 30.83% improvements on CPU cycle counts, are observed for three security levels.

Keywords: Batch Digital Signature Generation, Post-Quantum Cryptography, Commutative Matrix Multiplication, Crystals Dilithium, Digital Signature, Batch Digital Signature Verification, Crystals Kyber, Batch Encryption

ÖZ

CRYSTALS DILITHIUM İMZA ŞEMASI VE CRYSTALS KYBER ŞİFRELEME ŞEMASI İÇİN VERİMLİ TOPLU KUANTUM ERTESİ ALGORİTMALAR

Türe, Nazlı Deniz
Doktora, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Oğuz Yayla
Ortak Tez Yöneticisi : Prof. Dr. Murat Cenk

Eylül 2024, 119 sayfa

Dijital imzalar kimlik doğrulama ve veri bütünlüğü sağlarlar ve bilgi teknolojileri, finans, eğitim, hukuk gibi birçok alanda yaygın olarak kullanılırlar. Kullanıcı ile sunucu arasındaki iletişimlerde kimlik doğrulama amaçlı kullanılarak sunucuların siber saldırılara karşı güvenliklerinin sağlanması konusunda da oldukça önemlidirler. Diğer bir yandan şifreleme ise veri güvenliğini sağlamak amacıyla güvenli iletişim, bulut ve veritabanı güvenliği gibi birçok alanda kullanılmaktadır. Dijital imzalar ve şifreleme mekanizmalarında sıklıkla açık anahtarlı kriptosistemler kullanılır. Açık anahtarlı kriptosistemlerin güvenliği ise tam sayılarda çarpanlara ayırma ve ayrık logaritma problemi gibi çözümü zor matematiksel problemlere dayanmaktadır. P. Shor (1999), güvenlikleri ayrık logaritma ve tam sayılarda çarpanlara ayırma problemlerine dayanan kriptosistemlerin, gelişmiş kuantum bilgisayarlar kullanılarak kırılabileceğini göstermiştir. Güvenliklerini kaybedecek olan algoritmaların, yeni nesil kuantum güvenli algoritmalar ile değiştirilmesi için NIST (Ulusal Standartlar ve Teknoloji Enstitüsü, ABD) tarafından kuantum sonrası standartlaşma süreci düzenleşmiştir. Bunun sonucunda Crystals Dilithium, Falcon ve SPHINCS+ dijital imza standartları, Crystals Kyber ise şifreleme/anahtar kapsülleme standardı olarak seçilmiştir.

Bu çalışmada, NIST'in (Ulusal Standartlar ve Teknoloji Enstitüsü, ABD) kuantum ertesi kriptografi dijital imza standardı olarak belirlemiş olduğu Crystals Dilithium

algoritması ile toplu imza üretimi ve bir kullanıcıdan gelen imzaların doğrulaması, kuantum ertesi şifreleme standardı olarak belirlenmiş olan Crystals Kyber ile bir kullanıcı için toplu şifreleme üzerine yoğunlaşılmıştır. Dilithium ve Kyber'ın barındırdığı en önemli işlemlerden biri de; girdileri polinom olan matris ve vektörlerin çarpımıdır. $m > 1$ olmak üzere m adet imza klasik Dilithium yöntemi ile üretildiğinde ya da doğrulandığında (m adet mesaj klasik Kyber ile şifrelendiğinde) m adet benzer çarpım gerekmektedir. Bu tezde, Dilithium ile m imzayı üretmek/doğrulamak için üçten büyük matrislerde ve Kyber ile m mesajı şifrelemek için ikiden büyük boyutlu matrislerde verimli matris çarpımlarının kullanılabileceği gösterilmiştir. Bu sebeple, Dilithium imza üretimi, doğrulama ve Kyber şifreleme algoritmaları analiz edilmiştir. Bu çalışmada, birden fazla mesaj veya imza için ayrı ayrı gerçekleştirilen işlemlerin, toplu şekilde yapılabilmesi için yeni algoritmalar tasarlaşmıştır. Bu amaçla, bu algoritmalardaki matris-vektör çarpımları, matris-matris çarpımlarına dönüştürülmüş ve yeni toplu Dilithium imza üretimi/doğrulama ve toplu Kyber şifreleme algoritmaları tasarlanmıştır. Ayrıca, verimli toplu Dilithium imza üretim algoritmasının tek bir imzanın üretilmesinde kullanılabileceği de gösterilmiştir. Bu, olasılık ve verimlilik analizleri ile doğrulanmıştır. Yeni toplu algoritmalar, orijinal Dilithium ve Kyber'in güvenliğini sağlayan bileşenlere sahiptir ve güvenlik açıklarına neden olmadan toplu imzalama/doğrulama ve şifrelemeye olanak sağlar. Matris-vektör çarpımlarını matris-matris çarpımlarına dönüştürerek, bu sistemlerde verimli değişmeli veya değişmeli olmayan matris-matris çarpım algoritmalarının kullanılmasına imkan tanır. Elde edilen verimlilik, belirli bir mimari, cihaz veya platformdan bağımsızdır ve matematiksel iyileştirmelere dayanmaktadır. Tasarlanan toplu algoritmalar, herhangi bir toplu işlem boyutunun seçilmesine ve uygun matris-matris çarpım tekniğinin uygulanmasına olanak sağlar. Bu bağlamda, farklı toplu işlem boyutlarına göre hem toplu hem de tekli imzalar için Dilithium imza algoritmasının kullanımı üzerine olasılık hesaplamaları yapılmış ve seçilen matris-matris çarpım tekniklerine dayalı olarak verimlilik yüzdeleri elde edilmiştir. Ayrıca, değişen toplu işlem boyutları için uygun matris-matris çarpım yöntemleri aracılığıyla toplu Kyber şifrelemesi ve Dilithium doğrulamasının sağladığı iyileştirme yüzdeleri hesaplanmıştır. Teorik hesaplamaları örneklemek amacıyla, örnek toplu işlem değerleri seçilmiş ve uygun matris-matris çarpım yöntemleri belirlenmiştir. Matrislerin boyutlarına göre, Dilithium ve Kyber'ın üç güvenlik seviyesi için çarpım formülleri elde edilmiştir. Hesaplamaları test etmek amacıyla, bu çalışmada tasarlanan toplu Dilithium imzalama, doğrulama ve Kyber şifreleme algoritmaları C programlama dilinde gerçeklenmiştir. Türetilen matris-matris çarpım formülleri de, Kyber ve Dilithium'un altyapısına uyacak şekilde C dilinde gerçeklenmiş ve yeni toplu algoritmalara entegre edilmiştir. Yeni algoritmalar ile referans algoritmalar arasında verimlilik karşılaştırmaları yapılmıştır. Sonuç olarak, Dilithium'un imzasının üç farklı güvenlik seviyesinde aritmetik karmaşıklıkta sırasıyla %28.1, %33.3 ve %31.5 oranında iyileşmeler gözlemlenmiştir. Verimli matris çarpım yöntemi kullanılan toplu Dilithium imza algoritması, üç güvenlik seviyesi için CPU döngü sayılarında sırasıyla %34.22, %17.40 ve %10.15 iyileştirmeler sağlamıştır. Toplu Dilithium imza üretimi için kullanılan çarpım for-

mülleri, aynı zamanda toplu Dilithium doğrulaması için de kullanılmıştır. Üç farklı güvenlik seviyesi için aritmetik karmaşıklıkta sırasıyla %28.13, %33.33 ve %31.25 oranında iyileşmeler gözlemlenmiştir. Ayrıca, üç güvenlik seviyesi için CPU döngü sayıları açısından sırasıyla %49.88, %56.60 ve %61.08 iyileşmeler elde edilmiştir. Verimli çarpım algoritmaları kullanılan toplu Kyber şifrelemesi uygulandığında, aritmetik karmaşıklıkta %12.50, %22.22 ve %28.13 oranında iyileşmeler ile birlikte, üç güvenlik seviyesi için CPU döngü sayılarında %22.34, %24.07 ve %30.83 oranında iyileşmeler gözlemlenmiştir.

Anahtar Kelimeler: Toplu Dijital İmza Üretimi, Toplu İmzalama, Kuantum Ertesi Kriptografi, Değişmeli Matris Çapımı, Crystals Dilithium, Dijital İmza, Toplu İmza Doğrulama, Crystals Kyber, Toplu Şifreleme

To the Women of the Republic...

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my esteemed advisor, Prof. Dr. Murat Cenk, who has been a guiding light throughout this journey. His support, patience, and brilliant ideas made this experience incredibly meaningful. This thesis would not have been possible without his invaluable insights, knowledge, and dedication. He has guided me not only academically but also with his values and exemplary personality, being an inspiring role model.

I am also sincerely grateful to Assoc. Prof. Dr. Oğuz Yayla, whose constructive feedback and support have significantly contributed to the quality and depth of this study. I also want to thank the examining committee members of my thesis, Prof. Dr. Zülfükar Saygı, Assoc. Prof. Dr. Fatih Sulak, Assoc. Prof. Dr. Ahmet Sınak, and Assist. Prof. Dr. Buket Özkaya for their valuable feedback, which greatly improved this thesis.

I would like to extend my sincere appreciation to FAME CRYPT for enabling me to start my Ph.D. journey, for the support during my research, for paving the way for my professional development, and for the opportunities they have provided. I am especially thankful to Assoc. Prof. Dr. Ali Doğanaksoy, Assoc. Prof. Dr. Fatih Sulak, Dr. Muhiddin Uğuz, my friends and colleagues Dr. Melis Aslan, Dr. Sermin Kocaman, and Mustafa Kırcalı for their encouragement and contributions throughout this process, providing support both academically and mentally.

I want to thank the academic personnel and staff at the Institute of Applied Mathematics for all their support and guidance.

I would like to thank The Scientific and Technological Research Council of Turkey (TÜBİTAK) for supporting this work within the scope of the 2244 Industrial Ph.D. Program.

I owe a special debt of gratitude to Sevim Seda Odacıoğlu for her invaluable assistance in the software-related aspects of this thesis. Her support was crucial in navigating the technical challenges encountered during this research. She generously dedicated her valuable time to help me, sharing not only her expertise but also her patience and encouragement throughout the process. This thesis would not have reached its current level without her.

I sincerely appreciate all my teachers for their support and guidance during my educational journey.

I am profoundly thankful to my family for their endless support, the opportunities they have given me, their boundless love, and the care they have shown me throughout my life. None of these achievements would have been possible without their support.

I would also like to express my heartfelt gratitude to my friends Merve Nur Aşkar, Beril Başak Tukaç Geven, Başak Ülgü, Dr. Asya Erol Bayır, Ezgi Eyici, Görkem Genç, Dr. Göktuğ Aydoğan, and Kemal Özdemir who have been a part of my life since high school, supporting me unconditionally and shaping who I am today. Your unwavering support and friendship have meant the world to me.

I am sincerely thankful to my friends Talya Tümer Sivri, Ali Haydar Sivri, Mehmet Esat Memiş, Dr. Oğuzhan Kılıçaslan, Ali Onur Geven, Özge Selçuk, and Dr. Dt. İpek Gizem Bekiroğlu who have stood by me through every moment, offering unyielding support during my toughest times, never withholding their assistance, and sharing in both my happiness and sorrows as we navigated the path to adulthood together.

I am deeply grateful to my dear friend Merve Gözde Sayın, who has been by my side throughout the years, supporting me through all my challenges, making me feel her love, expanding my perspective, and enriching my world. Her steadfast support has played a significant role in shaping the person I am today. Finally, I also wish to express my heartfelt gratitude to my dear friend Yağmur Kılıçaslan, who came into my life perhaps a bit late but undoubtedly at the perfect time, showing me that anything is possible, always made me feel her love and the value she places on me, discovering the hidden room within me that I didn't know existed, and playing a crucial role in bringing out the beauty it holds.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
ARM	Advanced RISC Machines
AVX	Advanced Vector Extensions
CCA	Chosen Ciphertext Attack
CPA	Chosen Plaintext Attack
CPU	Central Processing Unit
DES	Data Encryption Standard
DFT	Discrete Fourier transform
DSS	Digital Signature Standard
ECDSA	Elliptic Curve Digital Signature Algorithm
FFT	Discrete Fourier Transform
GPU	Graphics Processing Unit
KEM	Key Encapsulation Mechanism
LWE	Learning with Errors
M-LWE	Module-Learning with Errors
MMP	Matrix-Matrix Product
MVP	Matrix-Vector Product
NIST	National Institute of Standards and Technology
NTT	Number Theoretic Transform
RSA	Rivest–Shamir–Adleman
SHA	Secure Hash Algorithms
SHAKE	Secure Hash Algorithm and KECCAK
SIMD	Single Instruction, Multiple Data
SSL	Secure Sockets Layer
TLS	Transport Layer Security

CHAPTER 1

INTRODUCTION

Modern cryptography techniques ensure information security. The main concepts of modern cryptography are encryption, authentication, key management, and data integrity.

Encryption is the technique that aims to convert a message into a ciphertext to hide the information. There exist two main categories of encryption: symmetric and asymmetric encryption. In symmetric encryption, the sender and the receiver use the same secret key to encrypt or decrypt a message. AES [25] and DES [20] are examples of well-known symmetric encryption algorithms. Asymmetric encryption uses two different, related keys for encryption and decryption. They are called the public key and the private key. Asymmetric encryption is generally used as a key exchange and digital signature algorithm. Diffie-Hellman [10] key exchange, RSA [19], ElGamal [12], and ECDSA [17] are some asymmetric cryptographic protocols that are commonly used. The security of these algorithms is generally provided by mathematical problems such as the discrete logarithm problem and the integer factorization problem.

Authentication is another feature that cryptographic algorithms provide and ensures that the data comes from a verified source. In modern cryptography, digital signatures (e.g., RSA digital signature protocol, ECDSA, etc.) are widely used for authentication.

Ensuring data integrity is one of the important goals of cryptography. Hash functions, such as SHA-2 [21] and SHA-3 [22], provide data integrity. The hash output changes

completely, even with a small change to the data.

Key management protocols are used to secure key generation, destruction, distribution, and storage. They guarantee that users can exchange and store the keys securely. Diffie-Hellman is one of the best-known key exchange algorithms.

Modern cryptography includes various cryptographic protocols in addition to individual algorithms to aim for secure communication. These protocols, such as TLS, provide data integrity, identity verification, encryption, and authentication.

1.1 Motivation

Digital signatures are essential to provide data integrity, authenticity, and security. Digital signatures are widely used in instant messaging applications, financial transactions, education, legal documents, and many other digital documents. Moreover, they are crucial for servers such as TLS (Transport Layer Security) servers. The servers often face cyber attacks. To prevent unauthorized access, using digital signatures for both the server and user to authenticate each other is essential. RSA and ECDSA are used as a digital signing algorithm on many major servers, such as TLS servers. In [30], Peters and Dolev (2021) introduce an innovative approach to enhance the security of TLS 1.3 in the post-quantum context. They emphasize the critical role of incorporating Dilithium, a lattice-based digital signature scheme, to achieve strong authentication within TLS 1.3. By eliminating the need for handshake signatures, their framework leverages the efficiency and security of Dilithium, resulting in a more resilient and effective post-quantum TLS implementation.

Servers use digital signatures whenever a secure connection between a client and the server is set. So, it can be said that digital signatures are used thousands or even millions of times daily on a busy server. For example, TLS servers can establish multiple connections per second. The processes need to be fast so that the system's flow is not disrupted. For this reason, performing multiple signing operations at once is faster and more advantageous for systems. Benjamin [6] has developed a system based on ECDSA and RSA that provides multiple signings for TLS. Techniques such as ElGamal, ECDSA, Crystals Dilithium [11] and Falcon [15] have implemented

multiple signing in previous studies ([1], [8], [16]). Fiat [14] and Tanwar et al. [36] present a batch algorithm that depends on the RSA system. Pavlovski et al. [27] propose an efficient batch signature generation via binary tree structures.

In addition to batch digital signature generation, batch digital signature verification may also have many uses in daily life, especially for high-traffic messaging systems, SSL/TLS certificate chains, and legal or financial settings. In high-traffic messaging applications, validating multiple messages simultaneously will reduce processing time and improve performance with the help of batch verification strategies. By applying batch verification in SSL/TLS certificate chains, it is possible to validate the entire chain of certificates at once. This improves website load times and accelerates the handshake procedure. Also, in legal or financial settings, there could be multiple documents (e.g., an agreement, a contract, etc.) signed by a single user. The receiver (e.g., a lawyer) could use batch verification to check the signatures easily. [5] introduces an approach to verifying modular exponentiation in cryptography, particularly for digital signatures, through batch verification. A practical method of batch verification of short signatures is described in [13]. [18] includes analyses of batch verification with DSS, ECDSA, and RSA.

Encryption algorithms are another type of cryptosystem used for data security. Encryption is used to protect information from unauthorized users. It is utilized in numerous areas, including banking, cloud data storage, communication, and securing private and sensitive data. RSA and AES are two of the most commonly used encryption schemes available worldwide. Cloud systems, IoT communication, databases, and secure communication systems are systems that have intensive encryption traffic. It is essential to make encryptions in these systems more efficient so that the flow can continue safely and fast. In this sense, it is advantageous to use batch encryption structures in these systems. Fiat [14] has introduced a system where RSA is used in batch encryption. In [9], a fully homomorphic encryption scheme over the integers of van Dijk et al. [37] is extended into its batch version.

One of the forms of attack on modern cryptographic algorithms is quantum attacks. High-capacity quantum computers can be used to break many of the cryptographic algorithms in use today, as demonstrated in 1994 by P. Shor [33]. Since then, a lot

of work has been done, and many developments have been achieved. For this reason, the National Institute of Standards and Technology (NIST) has organized a contest to standardize algorithms that provide security against attacks via quantum computers. As a result of the third round of evaluations [2], CRYSTALS Dilithium [11] is selected as the digital signing standard (NIST FIPS 204-Module-Lattice-Based Digital Signature Standard [23]), while CRYSTALS Kyber [4] is chosen as the encryption/key encapsulation standard (NIST FIPS 203-Module-Lattice-Based Key-Encapsulation Mechanism Standard [24]). Following this process, Dilithium's integrability into TLS 1.3 is demonstrated in [34]. However, in [26], it is shown that using post-quantum in the TLS server will cause performance degradation. On the encryption side, Kyber is replacing the current encryption methods after it has been selected as the encryption/KEM standard. In [31], the Kyber Key Encapsulation Mechanism is integrated into the cloud architecture to improve security. Moreover, [32] suggests an effective method for securing the device-to-device connection with the use of Crystals Kyber for data transmission.

This work focuses on efficient batch post-quantum digital signature generation, batch verification of signatures from the same user using Dilithium, and batch encryption to the same recipient with Kyber. Instead of multiplying a matrix and a column vector, it is possible to multiply the matrix A by another matrix whose columns are column vectors formed for each message or signature. In this way, transforming the matrix-vector multiplication of Dilithium and Kyber into matrix-matrix multiplications for multiple signing, verifying, or encrypting is the main purpose of this work. Therefore, matrix-matrix multiplication algorithms using the least number of multiplications, such as [35], [40], [7], and [29], in multiple signing can increase efficiency. Note that since the entries are large-size polynomials, reducing the number of multiplications contributes significantly to complexity.

Dilithium's and Kyber's structures are examined to increase performance, and the operations with the highest arithmetic complexity are identified. The security of Dilithium and Kyber is based on the Module-Learning with Error (M-LWE) problem. In Dilithium's signing algorithm, if y is a column vector that is computed using the secret key, the column vector w is obtained as $w = A \cdot y$, where A is the matrix that the signer and the verifier can compute. The entries of the column vectors and

the matrix are the elements of the selected commutative polynomial ring. Similarly, Kyber's encryption algorithm includes the $\hat{A} \cdot \hat{r}$ matrix-vector operation, where $\hat{A}$ is the public matrix, and $\hat{r}$ is the polynomial vector generated by using the random coins. The highlight is that in Dilithium and Kyber, matrix-vector multiplication is the operation with the highest arithmetic complexity and forms the skeleton of the algorithms.

1.2 Contributions

This study proposes a design of batch signature generation and verification algorithms for Dilithium's various security levels (i.e., Dilithium 2, Dilithium 3, and Dilithium 5) and batch encryption algorithm for Kyber's different security levels (i.e., Kyber512, Kyber768, and Kyber1024). In this study, the Dilithium signature generation, verification, and Kyber encryption algorithms are thoroughly analyzed and redesigned for batch processing. Instead of performing individual operations for each message or signature, the matrix-vector multiplications inherent to these algorithms are transformed into matrix-matrix multiplications, enabling more efficient batch processing for both Dilithium signature generation/verification and Kyber encryption. Moreover, it is demonstrated that the batch Dilithium signature generation algorithm is applicable for generating a single signature as well. This is validated through efficiency and probability calculations. These newly designed batch algorithms retain the security features of the original Dilithium and Kyber, ensuring that batch processing can be utilized without introducing security vulnerabilities. By converting matrix-vector operations into matrix-matrix operations, efficient non-commutative or commutative multiplication techniques can be incorporated into these systems. To this end, efficient matrix multiplication methods are analyzed, and the most effective algorithms are selected according to the matrix sizes. Strassen's Algorithm, Rosowski's fast commutative matrix multiplication method, and Winograd's method are the prominent ones among these algorithms. We derive the multiplication formulas for Kyber and Dilithium's three security levels. The resulting efficiency improvements are based on mathematical advancements and are independent of any specific architecture, device, or platform.

The flexibility of the designed batch algorithms allows for the selection of different batch sizes, with the appropriate matrix-matrix multiplication techniques applied accordingly. In this context, probability calculations are performed to assess the effectiveness of the Dilithium signature algorithm for both single and multiple signatures across various batch sizes, and efficiency improvements are evaluated based on the chosen multiplication techniques. Additionally, efficiency gains for batch Kyber encryption and Dilithium verification are calculated using matrix-matrix multiplication methods tailored to different batch sizes.

To validate the theoretical calculations, for the specific batch values, corresponding multiplication methods are combined with the batch algorithms. The proposed batch Dilithium signing, verification, and Kyber encryption algorithms are implemented in the C programming language, along with the derived matrix-matrix multiplication formulas, which are integrated into the batch algorithms for both systems. Efficiency comparisons are made between the new batch algorithms and the original reference algorithms.

As a result, it is observed that using [29] for Dilithium 2, Dilithium 3, Dilithium 5, Kyber768, and Kyber1024 is more functional, while [35] and [39] are useful for Kyber512. Since Batch Dilithium 2 and Kyber1024 contain the multiplication of two square matrices of size $(2^d \times 2^d)$, it is possible to apply [7], [35], [40] to it recursively. However, note that multiplication in [29] cannot be used recursively since it requires entries to be commutative, and matrix multiplications are non-commutative. At three distinct security levels of Dilithium's signature, improvements in the arithmetic complexities are observed as 28.1%, 33.3%, and 31.5%, respectively. By implementing the proposed batch Dilithium signature algorithm and the efficient matrix multiplication algorithms, CPU cycle counts for three security levels are improved by 34.22%, 17.40%, and 10.15%, respectively. Batch Dilithium verification uses the same multiplication formulas as batch Dilithium signature generation. Improvements in the arithmetic complexity are observed up to 28.13%, 33.33%, and 31.25% at three distinct security levels. Moreover, improvements in CPU cycle counts for the three security levels are 49.88%, 56.60%, and 61.08%, respectively. Furthermore, improvements in arithmetic complexity of 12.50%, 22.22%, and 28.13%, as well as improvements in CPU cycle counts of 22.34%, 24.07%, and 30.83%, are observed for

three security levels of Kyber when batch encryption is implemented using efficient multiplication algorithms.

1.3 Organization

The organization of the remainder of the paper is as follows: Chapter 2, presents the fundamental concepts and necessary background information. In Chapter 3, efficient matrix multiplication techniques used in this study and complexity analyses of the targeted matrices are explained. Chapter 4, 5, and 6 cover the presentation of the new batch algorithms of Dilithium signature generation, Dilithium signature verification from a single user, Kyber encryption to a single recipient, as well as their arithmetic complexity and implementation analyses. Chapter 7 discusses the platform-independent nature of the batch algorithms and outlines potential future developments. Chapter 8 summarizes this research's accomplishments. Finally, the formulas derived using some efficient matrix-matrix multiplication methods are explained in Appendices A, B, C, D, and E.

CHAPTER 2

PRELIMINARIES

This chapter presents the fundamental concepts and necessary background information to support the upcoming chapters.

2.1 Notations and Subfunctions

The functions and notations that are used in the remaining sections are given in Table 2.1 and Table 2.2.

Table 2.1: Functions and Their Definitions

Function	Definition
NTT	Number Theoretic Transform described in [11]
NTT^{-1}	Inverse operation of Number Theoretic Transform
H	Hash Function (SHAKE-256)
H'	Hash Function (SHA3-256)
G	Hash Function (SHA3-512)
$HighBits$ and $LowBits$	Decomposing operations defined in [11]
$\|z\|_\infty$	$\|z \mod^{\pm} q\|$ defined in [11]
$\|$	Concatenation Operation
$SampleInBall$	256-bit array generator using a seed
CBD	Centered Binomial Distribution
$PRF(s, b)$	Pseudorandom Function (SHAKE-256($s\|b$))
XOF	Extendable Output Function (SHAKE-128)
$Encode$	Function that converts a byte array to a polynomial
$Decode$	Function that converts a polynomial to a byte array
$Parse$	A function to sample elements in R_q defined in [4]
$Compress$	Compression function defined in [4]
$Decompress$	Decompression function defined in [4]

Table 2.2: Notations and Their Definitions

Notation	Definition
$\boldsymbol{A}$	Matrix (Bold Upper Case Letter)
$\boldsymbol{A}[i][j]$	The entry in the i^{th} row and j^{th} column of $\boldsymbol{A}$
$\boldsymbol{v}$	Column vector (Bold Lower Case Letter)
$\boldsymbol{v}[i]$	i^{th} entry of $\boldsymbol{v}$
R	Commutative ring $\mathbb{Z}[X]/(x^n+1)$
R_q	Commutative ring $\mathbb{Z}_q[x]/(x^n+1)$
$R_q^{k\times l}$	$k \times l$ matrices whose entries are from R_q
R_q^k	Column vectors of length k whose entries are from R_q
$\leftarrow$	Uniform sampling
$\mathcal{B}^k$	Set of byte arrays of length k
$\boldsymbol{A}^{k\times l}$	A matrix $\boldsymbol{A}$ that has k rows and l columns

2.2 Algebraic Structures and Transform Methods

This section aims to build a foundational understanding of algebraic structures and transform methods. The mathematical basis for examining the behavior of the proposed algorithms is provided by the following definitions. The algorithms at the focus of this study utilize ring arithmetic.

Definition 1. *A **ring** is a set R together with two binary operations: + (addition) and $\cdot$ (multiplication). It satisfies the following three axioms:*

- *R is an abelian group under addition.*
  - $(x+y)+z = x+(y+z), \quad \forall x,y,z \in R.$
  - $x+y = y+x, \quad \forall x,y \in R.$
  - *For all x in R there is an element 0 such that $x+0=x$.*
  - *For all x in R there exists $-x$ such that $x+(-x)=0$.*
- *Multiplication is associative on R.*
  - $(x\cdot y)\cdot z = x\cdot(y\cdot z), \quad \forall x,y,z \in R.$
  - *For all x in R, there is an element $x \in R$ such that $x\cdot 1 = x$ and $1\cdot x = x$.*

- *Multiplication is distributive with respect to addition.*
  - $x \cdot (y + z) = (x \cdot y) + (x \cdot z), \quad \forall x, y, z \in R.$
  - $(y + z) \cdot x = (y \cdot x) + (z \cdot x), \quad \forall x, y, z \in R.$

Definition 2. *A **commutative ring** R is a ring in which multiplication is a commutative operation.*

- $x \cdot y = y \cdot x, \quad \forall x, y \in R.$

Definition 3. *Assume that I is an ideal of ring R. The set of cosets of I in R, where each coset is of the form $r + I$ for some element $r \in R$, is known as the **quotient ring** and denoted as R/I. Equivalency classes form the elements of the quotient ring, and addition and multiplication are described as follows:*

- $(r + I) + (s + I) = (r + s) + I$
- $(r + I) \cdot (s + I) = (r \cdot s) + I.$

Definition 4. *A **matrix ring** is a set of matrices whose entries are from the ring R that forms a ring itself under matrix addition and matrix multiplication.*

Definition 5. *A matrix with elements that are polynomials over a specific ring or field is called a **polynomial matrix**.*

The algorithms in this study incorporate polynomial matrices in their structure. To efficiently multiply the entries of the polynomial matrices, the Number Theoretic Transform (NTT) is employed, which is a variant of the Fast Fourier Transform (FFT) operating over a finite field or a commutative ring. The FFT is an efficient algorithm for computing the Discrete Fourier Transform (DFT), and it is explained in Algorithm 1. To conduct the necessary complexity and efficiency analysis, it is crucial to determine the computational cost of the NTT algorithm. The arithmetic complexity of the NTT algorithm is outlined in Theorem 1.

Algorithm 1: Fast Fourier Transform (FFT) Algorithm [38]

input : $n = 2^k$ with $k \in \mathbb{N}$, $f = \sum_{0 \le j < n} f_j x^j \in R[X]$, and the powers $\omega, \omega^2, \ldots, \omega^{n-1}$ of a primitive n-th root of unity $\omega \in R$

output : $\mathrm{DFT}_\omega(f) = (f(1), f(\omega), \ldots, f(\omega^{n-1})) \in R^n$

if $n = 1$ **then**

  return f_0

$r_0 \leftarrow \sum_{0 \le j < n/2} (f_j + f_{j+n/2}) x^j, \quad r_1^* \leftarrow \sum_{0 \le j < n/2} (f_j - f_{j+n/2}) \omega^j x^j$

call the algorithm recursively to evaluate r_0 **and** r_1^* **at the powers of** ω^2

return $\big(r_0(1), r_1^*(1), r_0(\omega^2), r_1^*(\omega^2), \ldots, r_0(\omega^{n-2}), r_1^*(\omega^{n-2})\big)$

Theorem 1. *Let n be a power of 2 and $\omega \in R$ be a primitive n-th root of unity. Then, DFT$_\omega$ can be computed with Algorithm 1, using $n \log n$ additions in ring R and $\frac{n}{2} \log n$ multiplications by powers of ω, in a total of $\frac{3}{2} n \log n$ ring operations ([38]).*

Proof. The correctness is established by using induction on k. If $k = 0$, then $f = f_0$ is constant and the algorithm yields the correct output. If $k > 1$, it is necessary to show that $f(\omega^{2\ell}) = r_0(\omega^{2\ell})$ and $f(\omega^{2\ell+1}) = r_1^*(\omega^{2\ell})$ for all $0 \le \ell < n/2$.

By retracing the calculations and using the induction hypothesis, we find

$$\begin{aligned} r_0(\omega^{2\ell}) &= \sum_{0 \le j < n/2} (f_j + f_{j+n/2}) \omega^{2\ell j} = \sum_{0 \le j < n/2} f_j \omega^{2\ell j} + \sum_{0 \le j < n/2} f_{j+n/2} \omega^{2\ell j} \omega^{\ell n} \\ &= \sum_{0 \le j < n} f_j \omega^{2\ell j} = f(\omega^{2\ell}), \end{aligned}$$

and

$$\begin{aligned} r_1^*(\omega^{2\ell}) &= \sum_{0 \le j < n/2} (f_j - f_{j+n/2}) \omega^j \omega^{2\ell j} \\ &= \sum_{0 \le j < n/2} f_j \omega^{(2\ell+1) j} + \sum_{0 \le j < n/2} f_{j+n/2} \omega^{(2\ell+1) j} \omega^{\ell n} \omega^{n/2} \\ &= \sum_{0 \le j < n} f_j \omega^{(2\ell+1) j} = f(\omega^{2\ell+1}) \end{aligned}$$

Let $A(n)$ and $M(n)$ represent the number of additions and multiplications in ring R, respectively, are required by the algorithm for an input of size n. The computational

cost for each step is as follows: 0 in steps 1, 2, and 4, n additions and $n/2$ multiplications in step 3, and $2A(n/2)$ additions and $2M(n/2)$ multiplications in step 4. This yields

$$A(n) = n \log n \quad \text{and} \quad M(n) = \frac{1}{2} n \log n.$$

□

The concepts introduced in this section form the structural foundation of the algorithms that are central to this thesis. The next section provides essential background on post-quantum cryptography and the post-quantum algorithms that contain these mathematical concepts.

2.3 Post-Quantum Cryptography

Modern cryptography is divided into categories such as public-key encryption, symmetric encryption, digital signing, key exchange, and key encapsulation. One of the most widely used cryptosystems is public-key cryptography. These algorithms include ElGamal, ECDSA, Diffie-Hellman Key Exchange, and RSA. The security of public-key cryptosystems is based on difficult mathematical problems such as the integer factorization problem and the discrete logarithm problem. It is very difficult to solve these problems with classical computers if the correct variables are selected.

Definition 6. *For a given number M,* ***Integer Factorization Problem*** *is the problem of determining p_i and x_i that satisfy $M = \prod p_i^{x_i}$.*

Definition 7. ***Discrete Logarithm Problem*** *is the problem of finding x that satisfies $a^x \equiv b \mod n$, using the values of a, b, and n.*

Quantum computers are devices that use concepts from quantum mechanics, such as superposition and entanglement. It has been demonstrated by P. Shor [33] that cryptosystems whose security depends on solving the discrete logarithm problem and the integer factorization problem can be broken using high-capacity quantum computers.

In order to replace cryptographic algorithms that will lose their security with the next-generation quantum-secure algorithms, NIST (National Institute of Standards and Technology, USA) has organized the post-quantum standardization process. At the end of this process, Crystals Kyber [4] is selected as the encryption/KEM standard, while Crystals Dilithium [11], Falcon [15], and SPHINCS+ [3] are selected as the digital signature standards.

2.4 Crystals Dilithium

In the NIST standardization process, the digital signature algorithm Crystals Dilithium, which is based on Module-Learning with Errors, has been selected as a post-quantum signing standard. Dilithium's algorithms for key generation, signing, and verification are given in Algorithm 2, Algorithm 3, and Algorithm 4, respectively.

The mechanism for generating keys in Dilithium is described in Algorithm 2. Using the 256-bit long ζ value, ρ, ρ', and K are formed in the first stage. The public polynomial matrix $\hat{\boldsymbol{A}}$ is generated through the parameter ρ, while the hidden polynomial vectors $\boldsymbol{s_1}$ and $\boldsymbol{s_2}$ are produced by the same value. Step 5 involves calculating $\boldsymbol{t}$ using the generated polynomial vectors and matrix. In the end, the hash and sub-functions are used to create the key pair.

Algorithm 3 provides a description of the Dilithium signature generation algorithm. The ρ generated during the signature process is used to obtain the public polynomial matrix $\hat{\boldsymbol{A}}$. With NTT's help, the message is signed using the private key and $\hat{\boldsymbol{A}}$. The correctness of the signature phase is checked. If the requirements fail to be fulfilled, this process must be repeated. The probability that the signature occurs correctly is explained in detail in Dilithium's official document [11], and the probability that the entire signature will be correct is computed as $e^{-256 \cdot \beta \cdot k/\gamma_2}$.

Using the public key, the signature's validity is verified in Algorithm 4.

Table 2.3 displays the parameters of Dilithium, where (k, l) represents the dimensions of the public matrix $\hat{\boldsymbol{A}}^{k \times l}$. $\hat{\boldsymbol{A}}$ is a matrix whose entries are polynomials from the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$.

Algorithm 2: Dilithium Key Generation [11]

output : Public key pk, Secret key sk

$\zeta \leftarrow \{0,1\}^{256}$

$(\rho, \rho', K) \in \{0,1\}^{256} \times \{0,1\}^{512} \times \{0,1\}^{256} := H(\zeta)$

$\hat{\boldsymbol{A}} \in R_q^{k \times l} := ExpandA(\rho)$ ▷ $\boldsymbol{A}$ is generated in NTT representation as $\hat{\boldsymbol{A}}$

$(\boldsymbol{s_1}, \boldsymbol{s_2}) \in S_\eta^l \times S_\eta^k := ExpandS(\rho')$

$\boldsymbol{t} := NTT^{-1}(\hat{\boldsymbol{A}} \cdot NTT(\boldsymbol{s_1})) + \boldsymbol{s_2}$

$(\boldsymbol{t_1}, \boldsymbol{t_0}) := Power2Round_q(\boldsymbol{t}, d)$

$tr \in \{0,1\}^{256} := H(\rho \| \boldsymbol{t_1})$

return $pk = (\rho, \boldsymbol{t_1})$, $sk = (\rho, K, tr, \boldsymbol{s_1}, \boldsymbol{s_2}, \boldsymbol{t_0})$

Algorithm 3: Dilithium Signing [11]

input : Secret key $sk = (\rho, K, tr, \boldsymbol{s_1}, \boldsymbol{s_2}, \boldsymbol{t_0})$, message M

output : Signature σ

$\hat{\boldsymbol{A}} \in R_q^{k \times l} := ExpandA(\rho)$ ▷ $\boldsymbol{A}$ is generated in NTT representation as $\hat{\boldsymbol{A}}$

$\mu \in \{0,1\}^{512} := H(tr \| M)$

$\kappa := 0$, $(\boldsymbol{z}, \boldsymbol{h}) = \perp$

$\rho' \in \{0,1\}^{512} := H(K \| \mu)$

$\hat{\boldsymbol{s}}_1 := NTT(\boldsymbol{s}_1)$, $\hat{\boldsymbol{s}}_2 := NTT(\boldsymbol{s}_2)$, $\hat{\boldsymbol{t}}_0 := NTT(\boldsymbol{t}_0)$

while $(\boldsymbol{z}, \boldsymbol{h}) = \perp$ **do**

  $\boldsymbol{y} \in \tilde{S}_{\gamma_1}^l := ExpandMask(\rho', \kappa)$

  $\boldsymbol{w} := NTT^{-1}(\hat{\boldsymbol{A}} \cdot NTT(\boldsymbol{y}))$

  $\boldsymbol{w_1} := HighBits_q(\boldsymbol{w}, 2\gamma_w)$

  $\tilde{c} \in \{0,1\}^{256} := H(\mu \| \boldsymbol{w_1})$

  $c \in B_\tau := SampleInBall(\tilde{c})$

  $\boldsymbol{z} := \boldsymbol{y} + NTT^{-1}(\hat{c}\hat{\boldsymbol{s}}_1)$ ▷ $\hat{\boldsymbol{c}} = NTT(\boldsymbol{c})$

  $\boldsymbol{r_0} := LowBits_q(\boldsymbol{w} - NTT^{-1}(\hat{c} \cdot \hat{\boldsymbol{s}}_2), 2\gamma_2)$

  if $\|\boldsymbol{z}\|_\infty \geq \gamma_1 - \beta$ ***or*** $\|\boldsymbol{r_0}\|_\infty \geq \gamma_2 - \beta$ **then**

    $(\boldsymbol{z}, \boldsymbol{h}) := \perp$

  else

    $\boldsymbol{h} := MakeHint_q(-NTT^{-1}(\hat{c} \cdot \hat{\boldsymbol{t}}_0), \boldsymbol{w} - c\boldsymbol{s_2} + NTT^{-1}(\hat{c} \cdot \hat{\boldsymbol{t}}_0), 2\gamma_2)$

    if $\|c\boldsymbol{t}_0\|_\infty \geq \gamma_2$ ***or*** *the # of 1's in* $\boldsymbol{h}$ *is greater than* ω **then**

      $(\boldsymbol{z}, \boldsymbol{h}) := \perp$

  $\kappa := \kappa + l$

return $\sigma = (\tilde{c}, \boldsymbol{z}, \boldsymbol{h})$

Algorithm 4: Dilithium Verification [11]

input : Public key $pk = (\rho, \boldsymbol{t_1})$, message M, signature σ

output : Valid or Not

1 $\hat{\boldsymbol{A}} \in R_q^{k \times l} := ExpandA(\rho)$ ▷ $\boldsymbol{A}$ is generated in NTT representation as $\hat{\boldsymbol{A}}$

2 $\mu \in \{0,1\}^{512} := H(H(\rho \| \boldsymbol{t_1}) \| M)$

3 $c := SampleInBall(\tilde{c})$

4 $\boldsymbol{w_1'} := UseHint_q(\boldsymbol{h}, NTT^{-1}(\hat{\boldsymbol{A}} \cdot NTT(\boldsymbol{z}) - NTT(c) \cdot NTT(\boldsymbol{t_1} \cdot 2^d)))$

5 **return** [$\| \boldsymbol{z} \|_\infty < \gamma_1 - \beta$] **and** [$\tilde{c} = H(\mu \| \boldsymbol{w_1'})$] **and** [# of 1's in $\boldsymbol{h}$ is $\leq \omega$]

Table 2.3: Parameter Sets for Dilithium [11]

Security Level	**Algorithm**	n	$(k \times l)$	q
2	Dilithium 2	256	(4×4)	8380417
3	Dilithium 3	256	(6×5)	8380417
5	Dilithium 5	256	(8×7)	8380417

Since all of the elements in the matrices and column vectors are polynomials from the commutative ring $\mathbb{Z}_q[X]/(x^{256}+1)$, where q is in Table 2.3, the matrix-vector multiplications in the 5^{th} step of the Algorithm 2, the 8^{th} step of the Algorithm 3, and the 4^{th} step of the Algorithm 4 are one of the most expensive operations of Dilithium. Consider an example where $\boldsymbol{A}$ belongs to $R_q^{6\times5}$ and $\boldsymbol{s'}$ belongs to R_q^5 to illustrate this operation. Thirty polynomial multiplications are required to obtain $\boldsymbol{A} \cdot \boldsymbol{s'}$ where all of the polynomials come from $\mathbb{Z}_q[X]/(x^{256}+1)$. In order to sign six distinct messages for six distinct users, 180 polynomial multiplications would be needed. Reducing the amount of multiplications is an efficient way to sign multiple messages. In Chapters 4 and 5, batch Dilithium signature generation and batch Dilithium verification from a single user are proposed, respectively.

2.5 Crystals Kyber

Crystals Kyber is an M-LWE (Module-Learning with Errors) based encryption/key encapsulation algorithm selected as a post-quantum standard in the NIST's standardization process. It includes CPAPKE key generation, CCAKEM key generation, CPAPKE encryption, CCAKEM encapsulation, CPAPKE decryption, and CCAKEM decapsulation algorithms. CPAPKE encryption and CCAKEM encapsulation phases

are given in Algorithm 5 and Algorithm 6, respectively.

Kyber CPAPKE encryption details are given in Algorithm 5. The algorithm creates the ciphertext c after receiving as inputs the random coin r, the message m, and the public key pk. The nonce N is fixed to zero at the beginning of the process. Then, polynomial vector $\hat{\boldsymbol{t}}$ and ρ value are extracted by using the public key pk. Using ρ value, $Parse$, and XOF, the public matrix $\hat{\boldsymbol{A}}^T$ is generated in NTT domain. Then, using random coin r and nonce N; $\boldsymbol{r}$, $\boldsymbol{e}_1$, and e_2 are sampled with the help of the CBD and PRF. $\boldsymbol{u}$ and v are calculated using the public matrix and the polynomial vectors produced in the previous steps. Finally, $Encode$ and $Compress$ functions are used to obtain the ciphertext c.

Algorithm 5: Kyber CPAPKE Encryption [4]

input : Public key pk, message m, random coin r

output : Ciphertext c

$N := 0$

$\hat{\boldsymbol{t}} := Decode_{12}(pk)$

$\rho := pk + 12 \cdot k \cdot n/8$

for $i = 0, 1, \ldots, k-1$ **do**

  for $j = 0, 1, \ldots, k-1$ **do**

    $\hat{\boldsymbol{A}}^T[i][j] := Parse(XOF(\rho, i, j))$ ▷ $\boldsymbol{A}$ is generated in NTT representation as $\hat{\boldsymbol{A}}$

for $i = 0, 1, \ldots, k-1$ **do**

  $\boldsymbol{r}[i] := CBD_{\eta_1}(PRF(r, N))$

  $N := N + 1$

for $i = 0, 1, \ldots, k-1$ **do**

  $\boldsymbol{e_1}[i] := CBD_{\eta_2}(PRF(r, N))$

  $N := N + 1$

$e_2 := CBD_{\eta_2}(PRF(r, N))$

$\hat{\boldsymbol{r}} := NTT(\boldsymbol{r})$

$\boldsymbol{u} := NTT^{-1}(\hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{r}}) + \boldsymbol{e_1}$

$v := NTT^{-1}(\hat{\boldsymbol{t}}^T \cdot \hat{\boldsymbol{r}}) + e_2 + Decompress_q(Decode_1(m), 1)$

$c_1 := Encode_{d_u}(Compress_q(\boldsymbol{u}, d_u))$

$c_2 := Encode_{d_v}(Compress_q(v, d_v))$

return $c = (c_1 \| c_2)$

Table 2.4: Parameter Sets for Kyber [4]

Security Level	**Algorithm**	n	k	q	η_1	η_2	(d_u, d_v)
2	Kyber512	256	2	3329	3	2	(10,4)
3	Kyber768	256	3	3329	2	2	(10,4)
5	Kyber1024	256	4	3329	2	2	(11,5)

Algorithm 6 provides the details of the Kyber CCAKEM encapsulation mechanism. The algorithm takes pk as an input and generates the ciphertext c and the secret shared key K. It starts with generating the byte array m of length 32. Then, it is updated using the hash function H'. pk and m are sent as input to the H' and G hash functions, yielding $\bar{K}$ and r as a result. Next, pk is given as the public key, m as the message, and r as a random coin to Algorithm 6, and thus, the ciphertext c is obtained. The secret shared key K is determined through KDF, H' hash functions, and $\bar{K}$, c inputs.

The parameters of Kyber are shown in Table 2.4, where k is the size of the public matrix $\hat{\boldsymbol{A}}^{k \times k}$. The entries of the matrix $\hat{\boldsymbol{A}}$ are the polynomials from the ring $R'_q = \mathbb{Z}_q[x]/(x^n + 1)$. η_1, η_2, and (d_u, d_v) are the parameters chosen to ensure the balance between ciphertext size and security.

Algorithm 6: Kyber CCAKEM Encapsulation [4]

input : Public key pk

output : Ciphertext c, Shared Key K

1 $m \leftarrow \mathcal{B}^{32}$

2 $m \leftarrow H'(m)$

3 $(\bar{K}, r) := G(m \| H'(pk))$

4 $c := KYBER.CPAPKE.Enc(pk, m, r)$

5 $K := KDF(\bar{K} \| H'(c))$

6 **return** $c = (c, K)$

The matrix-vector multiplication in the 15^{th} step of the Algorithm 5 is one of the most costly operations in Kyber Encryption. Similar to the Dilithium algorithm, the entries of the matrices and the vectors are polynomials. Although the matrix and vector sizes are small, many polynomial multiplications are required to encrypt messages. Reducing the number of multiplications required to encrypt multiple messages can be very beneficial in terms of performance. The system that provides efficient batch encryption to a single recipient using the Kyber algorithm is described in Chapter 6.

CHAPTER 3

EFFICIENT MATRIX MULTIPLICATION METHODS AND COMPLEXITY ANALYSES OF TARGETED MATRICES

In this chapter, efficient matrix multiplication methods are explored, and complexity analyses of targeted matrix multiplications are conducted.

3.1 Efficient Matrix Multiplication Methods

There exist various matrix multiplication methods that aim to multiply two matrices more efficiently than the naive method. These algorithms vary based on different matrix sizes and properties. For instance, some methods are applicable only to square matrices, while others are designed for all matrix sizes. Depending on their structure and characteristics, some efficient matrix multiplication algorithms can also be used recursively. Additionally, some algorithms can be applied commutatively, while others are non-commutative.

The primary goal of efficient matrix algorithms is to reduce the total number of multiplication operations. However, an increase in the number of addition/subtraction operations may be observed. These methods are preferred because the cost of multiplication operations is higher than that of addition/subtraction operations.

In this study, various efficient matrix-matrix multiplication methods have been utilized within the batch versions of Dilithium and Kyber. For the selected sample batch numbers, Strassen's algorithm has been adapted for the batch versions of Kyber512, Kyber1024, and Dilithium 2. This is to demonstrate the applicability of Strassen-

like matrix multiplication techniques to our new method. For Kyber768, Dilithium 2, Dilithium 3, and Dilithium 5, Rosowski's fast commutative matrix multiplication algorithm has been employed. This method is an example of commutative matrix multiplication techniques and significantly reduces the number of multiplications. Additionally, improvements expected from using Winograd's inner product algorithm for various batch values of Kyber512 have also been calculated. These improvements are detailed in Chapters 4, 5, and 6.

3.1.1 Strassens's Algorithm

Let $\boldsymbol{M}$ and $\boldsymbol{N}$ be square matrices and let $\boldsymbol{L} = \boldsymbol{M} \cdot \boldsymbol{N}$. The matrices $\boldsymbol{M}$ and $\boldsymbol{N}$ are divided into block matrices of equal size as follows:

$$\underbrace{\begin{bmatrix} \boldsymbol{M}_{11} & \boldsymbol{M}_{12} \\ \boldsymbol{M}_{21} & \boldsymbol{M}_{22} \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} \boldsymbol{N}_{11} & \boldsymbol{N}_{12} \\ \boldsymbol{N}_{21} & \boldsymbol{N}_{22} \end{bmatrix}}_{N} = \underbrace{\begin{bmatrix} \boldsymbol{L}_{11} & \boldsymbol{L}_{12} \\ \boldsymbol{L}_{21} & \boldsymbol{L}_{22} \end{bmatrix}}_{L}$$

The following pre-computations are done to obtain the matrix $\boldsymbol{L}$:

$$\boldsymbol{P}_1 = (\boldsymbol{M}_{11} + \boldsymbol{M}_{22}) \cdot (\boldsymbol{N}_{11} + \boldsymbol{N}_{22})$$
$$\boldsymbol{P}_2 = (\boldsymbol{M}_{21} + \boldsymbol{M}_{22}) \cdot \boldsymbol{N}_{11}$$
$$\boldsymbol{P}_3 = \boldsymbol{M}_{11} \cdot (\boldsymbol{N}_{12} - \boldsymbol{N}_{22})$$
$$\boldsymbol{P}_4 = \boldsymbol{M}_{22} \cdot (\boldsymbol{N}_{21} - \boldsymbol{N}_{11})$$
$$\boldsymbol{P}_5 = (\boldsymbol{M}_{11} + \boldsymbol{M}_{12}) \cdot \boldsymbol{N}_{22}$$
$$\boldsymbol{P}_6 = (\boldsymbol{M}_{21} - \boldsymbol{M}_{11}) \cdot (\boldsymbol{N}_{11} + \boldsymbol{N}_{12})$$
$$\boldsymbol{P}_7 = (\boldsymbol{M}_{12} - \boldsymbol{M}_{22}) \cdot (\boldsymbol{N}_{21} + \boldsymbol{N}_{22})$$

Then, the equal-sized block sub-matrices of $\boldsymbol{L}$ are computed as:

$$\boldsymbol{L}_{11} = \boldsymbol{P}_1 + \boldsymbol{P}_4 - \boldsymbol{P}_5 + \boldsymbol{P}_7 \qquad \boldsymbol{L}_{12} = \boldsymbol{P}_3 + \boldsymbol{P}_5$$
$$\boldsymbol{L}_{21} = \boldsymbol{P}_2 + \boldsymbol{P}_4 \qquad \boldsymbol{L}_{22} = \boldsymbol{P}_1 - \boldsymbol{P}_2 + \boldsymbol{P}_3 + \boldsymbol{P}_6$$

Strassen's method [35] requires 7 multiplications and 18 additions instead of 8 multiplications and 4 additions to obtain $\boldsymbol{M}^{2\times 2} \cdot \boldsymbol{N}^{2\times 2}$. If $\boldsymbol{M}^{2^m \times 2^m}$ and $\boldsymbol{N}^{2^m \times 2^m}$ are square matrices and $n = 2^m$, then Strassen's algorithm can be used recursively and has a total arithmetic complexity of $7n^{2.81} - 6n^2$.

3.1.2 Winograd's Variant

Let $\boldsymbol{M}$ and $\boldsymbol{N}$ be square matrices and let $\boldsymbol{L} = \boldsymbol{M} \cdot \boldsymbol{N}$. The matrices $\boldsymbol{M}$ and $\boldsymbol{N}$ are divided into block matrices of equal size as follows:

$$\underbrace{\begin{bmatrix} \boldsymbol{M}_{11} & \boldsymbol{M}_{12} \\ \boldsymbol{M}_{21} & \boldsymbol{M}_{22} \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} \boldsymbol{N}_{11} & \boldsymbol{N}_{12} \\ \boldsymbol{N}_{21} & \boldsymbol{N}_{22} \end{bmatrix}}_{N} = \underbrace{\begin{bmatrix} \boldsymbol{L}_{11} & \boldsymbol{L}_{12} \\ \boldsymbol{L}_{21} & \boldsymbol{L}_{22} \end{bmatrix}}_{L}$$

The following pre-computations are done to obtain the matrix $\boldsymbol{L}$:

$$\begin{aligned}
\boldsymbol{P}_1 &= \boldsymbol{M}_{11}\boldsymbol{N}_{11} \\
\boldsymbol{P}_2 &= \boldsymbol{M}_{12}\boldsymbol{N}_{21} \\
\boldsymbol{P}_3 &= \boldsymbol{M}_{22}(\boldsymbol{N}_{11} - \boldsymbol{N}_{12} - \boldsymbol{N}_{21} + \boldsymbol{N}_{22}) \\
\boldsymbol{P}_4 &= (\boldsymbol{M}_{11} - \boldsymbol{M}_{21})(-\boldsymbol{N}_{12} + \boldsymbol{N}_{22}) \\
\boldsymbol{P}_5 &= (\boldsymbol{M}_{21} + \boldsymbol{M}_{22})(-\boldsymbol{N}_{11} + \boldsymbol{N}_{12}) \\
\boldsymbol{P}_6 &= (\boldsymbol{M}_{11} + \boldsymbol{M}_{12} - \boldsymbol{M}_{21} - \boldsymbol{M}_{22})\boldsymbol{N}_{22} \\
\boldsymbol{P}_7 &= (\boldsymbol{M}_{11} - \boldsymbol{M}_{21} - \boldsymbol{M}_{22})(\boldsymbol{N}_{11} - \boldsymbol{N}_{12} + \boldsymbol{N}_{22})
\end{aligned}$$

Then, the equal-sized block sub-matrices of $\boldsymbol{L}$ are computed as:

$$\begin{aligned}
\boldsymbol{L}_{11} &= \boldsymbol{P}_1 + \boldsymbol{P}_2 & \boldsymbol{L}_{12} &= \boldsymbol{P}_1 + \boldsymbol{P}_5 + \boldsymbol{P}_6 - \boldsymbol{P}_7 \\
\boldsymbol{L}_{21} &= \boldsymbol{P}_1 - \boldsymbol{P}_3 + \boldsymbol{P}_4 - \boldsymbol{P}_7 & \boldsymbol{L}_{22} &= \boldsymbol{P}_1 + \boldsymbol{P}_4 + \boldsymbol{P}_5 - \boldsymbol{P}_7
\end{aligned}$$

Wonograds's variant requires 7 multiplications and 15 additions -which is optimal ([40], [28]) for Strassen-like algorithms- instead of 8 multiplications and 4 additions

to obtain $\boldsymbol{M}^{2\times2} \cdot \boldsymbol{N}^{2\times2}$. If $\boldsymbol{M}^{2^m\times2^m}$ and $\boldsymbol{N}^{2^m\times2^m}$ are square matrices, and $n = 2^m$ then the algorithm can be used recursively and has a total arithmetic complexity of $6n^{2.81} - 5n^2$.

3.1.3 Optimized Block Decomposition Method for Winograd's Variant Algorithm

Let $\boldsymbol{M}$ and $\boldsymbol{N}$ be two matrices of size $2^k \times 2^k$ and let $\boldsymbol{L} = \boldsymbol{M} \cdot \boldsymbol{N}$. The matrix $\boldsymbol{L}$ can be obtained with the complexity $5n^{\log_2 7} + 0.5n^{\log_2 6} + 2n^{\log_2 5} - 6.5n^2$ using the improved block decomposition method on Winograd's Variant algorithm described in [7]. There are three main steps to obtain the matrix $\boldsymbol{L}$: Component Matrix Formation (CMF), Component Multiplication (CM), and Reconstruction (R).

Component Matrix Formation (CMF):

To compute $CMF(\boldsymbol{M})$, the following must be obtained knowing that $CMF(\boldsymbol{M} + \boldsymbol{N}) = CMF(\boldsymbol{M}) \oplus CMF(\boldsymbol{N})$:

$$\boldsymbol{P}_1 = \boldsymbol{M}_{11} - \boldsymbol{M}_{21}$$
$$\boldsymbol{P}_2 = \boldsymbol{M}_{21} + \boldsymbol{M}_{22}$$

Note that $CMF(\boldsymbol{M}) = \boldsymbol{M}_{11}$ for $2^k = 1$. For $2^k \geq 2$, $CMF(\boldsymbol{M})$ is obtained as follows:

$$\begin{aligned} CMF(\boldsymbol{M}) =& (CMF(\boldsymbol{M}_{11}), CMF(\boldsymbol{M}_{12}), CMF(\boldsymbol{M}_{22}), CMF(\boldsymbol{P}_1), CMF(\boldsymbol{P}_2), \\ & K_1 \oplus CMF(\boldsymbol{M}_{12}), \underbrace{CMF(\boldsymbol{P}_1) \ominus CMF(\boldsymbol{M}_{22})}_{K_1}). \end{aligned}$$

After that, to compute $CMF(\boldsymbol{N})$, it is needed to calculate $\boldsymbol{P}_3$, $\boldsymbol{P}_4$, with the equations below:

$$\boldsymbol{P}_3 = -\boldsymbol{N}_{12} + \boldsymbol{N}_{22}$$
$$\boldsymbol{P}_4 = -\boldsymbol{N}_{11} + \boldsymbol{N}_{12}$$

Note that $CMF(\boldsymbol{N}) = \boldsymbol{N}_{11}$ for $2^k = 1$. For $2^k \geq 2$, $CMF(\boldsymbol{N})$ is computed as below:

$$\begin{aligned} CMF(\boldsymbol{N}) =& (CMF(\boldsymbol{N}_{11}), CMF(\boldsymbol{N}_{21}), K_2 \ominus CMF(\boldsymbol{N}_{21}), CMF(\boldsymbol{P}_3), \\ & CMF(\boldsymbol{P}_4), \underbrace{CMF(\boldsymbol{N}_{22} \ominus CMF(\boldsymbol{P}_4))}_{K_2}). \end{aligned}$$

Component Multiplication (CM):

$CMF(\boldsymbol{M})$ and $CMF(\boldsymbol{N})$ are multiplied component by component to obtain

$$CM(\boldsymbol{L}) = (CM_1, CM_2, CM_3, CM_4, CM_5, CM_6, CM_7).$$

Reconstruction (R):

To obtain $\boldsymbol{L} = \boldsymbol{M} \cdot \boldsymbol{N}$, the following must be computed knowing that $R(\boldsymbol{M} \oplus \boldsymbol{N}) = R(\boldsymbol{M}) \oplus R(\boldsymbol{N})$:

$$\begin{aligned} \boldsymbol{R}_1 &= CM_1 \oplus CM_2 \\ \boldsymbol{R}_2 &= R(\boldsymbol{R}_1) \\ \boldsymbol{R}_3 &= CM_1 \ominus CM_7 \\ \boldsymbol{R}_4 &= R(\boldsymbol{R}_3) \\ \boldsymbol{R}_5 &= R(CM_5) \\ \boldsymbol{R}_6 &= \boldsymbol{R}_4 \oplus \boldsymbol{R}_5 \\ \boldsymbol{R}_7 &= R(CM_6) \end{aligned} \qquad \begin{aligned} \boldsymbol{R}_8 &= \boldsymbol{R}_6 \oplus \boldsymbol{R}_7 \\ \boldsymbol{R}_9 &= R(CM_3) \\ \boldsymbol{R}_{10} &= R(CM_4) \\ \boldsymbol{R}_{11} &= -\boldsymbol{R}_9 \oplus \boldsymbol{R}_{10} \\ \boldsymbol{R}_{12} &= \boldsymbol{R}_4 \oplus \boldsymbol{R}_{11} \\ \boldsymbol{R}_{13} &= \boldsymbol{R}_6 \oplus \boldsymbol{R}_{10} \end{aligned}$$

Finally, $R(CM(\boldsymbol{L})) = [\boldsymbol{R}_2, \boldsymbol{R}_8, \boldsymbol{R}_{12}, \boldsymbol{R}_{13}]$ for $2^k \geq 2$ and $R(CM(\boldsymbol{L})) = CM_1$ for $2^k = 1$. Then:

$$\begin{aligned} \boldsymbol{L}_{11} &= \boldsymbol{R}_2 \\ \boldsymbol{L}_{21} &= \boldsymbol{R}_{12} \end{aligned} \qquad \begin{aligned} \boldsymbol{L}_{12} &= \boldsymbol{R}_8 \\ \boldsymbol{L}_{22} &= \boldsymbol{R}_{13} \end{aligned}$$

3.1.4 Rosowski's Fast Commutative Matrix Multiplication Algorithm

Rosowski's fast commutative matrix multiplication algorithm [29] is designed to efficiently multiply two matrices $\boldsymbol{M}^{x\times y}$ and $\boldsymbol{N}^{y\times z}$ whose entries are from a commutative ring. Let $\boldsymbol{M}^{x\times y} \cdot \boldsymbol{N}^{y\times z} = \boldsymbol{L}^{x\times z}$, and m_{ij}, n_{ij}, l_{ij} denotes the entries of $\boldsymbol{M}$, $\boldsymbol{N}$, $\boldsymbol{L}$, respectively. The entries of the matrix $\boldsymbol{L}$ are calculated in various ways according to different conditions.

Case 1: Let $x \geq 2$ be even and let y, $z \geq 1$. Then, the entries of the matrix $\boldsymbol{L}$ are computed with

$$y(xz + x + z - 1)/2 \tag{3.1}$$

multiplications as follows:

- $l_{i1} = \sum_{k=1}^{y/2} m_{i(2k-1)}(n_{(2k-1)1} + m_{i(2k)}) + \sum_{k=1}^{y/2} m_{i(2k)}(n_{(2k)1} - m_{i(2k-1)})$ for $i = 1, \ldots, x$.
- $l_{ij} = \sum_{k=1}^{y/2}(m_{i(2k-1)} + n_{(2k)j})(m_{i(2k)} + n_{(2k-1)1} + n_{(2k-1)j})$ $- \sum_{k=1}^{y/2} m_{i(2k-1)}(n_{(2k-1)1} + m_{i(2k)}) - \sum_{k=1}^{y/2} n_{(2k)j}(n_{(2k-1)1} + n_{(2k-1)j})$ for $i = 1, \ldots, x$ and $j = 2, \ldots, z$.

Case 2: Let $x \geq 3$ be odd and let $y \geq 1$, $z \geq 3$. Then, the entries of the matrix $\boldsymbol{L}$ are computed with

$$y(xz + x + z - 1)/2 \tag{3.2}$$

multiplications if z is odd and

$$(y(xz + x + z - 1) + x - 1)/2 \tag{3.3}$$

multiplications if z is even as follows:

- First, $\boldsymbol{M}$ and $\boldsymbol{N}$ divided into two sub-matrices as $\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M}_1^{x\times 3} & \boldsymbol{M}_2^{x\times(y-3)} \end{bmatrix}$ and $\boldsymbol{N} = \begin{bmatrix} \boldsymbol{N}_1^{3\times x} \\ \boldsymbol{N}_2^{(y-3)\times z} \end{bmatrix}$.

- Then, matrix $\boldsymbol{L}$ can be computed as $\boldsymbol{L} = \boldsymbol{L}_1 + \boldsymbol{L}_2 = \boldsymbol{M}_1\boldsymbol{N}_1 + \boldsymbol{M}_2\boldsymbol{N}_2$.
- $\boldsymbol{M}_1\boldsymbol{N}_1$ and $\boldsymbol{M}_2\boldsymbol{N}_2$ are obtained individually by using the formulas explained below.

Case 2.1: Let l_{ij} be the entries of $\boldsymbol{L}_1$ or $\boldsymbol{L}_2$. If y is odd, then the following formulas are used to obtain $\boldsymbol{L}_1$ or $\boldsymbol{L}_2$:

For $i = 1, \ldots, x$:

- $l_{i1} = (m_{i1} + n_{21})(m_{i2} + n_{12}) + (m_{i1} + n_{31})(m_{i3} + n_{13}) + m_{i1}(n_{11} - n_{12} - n_{13} - m_{i2} - m_{i3}) - n_{12}n_{21} - n_{13}n_{31}$
- $l_{i2} = (m_{i1} + n_{21})(m_{i2} + n_{12}) + (m_{i2} + n_{32})(m_{i3} + n_{23}) + m_{i2}(n_{22} - n_{21} - n_{23} - m_{i1} - m_{i3}) - n_{12}n_{21} - n_{23}n_{32}$
- $l_{i3} = (m_{i1} + n_{31})(m_{i3} + n_{13}) + (m_{i2} + n_{32})(m_{i3} + n_{23}) + m_{i3}(n_{33} - n_{31} - n_{32} - m_{i1} - m_{i2}) - n_{13}n_{31} - n_{23}n_{32}$

For $i = 1, \ldots, x$ and $j = 4, 6, 8, \ldots, y - 1$:

- $l_{ij} = (m_{i1}+n_{21})(m_{i2}+n_{12})+(m_{i1}+n_{31})(m_{i3}+n_{13})+(m_{i1}+n_{21}-n_{2j})(-m_{i2}-n_{12}+n_{1j}-n_{1(j+1)})+(m_{i1}+n_{31}-n_{3j})(-m_{i3}-n_{13}+n_{1(j+1)})-n_{12}n_{21}-n_{13}n_{31}-(n_{21}-n_{2j})(-n_{12}+n_{1j}-n_{1(j+1)})-(n_{31}-n_{3j})(-n_{13}+n_{1(j+1)})$
- $l_{i(j+1)} = (m_{i1} + n_{31})(m_{i3} + n_{13}) + (m_{i2} + n_{32})(m_{i3} + n_{23}) + (m_{i1} + n_{31} - n_{3j})(-m_{i3}-n_{13}+n_{1(j+1)})+(m_{i2}+n_{32}-n_{3j}-n_{3(j+1)})(-m_{i3}-n_{23}+n_{2(j+1)})-n_{13}n_{31}-n_{23}n_{32}-(n_{31}-n_{3j})(-n_{13}+n_{1(j+1)})-(n_{32}-n_{3j}-n_{3(j+1)})(-n_{23}+n_{2(j+1)})$

Case 2.2: Let l_{ij} be the entries of $\boldsymbol{L}_1$ or $\boldsymbol{L}_2$. If y is even, then the following formulas are used to obtain $\boldsymbol{L}_1$ or $\boldsymbol{L}_2$:

For $i = 1, \ldots, x$:

- $l_{i1} = (m_{i1} + n_{21})(m_{i2} + n_{12}) + (m_{i1} + n_{31})(m_{i3} + n_{13}) + m_{i1}(n_{11} - n_{12} - n_{13} - m_{i2} - m_{i3}) - n_{12}n_{21} - n_{13}n_{31}$

- $l_{i2} = (m_{i1} + n_{21})(m_{i2} + n_{12}) + (m_{i2} + n_{32})(m_{i3} + n_{23}) + m_{i2}(n_{22} - n_{21} - n_{23} - m_{i1} - m_{i3}) - n_{12}n_{21} - n_{23}n_{32}$
- $l_{i3} = (m_{i1} + n_{31})(m_{i3} + n_{13}) + (m_{i2} + n_{32})(m_{i3} + n_{23}) + m_{i3}(n_{33} - n_{31} - n_{32} - m_{i1} - m_{i2}) - n_{13}n_{31} - n_{23}n_{32}$
- $l_{i4} = (m_{i1}+n_{21})(m_{i2}+n_{12})+(m_{i1}+n_{21}-n_{24})(-m_{i2}-n_{12}+n_{14})+m_{i3}n_{34}-n_{12}n_{21}-(n_{21}-n_{24})(-n_{12}+n_{14})$

For $i = 1, \ldots, x$ and $j = 5, 7, 9, \ldots, y-1$:

- $l_{ij} = (m_{i1}+n_{21})(m_{i2}+n_{12})+(m_{i1}+n_{31})(m_{i3}+n_{13})+(m_{i1}+n_{21}-n_{2j})(-m_{i2}-n_{12}+n_{1j}-n_{1(j+1)})+(m_{i1}+n_{31}-n_{3j})(-m_{i3}-n_{13}+n_{1(j+1)})-n_{12}n_{21}-n_{13}n_{31}-(n_{21}-n_{2j})(-n_{12}+n_{1j}-n_{1(j+1)})-(n_{31}-n_{3j})(-n_{13}+n_{1(j+1)})$
- $l_{i(j+1)} = (m_{i1} + n_{31})(m_{i3} + n_{13}) + (m_{i2} + n_{32})(m_{i3} + n_{23}) + (m_{i1} + n_{31} - n_{3j})(-m_{i3}-n_{13}+n_{1(j+1)})+(m_{i2}+n_{32}-n_{3j}-n_{3(j+1)})(-m_{i3}-n_{23}+n_{2(j+1)})-n_{13}n_{31}-n_{23}n_{32}-(n_{31}-n_{3j})(-n_{13}+n_{1(j+1)})-(n_{32}-n_{3j}-n_{3(j+1)})(-n_{23}+n_{2(j+1)})$

3.1.5 Winograd's Algorithm for Inner Product

The inner product of two vectors can be computed more efficiently by applying Winograd's method [39]. Let $\boldsymbol{a} = [a_1, a_2, \ldots, a_n]$ and $\boldsymbol{b} = [b_1, b_2, \ldots, a_n]$ be vectors. To obtain their inner product with Winograd's method, the following steps are applied:

- $\alpha = \sum_{i=1}^{\lfloor n/2 \rfloor} a_{2j-1}a_{2j}$ is computed.
- $\beta = \sum_{i=1}^{\lfloor n/2 \rfloor} b_{2j-1}b_{2j}$ is computed.
- If n is even, then the inner product (a, b) is calculated as:

$$(a, b) = \sum_{i=1}^{\lfloor n/2 \rfloor} (a_{2j-1} + b_{2j})(a_{2j} + b_{2j-1}) - \alpha - \beta$$

- If n is odd, then the inner product (a, b) is calculated as:

$$(a, b) = \sum_{i=1}^{\lfloor n/2 \rfloor} (a_{2j-1} + b_{2j})(a_{2j} + b_{2j-1}) - \alpha - \beta + a_n b_n$$

Winograd's algorithm can be used to multiply two matrices efficiently. Let $\boldsymbol{M}^{x\times y}$ and $\boldsymbol{N}^{y\times z}$ be two matrices. Computing $\boldsymbol{L} = \boldsymbol{M} \cdot \boldsymbol{N}$ is equivalent to computing xz inner products. Calculating the matrix $\boldsymbol{L}$ requires

$$(x+z)y + (xz - x - z)\lfloor (y+1)/2 \rfloor \tag{3.4}$$

multiplications instead of xyz. Moreover, the required number of additions equals

$$(x+z)(\lfloor y/2 \rfloor - 1) + xz(y + \lfloor y/2 \rfloor + 1).$$

3.2 Targeted Matrix Multiplications and Their Complexity Analyses

In this thesis, several matrix-matrix multiplications with specific dimensions are utilized to demonstrate the efficiency and applicability of various matrix multiplication algorithms with the new methods that this study explains. Let (n_1, n_2, n_3) matrix multiplication be the product of a matrix of dimension $(n_1 \times n_2)$ and $(n_2 \times n_3)$. In the following chapters, we need efficient $(2, 2, 2)$, $(3, 3, 4)$, $(4, 4, 4)$, $(6, 5, 5)$, $(8, 7, 4)$ matrix multiplications. For the $(4, 4, 4)$ and $(2, 2, 2)$ multiplications, the best choice seems to be Strassen-like recursive multiplications [35], [40], [7] with 49 and 7 multiplications, respectively. Moreover, $(4, 4, 4)$ can be calculated with the commutative matrix multiplication method by [29] with 46 multiplications. For $(6, 5, 5)$ matrix multiplication can be performed with 100 multiplications, $(8, 7, 4)$ can be obtained with 154, and $(3, 3, 4)$ can be computed with 28 multiplications using [29]. The required number of multiplications for these matrix dimensions and the corresponding improvements compared to the standard schoolbook method are summarized in Table 3.1. We have analyzed efficient matrix multiplication methods and decided which are more beneficial for the targeted matrix multiplications. Then, we derive all explicit formulas for the required matrix multiplications and use them in our implementation. The multiplication formulas for these specific matrix-matrix multiplications are provided in Appendices A, B, C, D, and E for detailed reference. Furthermore, a detailed total complexity analysis of these matrix multiplications is provided in Section 3.2.1 to offer a comprehensive understanding of their computational cost.

Table 3.1: Comparison of Required Multiplications for Targeted Matrix Dimensions Using Various Multiplication Methods and Their Improvements Over the Standard Schoolbook Algorithm

Matrices	**Schoolbook**	**Strassen-like ([7], [35], [40])**		**Rosowski's ([29])**	
	#of Mult.	#of Mult.	Imprv. (%)	#of Mult.	Imprv. (%)
$(2, 2, 2)$	8	7	12.50	-	-
$(3, 3, 4)$	36	-	-	28	22.22
$(4, 4, 4)$	64	49	23.44	46	28.13
$(6, 5, 5)$	150	-	-	100	33.33
$(8, 7, 4)$	224	-	-	154	31.25

3.2.1 Arithmetic Complexity Analyses

Efficient matrix multiplication algorithms provide better multiplicative complexity than the naive approach. However, they contain more addition and subtraction operations. The costs of multiplication and addition operations on integers are nearly equal when performed on modern computers. However, if the entries of the matrices and vectors are not integers but polynomials from the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, multiplication is more expensive than the addition/subtraction operations. Although efficient matrix multiplication lowers the number of multiplicative operations, polynomial multiplication in R_q is still quite expensive; hence, using NTT is crucial for efficiency.

Let $a(x), b(x), c(x) \in R_q = \mathbb{Z}_q[x]/(x^n + 1)$ and $a(x) \cdot b(x) = c(x)$. Then, $c(x)$ is obtained using NTT by Theorem 1 as follows:

- $a(x)$ is transformed into its NTT form as $NTT(a(x))$, and it requires $\frac{n}{2} \log n$ multiplications and $n \log n$ additions in $\mathbb{Z}_q$.
- $b(x)$ is transformed into its NTT form as $NTT(b(x))$, and it requires $\frac{n}{2} \log n$ multiplications and $n \log n$ additions in $\mathbb{Z}_q$.
- $NTT(c)$ is computed as $NTT(a(x)) \cdot NTT(b(x))$, and it requires n multiplications in $\mathbb{Z}_q$.
- Finally, $c(x)$ is obtained as $NTT^{-1}(NTT(c(x)))$, and it requires $\frac{n}{2} \log n$ multiplications and $n \log n$ additions in $\mathbb{Z}_q$.

Therefore, multiplying two polynomials requires $\frac{3}{2}n\log n + n$ multiplications and $3n\log n$ additions in $\mathbb{Z}_q$ using NTT. Let $a(x), b(x), c(x) \in R_q = \mathbb{Z}_q[x]/(x^n+1)$ and $a(x) + b(x) = d(x)$. Then, it requires $n-1$ additions in $\mathbb{Z}_q$ to compute $d(x)$. Let M and A indicate operations on polynomials (multiplication and addition), with m and a as their respective coefficient-wise forms in $\mathbb{Z}_q$. Then, $M = (3n\log n)a + (\frac{3}{2}n\log n + n)m$ by Theorem 1 and $A = (n-1)a$.

3.2.1.1 $(4,4,p)$ Matrix Multiplication

Classical Method for $(4,4,p)$:

Let, $\hat{\boldsymbol{A}}^{l\times n}\cdot\hat{\boldsymbol{y}}^{n\times 1} = \hat{\boldsymbol{A}}^{4\times 4}\cdot\hat{\boldsymbol{y}}^{4\times 1}$ polynomial matrix-vector multiplication occurs $p \geq 2$ times. Then, the required number of polynomial multiplications is $16p$, and number of polynomial additions is $12p$. Then, the total number of operations can be computed as:

$$\begin{aligned} Total &= (16p)M + (12p)A \\ &= (16p)((3n\log n)a + (\frac{3}{2}n\log n + n)m) + (12p)((n-1)a) \end{aligned}$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$\begin{aligned} Total &= (16p)(\frac{9}{2}n\log n + n) + (12p)(n-1) \\ &= 72pn\log n + 28pn - 12p \end{aligned} \tag{3.5}$$

Let $n = 256$, then Equation 3.5 is equal to $154612p$. When $p = 4$, the total number of operations for $(4,4,4)$ is 618448.

Efficient Method for $(4,4,p)$:

Let $p \geq 2$. Then, $\hat{\boldsymbol{A}}^{l\times n}\cdot\hat{\boldsymbol{Y}}^{n\times p} = \hat{\boldsymbol{A}}^{4\times 4}\cdot\hat{\boldsymbol{Y}}^{4\times p}$ matrix operation requires $n(lp + l + p - 1)/2 = 4(4p + 4 + p - 1)/2 = 10p + 6$ polynomial multiplications and $46p - 18$ polynomial additions if it is done by using Rosowski's method that is explained in Section 3.1.4.

The total number of operations can be computed as:

$$\begin{aligned} Total &= (10p+6)M + (46p-18)A \\ &= (10p+6)((3n\log n)a + (\frac{3}{2}n\log n + n)m) + (46p-18)((n-1)a) \end{aligned}$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$\begin{aligned} Total &= (10p+6)(\frac{9}{2}n\log n + n) + (46p-18)(n-1) \\ &= 45pn\log n + 27n\log n + 56pn - 12n - 46p + 18 \end{aligned} \tag{3.6}$$

Let $n = 256$, then Equation 3.6 is equal to $106450p + 52242$. When $p = 4$, the total number of operations for $(4, 4, 4)$ is 478042. Rosowski's method is more efficient than the classical method for $(4, 4, p)$. These results can be observed from Table 3.2 and Figure 3.1. In addition to Rosowski's method, $(4, 4, 4)$ can be obtained with Strassen's Algorithm with:

$$\begin{aligned} Total &= 49M + 318A \\ &= 49((3n\log n)a + (\frac{3}{2}n\log n + n)m) + 318((n-1)a) \end{aligned}$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$\begin{aligned} Total &= 49(\frac{9}{2}n\log n + n) + 318(n-1) \\ &= \frac{441}{2}n\log n + 367n - 318 \end{aligned}$$

Since $n = 256$, it requires 545218 arithmetic operations, which is also better than the classical algorithm. Note that Strassen's algorithm can be used to compute $(4, 4, 4^k)$.

Table 3.2: Variation of the Total Number of Operations Required According to p for the Efficient and Classical Method for $(4, 4, p)$, and the Improvement Rates Provided by the Efficient Algorithm

p	**Classical Operations**	**Efficient Operations**	**Improvement (%)**
2	309224	265142	14.26
3	463836	371592	19.89
4	618448	478042	22.70
5	773060	584492	24.39
6	927672	690942	25.52
7	1082284	797392	26.32
8	1236896	903842	26.93
9	1391508	1010292	27.40
10	1546120	1116742	27.77

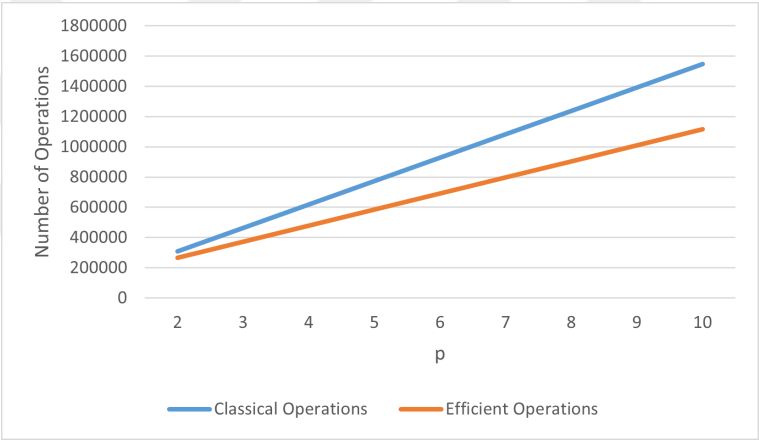

Figure 3.1: Variation of the Total Number of Operations Required According to p for the Efficient and Classical $(4, 4, p)$ Multiplication

3.2.1.2 $(6, 5, p)$ Matrix Multiplication

Classical Method for $(6, 5, p)$:

Let $\hat{\boldsymbol{A}}^{n\times l} \cdot \hat{\boldsymbol{y}}^{l\times 1} = \hat{\boldsymbol{A}}^{6\times 5} \cdot \hat{\boldsymbol{y}}^{5\times 1}$ polynomial matrix-vector multiplication occurs $p \geq 3$ times. Then, the required number of polynomial multiplications is $30p$, and number of polynomial additions is $24p$. Then, the total number of operations can be computed as:

$$
\begin{aligned}
Total &= (30p)M + (24p)A \\
&= (30p)((3n\log n)a + (\frac{3}{2}n\log n + n)m) + (24p)((n-1)a)
\end{aligned}
$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$
\begin{aligned}
Total &= (30p)(\frac{9}{2}n\log n + n) + (24p)(n-1) \\
&= 135pn\log n + 54pn - 24p
\end{aligned} \tag{3.7}
$$

Let $n = 256$, then Equation 3.7 is equal to $290280p$. When $p = 5$, the total number of operations for $(6, 5, 5)$ is 1451400.

Efficient Method for $(6, 5, p)$ When p is Odd:

Let $p \geq 3$ odd. Then, $\hat{\boldsymbol{A}}^{l\times n} \cdot \hat{\boldsymbol{Y}}^{n\times p} = \hat{\boldsymbol{A}}^{6\times 5} \cdot \hat{\boldsymbol{Y}}^{5\times p}$ matrix operation requires at most $n(lp + l + p - 1)/2 = (35p + 25)/2$ polynomial multiplications and $(271p - 257)/2$ polynomial additions if it is done by using Rosowski's method. Then, the total number of operations can be calculated as:

$$
\begin{aligned}
Total &= \frac{35p+25}{2}M + \frac{271p-257}{2}A \\
&= \frac{35p+25}{2}((3n\log n)a + (\frac{3}{2}n\log n + n)m) + \frac{271p-257}{2}((n-1)a)
\end{aligned}
$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$
\begin{aligned}
Total &= \frac{35p+25}{2}(\frac{9}{2}n\log n + n) + \frac{271p-257}{2}(n-1) \\
&= \frac{315pn\log n + 225n\log n}{4} + \frac{257-271p}{2} - 116n + 153np
\end{aligned} \tag{3.8}
$$

Let $n = 256$, then Equation 3.8 is equal to $\frac{400625p+171265}{2}$. When $p = 5$, the total number of operations for $(6, 5, 5)$ is 1087195. Rosowski's method is more efficient

than the classical method for $(6, 5, p)$ when p is odd. These can be examined from Table 3.3 and Figure 3.2.

Efficient Method for $(6, 5, p)$ When p is Even:

Let $p \geq 3$ even. Then, $\hat{\boldsymbol{A}}^{l\times n} \cdot \hat{\boldsymbol{Y}}^{n\times p} = \hat{\boldsymbol{A}}^{6\times 5} \cdot \hat{\boldsymbol{Y}}^{5\times p}$ matrix operation requires at most $(n(lp + l + p - 1) + l - 1)/2 = (35p + 30)/2$ polynomial multiplications and $(271p-340)/2$ polynomial additions if it is done by using Rosowski's method. Then, the total number of operations can be computed as:

$$\begin{aligned} Total &= \frac{35p+30}{2}M + \frac{271p-340}{2}A \\ &= \frac{35p+30}{2}((3n\log n)a + (\frac{3}{2}n\log n + n)m) + \frac{271p-340}{2}((n-1)a) \end{aligned}$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$\begin{aligned} Total &= \frac{35p+30}{2}(\frac{9}{2}n\log n + n) + \frac{271p-340}{2}(n-1) \\ &= \frac{315pn\log n}{4} + \frac{135n\log n - 271p}{2} - 155n + 153np + 170 \end{aligned} \tag{3.9}$$

Let $n = 256$, then Equation 3.9 is equal to $\frac{400625p+197460}{2}$. Rosowski's method is more efficient than the classical method for $(6, 5, p)$ when p is even. This can be examined from Table 3.3 and Figure 3.2.

Table 3.3: Variation of the Total Number of Operations Required According to p for the Efficient and Classical Method for $(6, 5, p)$, and the Improvement Rates Provided by the Efficient Algorithm

p	**Classical Operations**	**Efficient Operations**	**Improvement (%)**
3	870840	686570	21.16
4	1161120	899980	22.49
5	1451400	1087195	25.09
6	1741680	1300605	25.32
7	2031960	1487820	26.78
8	2322240	1701230	26.74
9	2612520	1888445	27.72
10	2902800	2101855	27.59

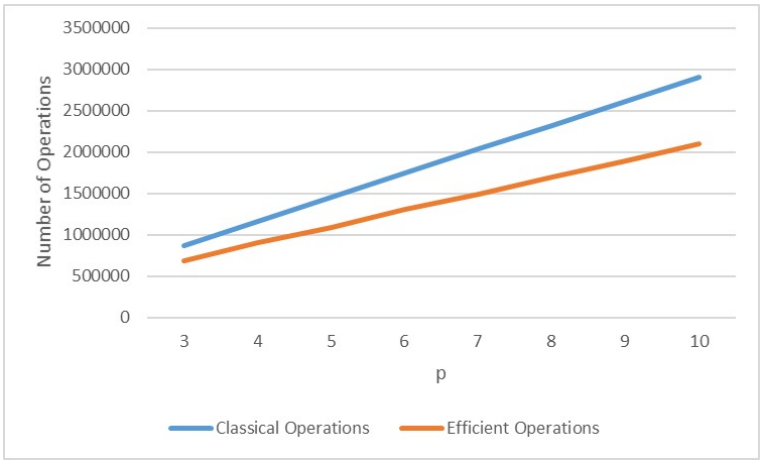

Figure 3.2: Variation of the Total Number of Operations Required According to p for the Efficient and Classical $(6, 5, p)$ Multiplication

3.2.1.3 $(8, 7, p)$ Matrix Multiplication

Classical Method for $(8, 7, p)$:

Let $\hat{\boldsymbol{A}}^{n\times l} \cdot \hat{\boldsymbol{y}}^{l\times 1} = \hat{\boldsymbol{A}}^{8\times 7} \cdot \hat{\boldsymbol{y}}^{7\times 1}$ polynomial matrix-vector multiplication occurs $p \geq 3$ times. Then, the required number of polynomial multiplications is $56p$, and number of polynomial additions is $48p$. Then, the total number of operations can be computed as:

$$\begin{aligned} Total &= (56p)M + (48p)A \\ &= (56p)((3n\log n)a + (\frac{3}{2}n\log n + n)m) + (48p)((n-1)a) \end{aligned}$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$\begin{aligned} Total &= (56p)(\frac{9}{2}n\log n + n) + (48p)(n-1) \\ &= 252pn\log n + 104pn - 48p \end{aligned} \tag{3.10}$$

Let $n = 256$, then Equation 3.10 is equal to $542672p$. When $p = 4$, the total number of operations for $(8, 7, 4)$ is 2170688.

Efficient Method for $(8,7,p)$ When p is Odd:

Let $p \geq 3$ odd. Then, $\hat{\boldsymbol{A}}^{l\times n} \cdot \hat{\boldsymbol{Y}}^{n\times p} = \hat{\boldsymbol{A}}^{8\times 7} \cdot \hat{\boldsymbol{Y}}^{7\times p}$ matrix operation requires at most $n(lp+l+p-1)/2 = (63p+49)/2$ polynomial multiplications and $(455p-365)/2$ polynomial additions if it is done by using Rosowski's method. Then, the total number of operations can be computed as:

$$\begin{aligned} Total &= \frac{63p+49}{2}M + \frac{455p-365}{2}A \\ &= \frac{63p+49}{2}((3n\log n)a + (\frac{3}{2}n\log n + n)m) + \frac{455p-365}{2}((n-1)a) \end{aligned}$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$\begin{aligned} Total &= \frac{63p+49}{2}(\frac{9}{2}n\log n + n) + \frac{455p-365}{2}(n-1) \\ &= \frac{567np\log n + 441n\log n}{4} + \frac{365-455p}{2} - 158n + 259np \end{aligned} \tag{3.11}$$

Let $n = 256$, then Equation 3.11 is equal to $\frac{712761p+371053}{2}$. Rosowski's method is more efficient than the classical method for $(8,7,p)$ when p is odd. These results can be observed in Table 3.4 and Figure 3.3.

Efficient Method for $(8,7,p)$ When p is Even:

Let $p \geq 3$ even. Then, $\hat{\boldsymbol{A}}^{l\times n} \cdot \hat{\boldsymbol{Y}}^{n\times p} = \hat{\boldsymbol{A}}^{8\times 7} \cdot \hat{\boldsymbol{Y}}^{7\times p}$ matrix operation requires at most $(n(lp+l+p-1)+l-1)/2 = (63p+56)/2$ polynomial multiplications and $(455p-474)/2$ polynomial additions if it is done by using Rosowski's method. Then, the total number of operations can be computed as:

$$\begin{aligned} Total &= \frac{63p+56}{2}M + \frac{455p-474}{2}A \\ &= \frac{63p+56}{2}((3n\log n)a + (\frac{3}{2}n\log n + n)m) + \frac{455p-474}{2}((n-1)a) \end{aligned}$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$\begin{aligned} Total &= \frac{63p+56}{2}(\frac{9}{2}n\log n + n) + \frac{455p-474}{2}(n-1) \\ &= \frac{567np\log n}{4} - \frac{455p}{2} - 209n + 259np + 126n\log n + 237 \end{aligned} \tag{3.12}$$

Let $n = 256$, then Equation 3.12 is equal to $\frac{712761p+409562}{2}$. When $p = 4$, the total number of operations for $(8, 7, 4)$ is 1630303. Rosowski's method is more efficient than the classical method for $(8, 7, p)$ when p is even. These results can be checked from Table 3.4 and Figure 3.3.

Table 3.4: Variation of the Total Number of Operations Required According to p for the Efficient and Classical Method for $(8, 7, p)$, and the Improvement Rates Provided by the Efficient Algorithm

p	**Classical Operations**	**Efficient Operations**	**Improvement (%)**
3	1628016	1254668	22.93
4	2170688	1630303	24.89
5	2713360	1967429	27.49
6	3256032	2343064	28.04
7	3798704	2680190	29.44
8	4341376	3055825	29.61
9	4884048	3392951	30.53
10	5426720	3768586	30.55

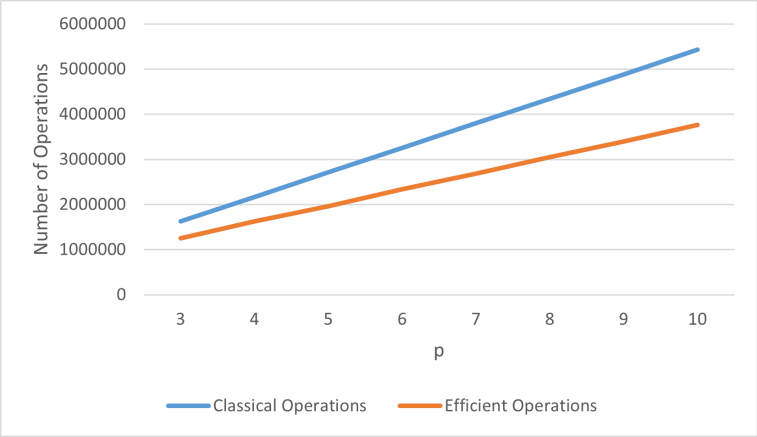

Figure 3.3: Variation of the Total Number of Operations Required According to p for the Efficient and Classical $(8, 7, p)$ Multiplication

3.2.1.4 $(2, 2, p)$ Matrix Multiplication

Classical Method for $(2, 2, p)$:

Let $\hat{\boldsymbol{A}}^{n\times l} \cdot \hat{\boldsymbol{r}}^{l\times 1} = \hat{\boldsymbol{A}}^{2\times 2} \cdot \hat{\boldsymbol{r}}^{2\times 1}$ polynomial matrix-vector multiplication occurs $p \geq 2$ times. Then, the required number of polynomial multiplications is $4p$, and number of polynomial additions is $2p$. Then, the total number of operations can be computed as:

$$\begin{aligned} Total &= (4p)M + (2p)A \\ &= (4p)((3n\log n)a + (\frac{3}{2}n\log n + n)m) + (2p)((n-1)a) \end{aligned}$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$\begin{aligned} Total &= (4p)(\frac{9}{2}n\log n + n) + (2p)(n-1) \\ &= 18pn\log n + 6pn - 2p \end{aligned} \tag{3.13}$$

Let $n = 256$, then Equation 3.13 is equal to $38398p$. When $p = 2$, the total number of operations for $(2, 2, 2)$ is 76796.

Efficient Method for $(2, 2, p \geq 3)$:

Let $p \geq 3$. Then, $\hat{\boldsymbol{A}}^{l\times n} \cdot \hat{\boldsymbol{R}}^{n\times p} = \hat{\boldsymbol{A}}^{2\times 2} \cdot \hat{\boldsymbol{R}}^{2\times p}$ matrix operation requires $(l+p)n + (lp-l-p)\lfloor (n+1)/2 \rfloor = 3p+2$ polynomial multiplications and $(l+p)(\lfloor n/2 \rfloor - 1) + lp(n + \lfloor n/2 \rfloor + 1) = 8p$ polynomial additions if it is done by using Winograd's Inner Product [39] method explained in Section 3.1.5. Then, the total number of operations can be computed as:

$$\begin{aligned} Total &= (3p+2)M + (8p)A \\ &= (3p+2)((3n\log n)a + (\frac{3}{2}n\log n + n)m) + (8p)((n-1)a) \end{aligned}$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$\begin{aligned} Total &= (3p+2)(\frac{9}{2}n\log n + n) + (8p)(n-1) \\ &= \frac{27pn\log n}{2} + 11pn + 9n\log n + 2n - 8p \end{aligned} \tag{3.14}$$

Let $n = 256$, then Equation 3.14 is equal to $30456p + 18944$. Therefore, Winograd's Inner Product method is more efficient than the classical method for $(2, 2, p \geq 3)$, which can be observed in Table 3.5 and Figure 3.4. Note that Strassen's algorithm can be used to compute $(2, 2, 2^k)$.

Efficient Method for $(2, 2, 2)$: If $p = 2$, Strassen's method can be used, and 7 polynomial multiplications and 18 polynomial additions are needed. The total number of operations can be obtained as follows:

$$\begin{aligned} Total &= 7M + 18A \\ &= 7((3n\log n)a + (\frac{3}{2}n\log n + n)m) + 18((n-1)a) \end{aligned}$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$\begin{aligned} Total &= 7(\frac{9}{2}n\log n + n) + 18(n-1) \\ &= \frac{63n\log n}{2} + 25n - 18 \end{aligned} \tag{3.15}$$

The classical method requires 76796 operations when $n = 256$. Since the batch approach needs 70894 operations according to Equation 3.15, it is more advantageous than the classical method. These results can be observed from Table 3.5 and Figure 3.4.

Table 3.5: Variation of the Total Number of Operations Required According to p for the Efficient and Classical Method for $(2, 2, p)$, and the Improvement Rates Provided by the Efficient Algorithm

p	**Classical Operations**	**Efficient Operations**	**Improvement (%)**
2	76796	70894	7.69
3	115194	110312	4.24
4	153592	140768	8.35
5	191990	171224	10.82
6	230388	201680	12.46
7	268786	232136	13.64
8	307184	262592	14.52
9	345582	293048	15.20
10	383980	323504	15.75

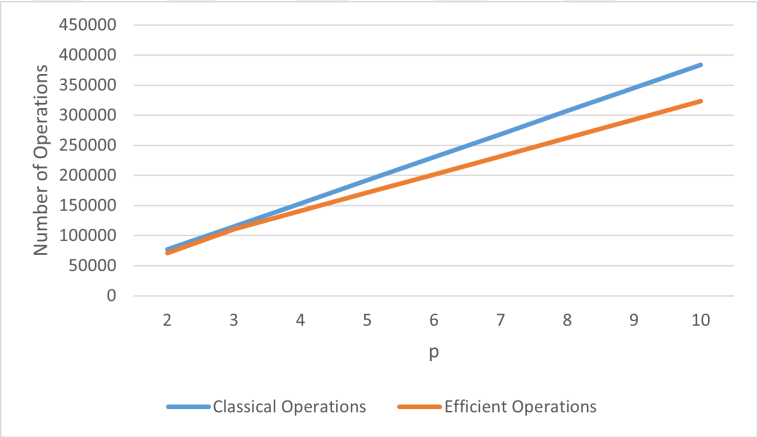

Figure 3.4: Variation of the Total Number of Operations Required According to p for the Efficient and Classical $(2, 2, p)$ Multiplication

3.2.1.5 $(3, 3, p)$ Matrix Multiplication

Classical Method for $(3, 3, p)$:

Let $\hat{\boldsymbol{A}}^{n\times l} \cdot \hat{\boldsymbol{r}}^{l\times 1} = \hat{\boldsymbol{A}}^{3\times 3} \cdot \hat{\boldsymbol{r}}^{3\times 1}$ polynomial matrix-vector multiplication occurs $p \geq 3$ times. Then, the required number of polynomial multiplications is $9p$, and number of polynomial additions is $6p$. Then, the total number of operations can be computed as:

$$
\begin{aligned}
Total &= (9p)M + (6p)A \\
&= (9p)((3n\log n)a + (\frac{3}{2}n\log n + n)m) + (6p)((n-1)a)
\end{aligned}
$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$
\begin{aligned}
Total &= (9p)(\frac{9}{2}n\log n + n) + (6p)(n-1) \\
&= \frac{81pn\log n}{2} + 15pn - 6p
\end{aligned} \tag{3.16}
$$

Let $n = 256$, then Equation 3.16 is equal to $86778p$. When $p = 4$, the total number of operations for $(3, 3, 4)$ is 347112.

Efficient Method for $(3, 3, p)$ When p is Odd:

Let $p \geq 3$ odd. Then, $\hat{\boldsymbol{A}}^{l\times n} \cdot \hat{\boldsymbol{R}}^{n\times p} = \hat{\boldsymbol{A}}^{3\times 3} \cdot \hat{\boldsymbol{R}}^{3\times p}$ matrix operation requires at most $n(lp+l+p-1)/2 = (6p+3)$ polynomial multiplications and $(104p-132)/2$ polynomial additions if it is done by using Rosowski's method. Then, the total number of operations can be computed as:

$$
\begin{aligned}
Total &= (6p+3)M + \frac{104p-132}{2}A \\
&= (6p+3)((3n\log n)a + (\frac{3}{2}n\log n + n)m) + \frac{104p-132}{2}((n-1)a)
\end{aligned}
$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$
\begin{aligned}
Total &= (6p+3)(\frac{9}{2}n\log n + n) + \frac{104p-132}{2}(n-1) \\
&= 27np\log n + 58np - 63n + \frac{27n\log n}{2} - 52p + 66
\end{aligned} \tag{3.17}
$$

Let $n = 256$, then Equation 3.17 is equal to $70092p + 11586$. Rosowski's method is more efficient than the classical method for $(3, 4, p)$ when p is odd. This inference can be observed from Table 3.6 and Figure 3.5.

Efficient Method for $(3, 3, p)$ When p is Even:

Let $p \geq 3$ even. Then, $\hat{\boldsymbol{A}}^{l\times n} \cdot \hat{\boldsymbol{R}}^{n\times p} = \hat{\boldsymbol{A}}^{3\times 3} \cdot \hat{\boldsymbol{R}}^{3\times p}$ matrix operation requires at most $(n(lp + l + p - 1) + l - 1)/2 = (6p + 4)$ polynomial multiplications and $(104p-176)/2$ polynomial additions if it is done by using Rosowski's method. Then, the total number of operations can be computed as:

$$\begin{aligned} Total &= (6p+4)M + \frac{104p-176}{2}A \\ &= (6p+4)((3n\log n)a + (\frac{3}{2}n\log n + n)m) + \frac{104p-176}{2}((n-1)a) \end{aligned}$$

Under the assumption that $m = a$, the total number of arithmetic operations is:

$$\begin{aligned} Total &= (6p+4)(\frac{9}{2}n\log n + n) + \frac{104p-176}{2}(n-1) \\ &= 27np\log n + 58np - 84n + 18n\log n - 52p + 88 \end{aligned} \tag{3.18}$$

Let $n = 256$, then Equation 3.18 is equal to $70092p + 15448$. When $p = 4$, the total number of operations for $(3, 3, 4)$ is 295816. Rosowski's method is more efficient than the classical method for $(3, 4, p)$ when p is even. This inference can be observed from Table 3.6 and Figure 3.5.

Table 3.6: Variation of the Total Number of Operations Required According to p for the Efficient and Classical Method for $(3, 3, p)$, and the Improvement Rates Provided by the Efficient Algorithm

p	**Classical Operations**	**Efficient Operations**	**Improvement (%)**
3	260334	221862	14.78
4	347112	295816	14.78
5	433890	362046	16.56
6	520668	436000	16.26
7	607446	502230	17.32
8	694224	576184	17.00
9	781002	642414	17.74
10	867780	716368	17.45

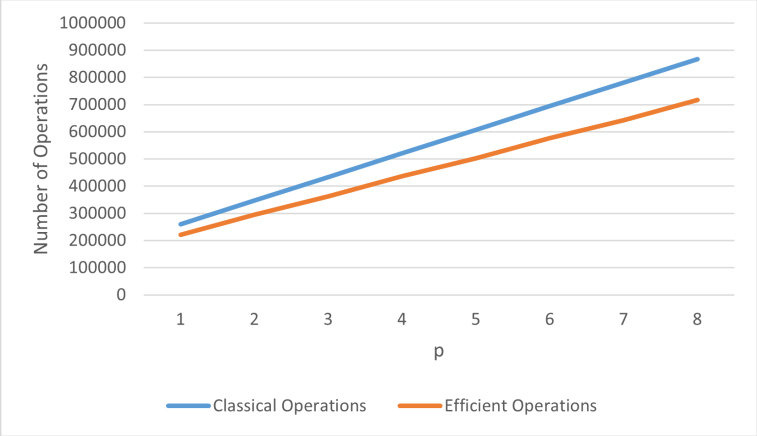

Figure 3.5: Variation of the Total Number of Operations Required According to p for the Efficient and Classical $(3, 3, p)$ Multiplication

CHAPTER 4

DILITHIUM SIGNATURE ALGORITHM FOR BATCH OPERATIONS

This chapter explains the proposed batch version of Dilithium's signing algorithm.

4.1 Batch Dilithium Signing

The batch Dilithium Signature Algorithm, which is based on the Dilithium Signature Algorithm specified in Algorithm 3 and allows signing more than one message at a time, is explained in Algorithm 7. As an example, batch numbers of the proposed algorithm are selected as 4, 5, and 4 for Dilithium 2, 3, and 5, respectively. How batch numbers can be selected is explained in detail in Section 4.3.

The inputs of the batch signing algorithm are m different messages and the user's secret key $sk = (\rho, K, tr, s_1, s_2, t_0)$. These messages are indexed as $M_0, M_1, \ldots, M_{m-1}$. First, the matrix $\hat{\boldsymbol{A}}$ is produced using ρ as in Algorithm 3. Then, μ_i, κ_i, and ρ'_i values corresponding to each message M_i are computed. κ_i values start with 0, and $(\boldsymbol{z'_i}, \boldsymbol{h'_i})$ are initialized to $\perp$. A $waitList = 0, 1, 2, \ldots, m-1$ list representing indices of messages waiting to be signed is defined. The first p indices in the list represent which messages are in the signing phase at the same time. At the end of the while loop, the indices of messages with appropriate signatures are deleted from this list. Just before the signature phase, $\boldsymbol{s_1}$, $\boldsymbol{s_2}$, and $\boldsymbol{t_0}$, which are the secret key elements, are converted to their NTT formats: $\hat{\boldsymbol{s_2}}$, $\hat{\boldsymbol{s_2}}$, and $\hat{\boldsymbol{t_0}}$. Once the preliminary preparations are completed, the signing loop begins. The candidate signatures produced must satisfy certain conditions. The status of these conditions is controlled by $status_i$.

Algorithm 7: Batch Dilithium Signing for m Different Messages

input : Secret key $sk = (\rho, K, tr, \boldsymbol{s_1}, \boldsymbol{s_2}, \boldsymbol{t_0})$, messages M_i, where $i = 0, 1, \ldots, m-1$

output : Signatures σ_i, where $i = 0, 1, \ldots, m-1$

$\hat{\boldsymbol{A}} \in R_q^{k \times l} := ExpandA(\rho)$ ▷ $\boldsymbol{A}$ is generated in NTT representation as $\hat{\boldsymbol{A}}$

for $i = 0, 1, \ldots, m-1$ **do**

  $\mu_i \in \{0,1\}^{512} := H(tr \| M_i)$, $\kappa_i := 0$, $(\boldsymbol{z_i'}, \boldsymbol{h_i'}) = \perp$, $\rho_i' \in \{0,1\}^{512} := H(K \| \mu_i)$

$waitList_i = i \quad \forall i = 0, 1, 2, \ldots, m-1$

$\hat{\boldsymbol{s}}_1 := NTT(\boldsymbol{s}_1)$, $\hat{\boldsymbol{s}}_2 := NTT(\boldsymbol{s}_2)$, $\hat{\boldsymbol{t}}_0 := NTT(\boldsymbol{t}_0)$

while *Length of* $waitList \geq p$ *(batch number)* **do**

  $status_i = 0 \quad \forall i = 0, 1, 2, \ldots, m-1$

  for $i = 0, 1, \ldots, p-1$ **do**

    $\boldsymbol{y}_i \in \tilde{S}_{\gamma_1}^l := ExpandMask(\rho'_{waitList_i}, \kappa_{waitList_i})$

  $\hat{\boldsymbol{Y}}$**:=** $(l \times p)$ matrix, whose columns are $NTT(\boldsymbol{y}_i)$'s

  $\hat{\boldsymbol{W}} := \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{Y}}$, $\hat{\boldsymbol{W}}$: $(k \times p)$ matrix, whose columns are $\hat{\boldsymbol{w_i}}$'s, where $i = 0, 1, 2, \ldots, p-1$

  $\boldsymbol{W} := (k \times p)$ matrix, whose columns are $NTT^{-1}(\hat{\boldsymbol{w_i}})$'s

  for $i = 0, 1, \ldots, p-1$ **do**

    $\boldsymbol{w_i'} := HighBits_q(\boldsymbol{i}, 2\gamma_2)$

    $\tilde{c}_{waitList_i} \in \{0,1\}^{256} := H(\mu_{waitList_i} \| \boldsymbol{w_i'})$

    $c_{waitList_i} \in B_\tau := SampleInBall(\tilde{c}_{waitList_i})$

    $\hat{c}_{waitList_i} = NTT(c_{waitList_i})$

    $\boldsymbol{z_i'} := \boldsymbol{y}_i + NTT^{-1}(\hat{c}_{waitList_i} \cdot \hat{\boldsymbol{s}}_1)$

    $\boldsymbol{r_i} := LowBits_q(\boldsymbol{w}_i - NTT^{-1}(\hat{c}_{waitList_i} \cdot \hat{\boldsymbol{s}}_2), 2\gamma_2)$

    if $\|\boldsymbol{z_i'}\|_\infty \geq \gamma_1 - \beta$ **then** $status_i = status_i + 1$

    if $\|\boldsymbol{r}_i\|_\infty \geq \gamma_2 - \beta$ **then** $status_i = status_i + 1$

    $\boldsymbol{h_i'} := MakeHint_q(-NTT^{-1}(\hat{c}_{waitList_i} \cdot \hat{\boldsymbol{t}}_0), \boldsymbol{w}_i - c_{waitList_i} \cdot \boldsymbol{s_2} + NTT^{-1}(\hat{c}_{waitList_i} \cdot \hat{\boldsymbol{t}}_0), 2\gamma_2)$

    if $\|c\boldsymbol{t}_0\|_\infty \geq \gamma_2$ **then** $status_i = status_i + 1$

    if *The # of 1's in* $\boldsymbol{h_i'}$ *is greater than* ω **then** $status_i = status_i + 1$

    $\kappa_{waitList_i} = \kappa_{waitList_i} + l$

  for $i = 0, 1, \ldots, p-1$ **do**

    if $!status_i$ **then**

      $\boldsymbol{z}_{waitList_i} = \boldsymbol{z_i'}$

      $\boldsymbol{h}_{waitList_i} = \boldsymbol{h_i'}$

      Delete $waitList_i$ from the list

return $\sigma_i = (\tilde{c}_i, \boldsymbol{z}_i, \boldsymbol{h}_i)$ where $i = 0, 1, \ldots, m-1$

Note that y_i's are computed using the ρ'_i's and κ_i's of the relevant messages and brought to their NTT format. The produced $\hat{\boldsymbol{y}}_{\boldsymbol{i}}$'s form the columns of the $\hat{\boldsymbol{Y}}$ matrix. The batch operation of the algorithm is carried out in the 11^{th} step. This step, which is matrix-vector multiplication in the original version, is turned into matrix-matrix multiplication. Each column of the matrix $\hat{\boldsymbol{W}}$ obtained by multiplication is converted to its normal form with NTT^{-1}. These are called w_i's and are used for p messages in the signing phase. In Steps 13–25, the operations performed for a single message in the original algorithm are performed for p messages. The candidate signatures generated for each message are checked with "if" statements, and as a result, the $status_i$'s are updated. Between stages 34 and 38, candidate signatures that meet all conditions are determined to be real signatures, and the indices of properly signed messages are deleted from the $waitList$. p messages waiting in the queue in the $waitList$ enter the loop with updated κ_i values (κ_i values are updated in step 25). The loop continues until less than p elements are left in the $waitList$. Messages that cannot be signed with Algorithm 7 are signed individually with the Original Dilithium Signature Algorithm (Algorithm 3). Thus, all messages are signed properly.

Table 4.1: Change of κ and $waitList$ According to the Number of Loops and Valid Signatures Generated

# of Loop	**Valid**	$\kappa = [\kappa_0, \kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa_6, \kappa_7]$	$waitList$
0 (start)	-	[0,0,0,0,0,0,0,0]	[0,1,2,3,4,5,6,7]
1	$(\tilde{c}_3, \boldsymbol{z}_3, \boldsymbol{h}_3)$	[4,4,4,4,0,0,0,0]	[0,1,2,4,5,6,7]
2	$(\tilde{c}_1, \boldsymbol{z}_1, \boldsymbol{h}_1)$	[8,8,8,4,4,0,0,0]	[0,2,4,5,6,7]
3	$(\tilde{c}_5, \boldsymbol{z}_5, \boldsymbol{h}_5)$	[12,8,12,4,8,4,0,0]	[0,2,4,6,7]
4	$(\tilde{c}_6, \boldsymbol{z}_6, \boldsymbol{h}_6)$	[16,8,16,4,12,4,4,0]	[0,2,4,7]
5	$(\tilde{c}_0, \boldsymbol{z}_0, \boldsymbol{h}_0)$	[20,8,20,4,16,4,4,4]	[2,4,7]

First, the matrix $\hat{\boldsymbol{A}}$ is generated via Algorithm 7 and μ_i, ρ'_i are computed for $i = 0, 1, 2, \ldots, 7$. Four messages are handled simultaneously since $p = 4$. The four messages that are processed are determined by the first four components of $waitList$. At the beginning, $waitList = [0, 1, 2, 3, 4, 5, 6, 7]$. Since the first four components are $[0, 1, 2, 3]$, the first while loop tries to generate signatures for the messages M_0, M_1, M_2, M_3. According to Table 4.1, the signature candidates produced at the end of the first loop are $(\tilde{c}_0, \boldsymbol{z}_0, \boldsymbol{h}_0)$, $(\tilde{c}_1, \boldsymbol{z}_1, \boldsymbol{h}_1)$, $(\tilde{c}_2, \boldsymbol{z}_2, \boldsymbol{h}_2)$, $(\tilde{c}_3, \boldsymbol{z}_3, \boldsymbol{h}_3)$. Only one of these four signature candidates satisfies the conditions and is valid, which

is $\sigma_3 = (\tilde{c}_3, \boldsymbol{z}_3, \boldsymbol{h}_3)$. The corresponding κ values are updated to [4,4,4,4,0,0,0,0]. Since the signature generated for M_3 is valid, three is removed from the $waitList$ and $waitList = [0, 1, 2, 4, 5, 6, 7]$. In the second round, signature candidates are produced for M_0, M_1, M_2, M_4. Assume that only $(\tilde{c}_1, \boldsymbol{z}_1, \boldsymbol{h}_1)$ is correct among the four signatures generated at the end of the second round. Then, $\kappa = [8, 8, 8, 4, 4, 0, 0, 0]$ and 1 is deleted from $waitList$ and the list becomes $waitList = [0, 2, 4, 5, 6, 7]$. This structure continues until the number of elements of $waitList$ is less than p. Thus, at the end of the Batch Dilithium 2 Signing algorithm, σ_3, σ_1, σ_5, σ_7, and σ_0 are produced, respectively.

Example: Let M_0, M_1, M_2, M_3, M_4, M_5, M_6, M_7 ($m = 8$) messages to be signed with Dilithium 2. The number of batch p is four and $\kappa = [\kappa_0, \kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa_6, \kappa_7]$ is initialized to [0,0,0,0,0,0,0,0] as in Algorithm 7, line 4. The signatures are generated in the while loop and checked to determine whether they comply with the required conditions. Let the generated signatures $(\tilde{c}_3, \boldsymbol{z}_3, \boldsymbol{h}_3)$, $(\tilde{c}_1, \boldsymbol{z}_1, \boldsymbol{h}_1)$, $(\tilde{c}_5, \boldsymbol{z}_5, \boldsymbol{h}_5)$, $(\tilde{c}_6, \boldsymbol{z}_6, \boldsymbol{h}_6)$, and $(\tilde{c}_0, \boldsymbol{z}_0, \boldsymbol{h}_0)$ are valid in order. For example, the change of κ and $waitList$ according to the number of loops and valid signatures generated is shown in Table 4.1.

The messages M_2, M_4, and M_7 that cannot be signed with the batch algorithm are signed individually using the original Dilithium 2 Signing Algorithm (Algorithm 3). One step further, it is possible to initialize κ values of the original Dilithium 2 signing algorithm to the values obtained at the end of the batch algorithm since some κ values are tried for these messages, and no correct results are obtained. Finally, all messages are signed by combining Algorithm 7 and Algorithm 3.

4.2 Making the Batch Algorithm More Efficient Compared to the Naive Approach

The process in step 11 of Algorithm 7 is emphasized to ensure that the batch algorithm is more efficient than the naive one. The matrix-vector multiplication, which is done separately for each message in the original algorithm, has now turned into matrix-matrix multiplication. Due to the structure of Dilithium, the elements of these

matrices are not integers but polynomials of degree 255. For this reason, although the matrix sizes are very small, one of the most expensive phases of the algorithm is the multiplication of these matrices. If these multiplications are done using the school-book method, the batch version does not seem to have an advantage over the original version. However, many matrix-matrix multiplication algorithms are available in the literature that can be used depending on the dimensions and properties of the matrices. If appropriate algorithms are chosen, the number of polynomial multiplications required by this operation can be reduced, and generating multiple signatures at once is more efficient than generating them one by one.

4.2.1 The Importance of Commutativity Property and Choosing The Proper Efficient Matrix-Matrix Multiplication Algorithms

The dimensions of the matrices to be multiplied in the 11^{th} step of Algorithm 7 according to the determined batch numbers and different security levels of Dilithium signing are shown in Table 4.2.

Table 4.2: Batch Numbers and Matrix Sizes According to Different Security Levels of Dilithium

Security Level	**Algorithm**	**Batch Number**	**Size of $\hat{A}$**	**Size of $\hat{Y}$**	**Size of $\hat{W}$**
2	Dilithium 2	4	(4×4)	(4×4)	(4×4)
3	Dilithium 3	5	(6×5)	(5×5)	(6×5)
5	Dilithium 5	4	(8×7)	(7×4)	(8×4)

Considering the matrix dimensions and structure of Dilithium, many matrix-matrix multiplication algorithms can be used. Dilithium's ring, $R = \mathbb{Z}_{8380417}[X]/(X^{256}+1)$, is commutative. For this reason, the multiplication of polynomials, which are the entries of matrices, is commutative. Thanks to this feature, the product of any two entries, a_{ij} and y_{kl} has the equality $a_{ij} \cdot y_{kl} = y_{kl} \cdot a_{ij}$. By taking advantage of this feature, a fast commutative matrix-matrix multiplication algorithm detailed in [29] can be used for all security levels. This algorithm works more efficiently than the Strassen [35] method or the non-commutative methods in the literature. Non-commutative methods can also be preferred when the matrix multiplication is performed recursively for larger-size matrices.

Strassen-like multiplications such as Strassen method, Cenk&Hassan's method [7], and Winograd's Multiplication [40] are known to be the best recursive matrix-matrix multiplications.

In this work, we derive the Batch Dilithium 2 algorithm, $(4, 4, 4)$ matrix-matrix multiplication formulas using Strassen's Method and the fast commutative method. Matrix multiplication formulas for $(6, 5, 5)$ and $(8, 7, 4)$ are obtained for Batch Dilithium 3 and Dilithium 5 using the fast commutative method.

4.3 Probability Computations and Choosing the Batch Sizes

Dilithium's signing algorithm requires specific requirements to be satisfied by signature candidates. The signature algorithm uses if statements to verify these conditions. Lines 20, 21, 23, and 24 in the batch algorithm (Algorithm 7) and 14, 18 in the classical algorithm (Algorithm 3) have these criteria. Step 14 is more dominant in the condition checks in Algorithm 3. Because of this, Formula 4.1 —which is defined in the [11]—is used to determine the probability that the signature created is valid (for step 14):

$$\approx e^{-256 \cdot \beta(l/\gamma_1 + k/\gamma_2)}. \tag{4.1}$$

The values of the variables included in the formula and the probabilities that a generated signature candidate will satisfy the requirements based on the various Dilithium security levels are given in Table 4.3. Signatures that do not satisfy the conditions are reproduced with the while loop.

Table 4.3: Variables Required to Compute the Probability

Variable	**Dilithium 2**	**Dilithium 3**	**Dilithium 5**
γ_1	2^{17}	2^{19}	2^{19}
γ_2	95232	261888	261888
(k, l)	(4,4)	(6,5)	(8,7)
β	78	196	120
Probability	≈ 0.24	≈ 0.196	≈ 0.26

Depending on the batch number, the probability of at least one of the p signatures produced by Dilithium 2, Dilithium 3, and Dilithium 5 being valid are calculated as $1-(1-0.24)^p$, $1-(1-0.196)^p$, and $1-(1-0.26)^p$, respectively.

Table 4.4: Probabilities of at Least one of p Signatures Being Correct According to Batch Number (p) for Dilithium 2, 3, and 5

	Probability		
p **(#of batch)**	**Dilithium 2**	**Dilithium 3**	**Dilithium 5**
3	0.561	0.480	0.595
4	0.666	0.582	0.700
5	0.746	0.664	0.778
6	0.807	0.730	0.836
7	0.854	0.783	0.878
8	0.889	0.825	0.910
9	0.915	0.860	0.933
10	0.936	0.887	0.951
11	0.951	0.909	0.964
12	0.963	0.927	0.973
13	0.972	0.941	0.980
14	0.979	0.953	0.985
15	0.984	0.962	0.989
16	0.988	0.970	0.992
17	0.991	0.975	0.994
18	0.993	0.980	0.996
19	0.995	0.984	0.997
20	0.996	0.987	0.998

The probability that at least one of the p signatures is valid increases with the batch number. When calculating the batch number p, two scenarios need to be taken into account. The first is the probability that when p number of messages are signed, at least one of these signatures is correct. The second is the rate of improvement that the chosen p will provide. Based on these two scenarios, the user may determine the priorities and select an appropriate batch number. Moreover, while choosing the batch number, the formulas to be derived based on the matrix sizes and the possible matrix-matrix multiplication algorithms should also be considered.

In Table 4.4, the probabilities that at least one of the p signatures is valid based on some eligible batch numbers (p) are given for Dilithium 2, 3, and 5.

Figures 4.1, 4.2, and 4.3 show the change of the probability of at least one of p signatures being correct according to the batch number p for Dilithium 2, 3, and 5, respectively. Table 4.4 and Figures 4.1, 4.2, and 4.3 show that the probability converges to 1 as the number of batches increases.

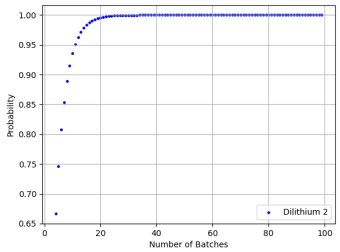

Figure 4.1: Change in Probability of at Least one of p Signatures Generated with Dilithium 2 Being Correct According to Batch Number (p)

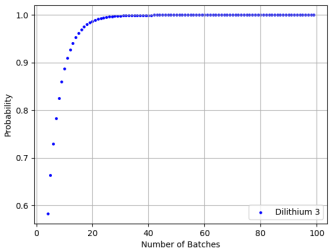

Figure 4.2: Change in Probability of at Least one of p Signatures Generated with Dilithium 3 Being Correct According to Batch Number (p)

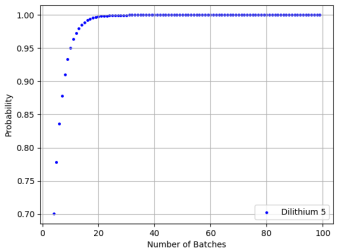

Figure 4.3: Change in Probability of at Least one of p Signatures Generated with Dilithium 5 Being Correct According to Batch Number (p)

The number of polynomial multiplication operations that need to be performed according to the selected batch number is given in Table 4.5 for the classical and the batch Dilithium signing algorithm. For the batch algorithm, Rosowski's algorithm, which is explained in Section 3.1.4, is used as the efficient matrix-matrix multiplication method. The required number of multiplications for the batch Dilithium algorithm is computed by using Formula 3.1, where $x = 4$, $y = 4$, and $z = p$. On the other hand, Formula 3.2 gives the number of multiplication operations performed in Dilithium 3 and 5, where p is odd. In this formula, $x = 6$, $y = 5$, $z = p$ for Dilithium 3 and $x = 8$, $y = 7$, $z = p$ for Dilithium 5. Similarly, Formula 3.3 is applied on Dilithium 3 and 5 if p is even. The classical algorithm requires $x \cdot y \cdot p$ multiplications, where (x, y) is the size of the public matrix $\hat{\boldsymbol{A}}$.

Table 4.5: Required Number of Multiplications for a Single Signature Assuming one of the p Signatures is Correct

	Dilithium 2		**Dilithium 3**		**Dilithium 5**	
p (# of Batch)	**Batch**	**Classical**	**Batch**	**Classical**	**Batch**	**Classical**
3	36	48	65	90	119	168
4	46	64	85	120	154	224
5	56	80	100	150	182	280
6	66	96	120	180	217	336
7	76	112	135	210	245	392
8	86	128	155	240	280	448
9	96	144	170	270	308	504
10	106	160	190	300	343	560
11	116	176	205	330	371	616
12	126	192	225	360	406	672
13	136	208	240	390	434	728
14	146	224	260	420	469	784
15	156	240	275	450	497	840
16	166	256	295	480	532	896
17	176	272	310	510	560	952
18	186	288	330	540	595	1008
19	196	304	345	570	623	1064
20	206	320	365	600	658	1120

For example, let 20 messages be signed with Batch Dilithium. The improvement rates that will be obtained by reducing the number of multiplications are given in Table 4.6. As can be observed from the table, the best improvement rate is obtained by selecting p as 5. According to improvement percentages and probability calculations, p can be selected by taking into account the available efficient matrix-matrix multiplication algorithms. Let C be the number of multiplications required for the classical method, B be the number of multiplications required for the efficient matrix-matrix multiplication algorithm used in the batch method, m be the number of messages to be signed, and p be the number of batches. The improvement rate is calculated with the following formula:

$$(Cm - (B(m - (p - 1)) + C(p - 1)))100/(Cm) \tag{4.2}$$

It can be observed in Table 4.6 that the rate of improvement for 20 messages provided by the batch algorithm decreases when p is increasing. Furthermore, similar tables

can be created using Formula 4.2 for various numbers of messages, and improvement rates can be observed.

Table 4.6: Improvement Rates (%) for 20 Signatures

	Dilithium 2	**Dilithium 3**	**Dilithium 5**
p (#of Batch)	**Impr (%)**	**Impr (%)**	**Impr (%)**
3	22.50	25.00	26.25
4	23.91	24.79	26.56
5	24.00	26.67	28.00
6	23.44	25.00	26.56
7	22.50	25.00	26.25
8	21.33	23.02	24.38
9	20.00	22.22	23.33
10	18.56	20.17	21.31
11	17.05	18.94	19.89
12	15.47	16.88	17.81
13	13.85	15.38	16.15
14	12.19	13.33	14.06
15	10.50	11.67	12.25
16	8.79	9.64	10.16
17	7.06	7.84	8.24
18	5.31	5.83	6.15
19	3.55	3.95	4.14
20	1.78	1.96	2.06

As observed in Table 4.3, the probabilities of signatures being valid for Dilithium 2, 3, and 5 are approximately 1/4, 1/5, and 1/4, respectively. So, one of the four signatures produced by Dilithium 2, one of the five for Dilithium 3, and one of the four for Dilithium 5 are expected to be valid with a high probability. Therefore, the probability that at least one of the four signatures produced by Dilithium 2 is valid is $1-(1-0.24)^4 \approx 0.67 = 67\%$. Similarly, the probability that at least one of the five signatures produced by Dilithium 3 is valid is 66%, and the probability that at least one of the four signatures produced by Dilithium 5 is valid is 70%. As an example, batch numbers are selected as 4, 5, and 4 for Dilithium 2, 3, and 5, respectively.

4.4 Arithmetic Complexity Analysis for Batch Dilithium Signing

The batch method must be used together with efficient matrix-matrix multiplication algorithms in order to be effective. For this reason, the batch numbers to be used in the Batch Dilithium Algorithm for each security level are determined by the method explained in Section 4.3. The dimensions of the matrices in the 11^{th} step of the Algorithm 7 according to each security level of Dilithium are given in Table 4.2. Since the entries of the matrices are polynomials and the multiplications of those entries are highly costly compared to their additions, the aim is to reduce the number of multiplications. The matrix-matrix multiplication algorithms in the literature with the minimum number of multiplications are generally obtained by using the Strassen-like multiplications recursively. However, since the sizes of the matrices in our work are small, they do not require recursions. So, the commutative matrix multiplication algorithms that have better multiplicative complexity than the non-commutative multiplication algorithms can also be used for our purposes since the entries are polynomials from the commutative ring R_q. Therefore, it is advantageous to use the commutative matrix-matrix multiplication algorithm described in [29].

4.4.1 Batch Dilithium 2 Signing

Assume that m messages are signed independently with Dilithium 2 using the classical approach. Each message needs 16 multiplications. However, due to the failure rate in the algorithm, $16 \cdot 4 = 64$ multiplications are expected to sign the message on average. Therefore, a total of $64m$ multiplications are needed for m messages.

In the batch method in Algorithm 7, step 11 is performed for four signatures. Since the probability that a signature candidate is valid for Dilithium 2 is approximately 0.24, one of the four signatures can be assumed to be valid. This process ends when $m - (p-1)$ messages are signed. Thus, the batch operation requires a total of $46 \cdot (m-(p-1)) = 46 \cdot (m-3)$ multiplication operations since $(4,4,4)$ operation can be obtained with 46 multiplications with Rosowski's method as explained in Section 3.2 and 3.1.4. The remaining $p - 1 = 3$ messages from the batch process are signed individually with classical Dilithium 2. Step 8 in the Algorithm 3 requires 16 multiplications to

sign each message. If we consider that the loop is repeated four times to generate a valid signature, the number of multiplications needed for a message is calculated as $16 \cdot 4 = 64$. Thus, the number of multiplications required for three messages is $3 \cdot 64 = 192$, and the number of multiplications required for batch Dilithium 2 is approximately $46 \cdot (m-3) + 192 = 46m + 54$. The number of multiplications required for the classical and batch methods, together with the improvement rates obtained with batch Dilithium 2, is presented in Table 4.7 for the selected number of messages up to 100. The improvement rate is calculated as $(64m - (46m + 54)) \cdot 100/64m$, and as the number of messages increases, this rate converges to 28.1%.

Table 4.7: Variation of the Number of Multiplications Required for Batch and Classical use of Dilithium 2, and the Improvement Rates Provided by Batch Method with [29] According to the Number of Messages

# of Messages m	**Batch** $(p = 4)$ $46m + 54$	**Classic** $64m$	**Improvement (%)**
4	238	256	7.0
5	284	320	11.25
6	330	384	14.1
7	376	448	16.1
8	422	512	17.58
9	468	576	18.85
10	514	640	19.69
20	974	1280	23.91
40	1894	2560	26.02
80	3734	5120	27.07
100	4654	6400	27.28

Figure 4.4: Variation of the Number of Multiplications Required According to the Number of Messages for Batch Dilithium 2 and Classical Dilithium 2

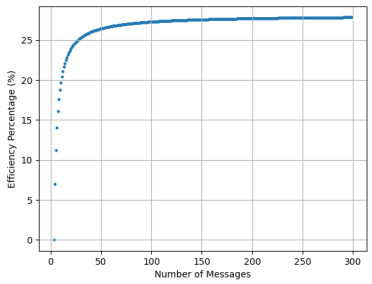

Figure 4.5: Variation of the Improvement rate (%) Provided by the Batch Algorithm Compared to the Classical Version Depending on the Number of Messages for Dilithium 2

Variation of the number of multiplications required according to the number of messages in Batch Dilithium 2 and Classical Dilithium 2 is shown in Figure 4.4. Variation of the improvement rate (%) provided by the batch algorithm compared to the classical version depending on the number of messages can be observed in Figure 4.5.

4.4.2 Batch Dilithium 3 Signing

If the classical method is used to sign m messages using Dilithium 3, it takes 30 multiplications for a message. But, because of the failure rate, $30 \cdot 5 = 150$ multiplications are done on average. For m messages, a total of $150m$ multiplications are expected.

The batch signature generation is performed for five signatures in Algorithm 7 at step 11. It is presumed that at least one of the five signatures is valid since the probability that a signature candidate is valid for Dilithium 3 is approximately 0.196. As a result of calculations similar to those in batch Dilithium 2, the batch operation requires a total of $100 \cdot (m-4)$ multiplication operations since $(6, 5, 4)$ operation can be obtained with 100 multiplications with Rosowski's method as explained in Section 3.2 and 3.1.4. When the remaining four messages are signed using the classical method, the total number of multiplications required for batch Dilithium 3 is $100m + 200$. According to the number of messages, the number of multiplications required for the classical or batch method, and the improvement rates obtained with batch Dilithium 3 are provided in Table 4.8. The improvement rate is computed using $(150m - (100m + 200)) \cdot 100/150m$. As the number of messages increases, this rate converges to 33.3%.

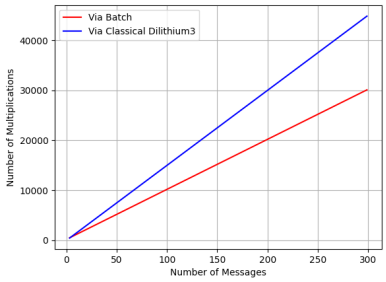

Figure 4.6: Variation of the Number of Multiplications Required According to the Number of Messages for Batch Dilithium 3 and Classical Dilithium 3

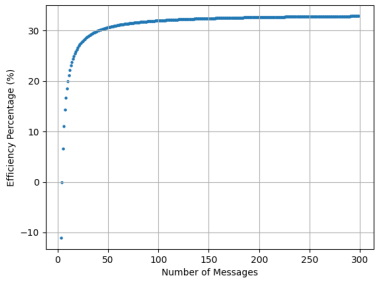

Figure 4.7: Variation of the Improvement Rate (%) Provided by the Batch Algorithm Compared to the Classical Version Depending on the Number of Messages for Dilithium 3

Variation of the number of multiplications required according to the number of messages in Batch Dilithium 3 and Classical Dilithium 3 is shown in Figure 4.6. Variation of the improvement rate (%) provided by the batch algorithm compared to the classical version depending on the number of messages can be observed in Figure 4.7.

Table 4.8: Variation of the number of multiplications required for batch and classical use of Dilithium 3, and the improvement rates provided by batch method with [29] according to the number of messages

# of Messages m	**Batch** ($p = 5$) $100m + 200$	**Classic** $150m$	**Improvement (%)**
5	700	750	6.7
6	800	900	11.11
7	900	1050	14.29
8	1000	1200	16.67
9	1100	1350	18.52
10	1200	1500	20.0
20	2200	3000	26.67
40	4200	6000	30.0
80	8200	12000	31.67
100	10200	15000	32.0

4.4.3 Batch Dilithium 5 Signing

Assume the classical method is used to sign m messages separately using Dilithium 5. It takes 56 multiplications for each message. Performing $56 \cdot 4 = 224$ multiplications is done under the assumption that a valid signature requires four repetitions. For m messages, a total of $224m$ multiplications must be done.

Similar to Batch Dilithium 2, 11^{th} step of the Algorithm 7 is performed for four signatures. Since the probability that a signature candidate is valid for Dilithium 5 is approximately 0.26, it is assumed that at least one of the four signatures will be valid. As a result of calculations similar to those in batch Dilithium 2 and 3, the batch operation requires a total of $154 \cdot (m - 3)$ multiplication operations since $(8, 7, 4)$ operation can be obtained with 154 multiplications with Rosowski's method as explained in Section 3.2 and 3.1.4. When the remaining three messages are signed using the classical method, the total number of multiplications required for batch

Dilithium 5 is $154m + 210$. According to the number of messages, the number of multiplications required for the classical or batch method, and the improvement rates obtained with batch Dilithium 5 are presented in Table 4.9.

Table 4.9: Variation of the Number of Multiplications Required for Batch and Classical Use of Dilithium 5, and the Improvement Rates Provided by Batch Method with [29] According to the Number of Messages

# of Messages m	**Batch** $(p = 4)$ $154m + 210$	**Classic** $224m$	**Improvement (%)**
4	826	896	7.81
5	980	1120	12.5
6	1134	1344	15.63
7	1288	1568	17.86
8	1442	1792	19.53
9	1596	2016	20.83
10	1750	2240	21.86
20	3290	4480	26.56
40	6370	8960	28.91
80	12530	17920	30.08
100	15610	22400	30.31

The improvement rate is computed using $(224m - (154m + 210)) \cdot 100/224m$. As the number of messages increases, this rate converges to 31.5%. Variation of the number of multiplications required according to the number of messages in Batch Dilithium 5 and Classical Dilithium 5 is shown in Figure 4.8. Variation of the improvement rate $(\%)$ provided by the batch algorithm compared to the classical version depending on the number of messages can be observed in Figure 4.9.

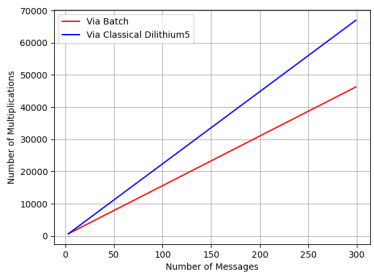

Figure 4.8: Variation of the number of multiplications required according to the number of messages for Batch Dilithium 5 and Classical Dilithium 5

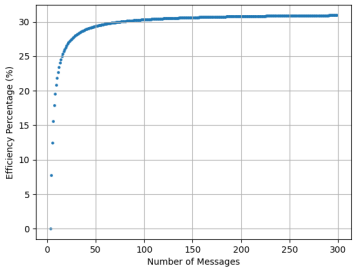

Figure 4.9: Variation of the improvement rate $(\%)$ provided by the batch algorithm compared to the classical version depending on the number of messages for Dilithium 5

4.5 Implementation Results

First, the proposed batch Dilithium signature algorithm is implemented in a way that is compatible with the structure of the classical Dilithium signature algorithm. Then, twenty random messages are generated to compare the batch implementations and the reference. Each message is signed individually using the reference implementation for CPU cycle counting. After that, the same messages are signed via the batch technique, and similar speed tests are performed. In this implementation, cycle counts are obtained utilizing a single core of the Intel Core i7-8700 processor.

Table 4.10 gives the cycle counts of only the multiplication process of signing 20 random messages with Classical Dilithium and Batch Dilithium. Using Batch Dilithium, 17 messages are signed, and CPU cycles are counted using the batch process. For the remaining three messages, classical Dilithium is used. The same 20 messages were also signed with Classical Dilithium. In order to make an accurate comparison, the cycle counts of signing 17 messages for Classical Dilithium are measured.

Table 4.10: CPU Cycle Counts of the Multiplication Stage Obtained by Signing 17 Random Messages ($m = 20 - 3$) with Reference and Batch Implementations of Dilithium 2, Dilithium 3, and Dilithium 5 (Cycle Counts are Obtained on One Core of Intel Core i7-8700)

Dilithium Signature Generation (Only the Multiplication Stage)				
Algorithm	**Batch Size**	**Reference**	**Batch**	**Improvement (%)**
Dilithium 2	4	2555202	1532297 (via [29])	40.03
Dilithium 2	4	2555202	1809040 (via [35])	29.20
Dilithium 3	5	4723348	3331311 (via [29])	29.47
Dilithium 5	4	5715468	4914677 (via [29])	14.01

Table 4.11 shows the tests' results for Dilithium's three security levels. Dilithium 2 is implemented using Strassen's algorithm and Rosowski's commutative algorithm. Since the latter has slightly better multiplicative complexity, it yields faster results. Dilithium 3 and Dilithium 5 are implemented only using Rosowski's method.

Table 4.11: CPU Cycle Counts Obtained by Signing 20 Random Messages ($m = 20$) with Reference and Batch Implementations of Dilithium 2, Dilithium 3, and Dilithium 5 (Cycle Counts are Obtained on One Core of Intel Core i7-8700)

Dilithium Signature Generation				
Algorithm	**Batch Size**	**Reference**	**Batch**	**Imprv. (%)**
Dilithium 2	4	31382991	20645253 (via [29])	34.22
Dilithium 2	4	31382991	20991482 (via [35])	33.11
Dilithium 3	5	50128969	41407391 (via [29])	17.40
Dilithium 5	4	46078440	41833217 (via [29])	10.15

4.6 Single Signature Using The Batch Dilithium Algorithm

The batch Dilithium signing algorithm is designed to sign m different messages. On the other hand, it can be used to sign a single message by making the following minor changes:

- Let M be the message to be signed. Then $M_i = M$ for all $i = 0, 1, \ldots, p-1$.
- μ and ρ' are obtained only once for a single message M.
- The iteration number x is used to indicate how many times the while loop repeats. $x = 0$ at the beginning.
- κ values are initialized as $\kappa_i = i \cdot l$ at the third step.
- There is no need to control the messages with the $waitList$ since there are only p similar messages that are processed.
- $status$ values are initialized as $status_i = 0$ where $i = 0, 1, 2, \ldots, p-1$.
- While loop continues until one of the signature candidates is correctly generated.

In this approach, $\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{p-1}$ are generated using κ_i and the same ρ' value. Then $NTT(\boldsymbol{y}_i) = \hat{\boldsymbol{y}}$ vectors become the columns of the matrix $\hat{\boldsymbol{Y}}$. At the end of the while loop, four different signature candidates for a single message are obtained. If one of them satisfies the conditions, then it is the correct signature for the message M. If

none of these p signature candidates satisfies the conditions, then the next $\boldsymbol{y}_i$ vectors are generated with the new $\kappa_i = (x \cdot p + i) \cdot l$ values. This process continues until a proper signature is obtained.

4.6.1 Complexity and Efficiency Analysis

Signing a single message efficiently with the batch Dilithium method depends on which κ value is required to sign the message correctly. In other words, it depends on how many times the while loop in Algorithm 3 repeats. Let κ_t be the required value to generate the proper signature and p be the batch number. Then, the while loop in Algorithm 3 repeats κ_t/l times, where l represents the number of columns of the public matrix $\hat{\boldsymbol{A}} \in R_q^{k \times l}$. It is not possible to know exactly how many iterations it will take for any message to be signed correctly. However, it is possible to calculate the number of iterations required with a probabilistic approach approximately.

Let P be the probability that a single message is correctly signed. Then, the expected number of iterations is $1/P$. If X is the number of multiplications needed to compute the matrix-vector or matrix-matrix operation, then the expected total cost can be computed as $(1/P) \cdot X$. Table 4.12 shows the expected number of iterations and expected total cost for classical and the batch method according to the change of the batch number p for Dilithium 2, 3, and 5. The following computations are performed to construct Table 4.12:

- The chance of a single vector being correctly signed using the classical approach is approximately 0.24, 0.196, and 0.26 for Dilithium 2, 3, and 5. Then, the expected number of iterations is $I_1 = (1/0.24) = 4.17$, $(1/0.196) = 5.10$, and $(1/0.26) = 3.85$ respectively.

- The chance of a single vector being correctly signed using the batch approach is approximately $(1 - (1 - 0,24)^p)$, $(1 - (1 - 0,196)^p)$, and $(1 - (1 - 0,26)^p)$ for Dilithium 2, 3, and 5. Then, the expected number of iterations is $I_4 = 1/(1 - (1 - 0,24)^p)$, $I_5 = 1/(1 - (1 - 0,196)^p)$, and $I_6 = 1/(1 - (1 - 0,26)^p)$ respectively.

- For the classical algorithm, the cost of computing $\hat{\boldsymbol{A}}^{l\times n}\cdot\hat{\boldsymbol{y}}^{n\times 1}$ is $C=l\cdot n$, and it is equal to 16, 30, and 56 for Dilithium 2, 3, and 5.
- When Rosowski's method is used to obtain $\hat{\boldsymbol{A}}^{l\times n}\cdot\hat{\boldsymbol{Y}}^{l\times p}$, the number of multiplications required is computed as follows:
  - Dilithium 2: $B_1=n(lp+l+p1)/2=10p+6$ since $n\geq 2$ and even.
  - Dilithium 3: $B_2=n(lp+l+p1)/2=(35p+25)/2$ if p is odd and $B_3=(n(lp+l+p1)+l1)/2=(35p+30)/2$ if p is even, since $n\geq 3$ and odd.
  - Dilithium 5: $B_4=n(lp+l+p1)/2=(63p+49)/2$ if p is odd and $B_5=(n(lp+l+p1)+l1)/2=(63p+56)/2$ if p is even, since $n\geq 3$ and odd.
- For the classical algorithm, the expected total cost is obtained as $C\cdot I_1$, $C\cdot I_2$, and $C\cdot I_3$ for Dilithium 2, 3, and 5.
- The expected total cost for the batch Dilithium 2 is obtained as $B_1\cdot I_4$.
- The expected total cost for batch Dilithium 3 is $B_2\cdot I_5$ for p is odd, and $B_3\cdot I_5$ for p is even.
- The expected total cost for batch Dilithium 5 is $B_4\cdot I_6$ for p is odd, and $B_5\cdot I_6$ for p is even.

Table 4.12: The Expected Total Cost and the Number of Iterations for the Batch and Classical Methods, Depending on the Variation in the Batch Number p for Dilithium 2, 3, and 5

	Dilithium 2				**Dilithium 3**				**Dilithium 5**			
	Classical		**Batch**		**Classical**		**Batch**		**Classical**		**Batch**	
p	**#of Iter.**	**Cost**	**#of Iter.**	**Cost**	**#of Iter.**	**Cost**	**#of Iter.**	**Cost**	**#of Iter.**	**Cost**	**#of Iter.**	**Cost**
2	4.17	66.67	2.37	61.55	5.10	153.06	-	-	3.85	215.38	-	-
3	4.17	66.67	1.78	64.17	5.10	153.06	2.08	135.34	3.85	215.38	1.68	200.08
4	4.17	66.67	1.50	69.03	5.10	153.06	1.72	146.01	3.85	215.38	1.43	219.96
5	4.17	66.67	1.34	75.02	5.10	153.06	1.51	150.59	3.85	215.38	1.29	233.90
6	4.17	66.67	1.24	81.75	5.10	153.06	1.37	164.41	3.85	215.38	1.20	259.63
7	4.17	66.67	1.17	89.04	5.10	153.06	1.28	172.45	3.85	215.38	1.14	278.89
8	4.17	66.67	1.13	96.77	5.10	153.06	1.21	187.79	3.85	215.38	1.10	307.67
9	4.17	66.67	1.09	104.87	5.10	153.06	1.16	197.76	3.85	215.38	1.07	329.96
10	4.17	66.67	1.07	113.28	5.10	153.06	1.13	214.17	3.85	215.38	1.05	360.76
11	4.17	66.67	1.05	121.96	5.10	153.06	1.10	225.46	3.85	215.38	1.04	385.03
12	4.17	66.67	1.04	130.86	5.10	153.06	1.08	242.71	3.85	215.38	1.03	417.25
13	4.17	66.67	1.03	139.95	5.10	153.06	1.06	254.96	3.85	215.38	1.02	442.84
14	4.17	66.67	1.02	149.20	5.10	153.06	1.05	272.87	3.85	215.38	1.01	476.03
15	4.17	66.67	1.02	158.59	5.10	153.06	1.04	285.84	3.85	215.38	1.01	502.49

According to Table 4.12, it is expected that signing a message with batch Dilithium 2 is more efficient than the classical approach when the batch number is selected as $p = 2$ or $p = 3$. Moreover, if $p = 3$, 4 or 5, batch Dilithium 3 can be used for a single message. Lastly, when the batch number is chosen as $p = 3$, it is expected that using batch Dilithium is more advantageous than the classical method.

4.7 Security of the Batch Signing Algorithm

The goal of the Batch Dilithium signature algorithm is to efficiently perform multiple signings without creating any security vulnerabilities. For this purpose, as in the classical Dilithium signature algorithm, μ_i and ρ'_i are generated for each message. In their generation, the relevant messages and parts of the secret key are used. Therefore, the values of μ_i and ρ'_i are unique to each message and are produced as in the classical algorithm. Then, using these values, the $\boldsymbol{y}_i$ vectors specific to each message are calculated. Thus, the security of the batch algorithm aligns with that of the classical Dilithium signing algorithm.The values and vectors calculated in the batch algorithm are identical to those in the classical algorithm. We provide efficiency by employing different computation techniques without changing the algorithms' structure. What distinguishes the batch algorithm from the classical one is that the operations are performed in multiple and efficient manners rather than one by one.

CHAPTER 5

DILITHIUM VERIFICATION ALGORITHM FOR BATCH OPERATIONS

This chapter explains the proposed batch version of the Dilithium verification scheme for signatures from the same signer.

5.1 Batch Dilithium Verification From a Single User

Algorithm 8 provides the specifics of the Batch Dilithium Verification algorithm, which is based on Algorithm 4. The goal of the batch verification algorithm is to validate multiple signatures in groups coming from the same user. Its input consists of signed messages, corresponding signatures, and the signer's public key. As a result, it calculates whether the signatures are valid or not.

At the beginning of the batch verification algorithm, the public matrix $\hat{\boldsymbol{A}}$ is generated in the NTT domain using ρ, which is part of the public key. Then, μ_i and c_i that are specific to each message are calculated. Since M_i and $\tilde{c}_i$ are used, μ_i and c_i are unique for each message. The most important step of the batch verification algorithm is to convert the matrix-vector multiplication $\hat{\boldsymbol{A}} \cdot NTT(z)$ contained in Algorithm 4 into matrix-matrix multiplication as seen in step 7 of Algorithm 8. m, the number of messages (or signatures), may not be divided by p, which is the number of batches. In this case, the messages are inserted into the matrix-matrix multiplication process in groups of p. This means that there are $\lfloor m/p \rfloor$ many matrix-matrix multiplications are performed. For the remaining messages, matrix-vector multiplication is performed as in the original algorithm. Thus, the $\hat{\boldsymbol{k}}_i$ are obtained. Then, $\boldsymbol{w_i'}$ and $status_i$, which

are unique for each signature, are calculated. Finally, the $status_i$ are given as output. They contain information about whether the signatures of each message are valid or not.

Algorithm 8: Batch Dilithium Verification for m Different Signatures from a Single User

input : Public key $pk = (\rho, \boldsymbol{t_1})$, messages M_i, signatures $\sigma_i = (\tilde{c}_i, \boldsymbol{z}_i, \boldsymbol{h}_i)$, where $i = 0, 1, \ldots, m-1$

output : Signatures σ_i are valid or not, where $i = 0, 1, \ldots, m-1$

$\hat{\boldsymbol{A}} \in R_q^{k \times l} := ExpandA(\rho)$ ▷ $\boldsymbol{A}$ is generated in NTT representation as $\hat{\boldsymbol{A}}$

for $i = 0, 1, \ldots, m-1$ **do**

  $\mu_i \in \{0,1\}^{512} := H(H(\rho \| \boldsymbol{t}_1) \| M_i)$

  $c_i := SampleInBall(\tilde{c}_i)$

for $i = 0, 1, \ldots, \lfloor m/p \rfloor - 1$ **do**

  $\hat{\boldsymbol{Z}}_i :=$ $(l \times p)$ matrix whose columns are $NTT(\boldsymbol{z}_{pi}), NTT(\boldsymbol{z}_{pi+1}), \ldots, NTT(\boldsymbol{z}_{pi+(p-1)})$

  $\hat{\boldsymbol{K}}_i = \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{Z}}_i$ whose columns are $\hat{\boldsymbol{k}}_{pi}, \hat{\boldsymbol{k}}_{pi+1}, \ldots, \hat{\boldsymbol{k}}_{pi+(p-1)}$

for $i = \lfloor m/p \rfloor, \ldots, m-1$ **do**

  $\hat{\boldsymbol{k}}_i = \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{z}}_i$ ▷ If $0 \not\equiv (m \mod p)$

$status_i = 1$ where $i = 0, 1, \ldots, m-1$

for $i = 0, 1, \ldots, m-1$ **do**

  $\boldsymbol{w}_i' := UseHint_q(\boldsymbol{h}_i, NTT^{-1}(\hat{\boldsymbol{k}}_{\boldsymbol{i}} - NTT(\boldsymbol{c}_i) \cdot NTT(\boldsymbol{t_1} \cdot \boldsymbol{2^d})), 2\gamma_2)$

  $status_i = [\| \boldsymbol{z}_i \|_\infty < \gamma_1 - \beta]$ **and** $[\tilde{c}_i = H(\mu_i \| \boldsymbol{w_i'})]$ **and** [# of 1's in $\boldsymbol{h}_i$ is $\leq \omega$]

return $status_i$ where $i = 0, 1, \ldots, m-1$

5.2 Making the Batch Algorithm More Efficient Compared to the Naive Approach

The sizes of the public matrix $\hat{\boldsymbol{A}}$, which are determined for different security levels of the Dilithium verification algorithm, are the same as the sizes of the public matrix of the Dilithium signature algorithm. As explained in detail in Section 4.4, in order for the matrix-matrix multiplication within the batch algorithm to be efficient, this operation must be performed with efficient multiplication algorithms. In order to make a sample implementation, batch sizes are selected to be the same as those used in the

batch signature algorithm. In Table 4.2, the dimensions given for the matrices $\hat{\boldsymbol{A}}$, $\hat{\boldsymbol{Y}}$, and $\hat{\boldsymbol{W}}$ in the batch signature algorithm are assigned to the matrices $\hat{\boldsymbol{A}}$, $\hat{\boldsymbol{Z}}_i$ and $\hat{\boldsymbol{K}}_i$ in the batch verification algorithm, respectively. Similarly, the efficient matrix multiplication algorithms [29] and [35] are used for efficient matrix-matrix multiplications. Since the entries of the matrices are polynomial, the decrease in the number of multiplications increases the efficiency of the algorithm.

Table 5.1: Required Number of Multiplications and Improvement Rates (%) for 20 Signatures (messages) for Dilithium Verification (for Kyber Encryption)

	Dilithium 2			**Dilithium 3**			**Dilithium 5**		
p (# of Batch)	**Batch**	**Classical**	**Imprv (%)**	**Batch**	**Classical**	**Imprv (%)**	**Batch**	**Classical**	**Imprv (%)**
3	248	320	22.50	450	600	25.00	826	1120	26.25
4	230	320	28.13	425	600	29.17	770	1120	31.25
5	224	320	30.00	400	600	33.33	728	1120	35.00
6	230	320	28.13	420	600	30.00	763	1120	31.88
7	248	320	22.50	450	600	25.00	826	1120	26.25
8	236	320	26.25	430	600	28.33	784	1120	30.00
9	224	320	30.00	400	600	33.33	728	1120	35.00
10	212	320	33.75	380	600	36.67	686	1120	38.75
11	260	320	18.75	475	600	20.83	875	1120	21.88
12	254	320	20.63	465	600	22.50	854	1120	23.75
13	248	320	22.50	450	600	25.00	826	1120	26.25
14	242	320	24.38	440	600	26.67	805	1120	28.13
15	236	320	26.25	425	600	29.17	777	1120	30.63
16	230	320	28.13	415	600	30.83	756	1120	32.50
17	224	320	30.00	400	600	33.33	728	1120	35.00
18	218	320	31.88	390	600	35.00	707	1120	36.88
19	212	320	33.75	375	600	37.50	679	1120	39.38
20	206	320	35.63	365	600	39.17	658	1120	41.25

Depending on the efficiency wanted to be achieved, different p values and also suitable, efficient matrix-matrix multiplication algorithms can be preferred in the Algorithm 8. Let the number of multiplications required by the selected efficient matrix-matrix multiplication algorithm be B, and the number of multiplications required by the matrix-vector multiplication performed while verifying a message be C. When the m signatures are verified with the classical Dilithium verification algorithm, $C \cdot m$ multiplications are required. If the signatures are verified by batch method, then $[B \cdot \lfloor m/p \rfloor + C \cdot (m - \lfloor m/p \rfloor \cdot p)]$ multiplications are needed. The improvement rate (%) provided by batch verification is calculated using the following formula:

$$(C \cdot m - [B \cdot \lfloor m/p \rfloor + C \cdot (m - \lfloor m/p \rfloor \cdot p)]) \cdot 100/(C \cdot m) \tag{5.1}$$

The values C and B are calculated in a similar way to that described in Section 4.3. In these calculations, the appropriate formula among Formulas 3.1, 3.2, and 3.3 are used according to the batch number p and the dimensions of the public matrix $\hat{\boldsymbol{A}}$.

Table 5.1 shows the improvement rates provided by the batch version (%) and the number of multiplications required for the verification of 20 signatures using classical and batch Dilithium 2, 3, and 5 according to the selected p.

5.3 Arithmetic Complexity Analysis for Batch Dilithium Verification

Due to the fact that they have similar matrix sizes and ring properties, the approach described in Section 4.4 for the batch Dilithium signing algorithm is also applied for batch Dilithium verification.

5.3.1 Batch Dilithium 2 Verification

Batch number p is selected as 4 for Batch Dilithium 2 as an example. In this case, in Algorithm 8 step 7, $\hat{\boldsymbol{A}}^{4\times4} \cdot \hat{\boldsymbol{Z}}^{4\times4}$ matrix multiplication occurs. This operation requires 46 multiplication if Rosowski's method [29] is used as an efficient matrix multiplication algorithm. To verify m signatures with this method, $\hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{Z}}$ matrix multiplication is computed $\lfloor m/p \rfloor$ times while $\hat{\boldsymbol{Z}}$ changes according to the different signature groups of 4. The signatures that are left are verified by using classical matrix-vector multiplication, and each requires 16 multiplications. Therefore, while $p = 4$, the number of multiplications required to verify m signatures with Batch Dilithium 2 is calculated with the following formula:

$$46 \cdot \lfloor m/p \rfloor + 16 \cdot (m - \lfloor m/p \rfloor \cdot p) \tag{5.2}$$

The number of multiplications required to verify m signature with classical Dilithium 2 is $16m$. The variance in the number of multiplications required for batch and classical use of Dilithium 2 verification, as well as the improvement rates provided by the batch method based on the number of messages, are shown in Table 5.2.

As it can be observed in Table 5.2, if $m \equiv 0 \mod p$, then the improvement rate is equal to 28.13%. While number of signatures m increases and $m \not\equiv 0 \mod p$, the rate converges to 28.13%. Figure 5.1 illustrates how the number of multiplications needed varies based on the number of signatures in Batch Dilithium 2 and Classical Dilithium 2. Figure 5.2 illustrates how the improvement rate (%) provided by the batch algorithm varies based on the number of signatures when compared to the classical version.

Table 5.2: Variation of the Number of Multiplications Required for Batch and Classical Use of Dilithium 2 Verification (Kyber1024 Encryption), and the Improvement Rates Provided by the Batch Method According to the Number of Signatures (Messages)

# of Signatures	**Batch** ($p = 4$)	**Classic**	**Improvement (%)**
4	46	64	28.13
5	62	80	22.50
6	78	96	18.75
7	94	112	16.07
8	92	128	28.13
9	108	144	25.00
10	124	160	22.50
20	230	320	28.13
40	460	640	28.13
70	814	1120	27.32
80	920	1280	28.13
90	1044	1440	27.50
100	1150	1600	28.13

Figure 5.1: Variation of the Number of Multiplications Required According to the Number of Signatures for Batch Dilithium 2 and Classical Dilithium 2 Verification

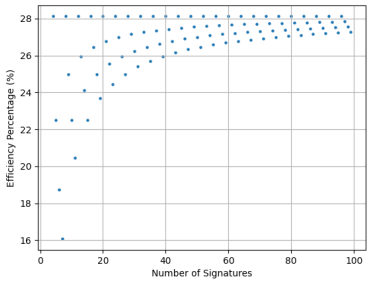

Figure 5.2: Variation of the Improvement Rate (%) Provided by the Batch Algorithm Compared to the Classical Version Depending on the Number of Signatures for Dilithium 2 Verification

As it can be observed in Table 5.2, if $m \equiv 0 \mod p$, then the improvement rate is equal to 28.13%. While number of signatures m increases and $m \not\equiv 0 \mod p$, the rate converges to 28.13%. Figure 5.1 illustrates how the number of multiplications needed varies based on the number of signatures in Batch Dilithium 2 and Classical Dilithium 2. Figure 5.2 illustrates how the improvement rate (%) provided by the batch algorithm varies based on the number of signatures when compared to the classical version.

5.3.2 Batch Dilithium 3 Verification

As a demonstration, let the batch number $p = 5$ for Batch Dilithium 3 Verification. This time, $\hat{\boldsymbol{A}}^{6\times5} \cdot \hat{\boldsymbol{Z}}^{5\times5}$ matrix multiplication is performed. Similar to Batch Dilithium 2 Verification, [29] is used as an efficient method, and it requires 100 multiplications for each matrix-matrix multiplication. If the remaining signatures are verified using the matrix-vector multiplication, 30 multiplications occur for each message. When Batch Dilithium 3 Verification is used for m signatures, the total number of multiplications needed is calculated by the following formula:

$$100 \cdot \lfloor m/p \rfloor + 30 \cdot (m - \lfloor m/p \rfloor \cdot p)$$

$30m$ multiplications are needed to validate a m signature using classical Dilithium 3. Table 5.3 illustrates the variation in the number of multiplications needed for batch and classical use of Dilithium 3 verification, together with the improvement rates provided by the batch approach based on the number of messages.

Table 5.3 illustrates that the improvement rate is equivalent to 33.33% if $m \equiv 0 \mod p$. The rate converges to 33.33% as long as the number of signatures increases and $m \not\equiv 0 \mod p$.

Figure 5.3 shows how the number of multiplications required in Batch Dilithium 3 and Classical Dilithium 3 differs according to the number of signatures. When compared to the classical version, Figure 5.4 shows how the improvement rate (%) obtainable via the batch algorithm changes depending on the number of signatures.

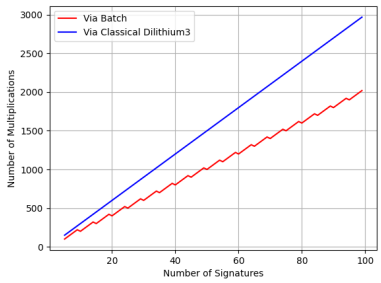

Figure 5.3: Variation of the Number of Multiplications Required According to the Number of Signatures for Batch Dilithium 3 and Classical Dilithium 3 Verification

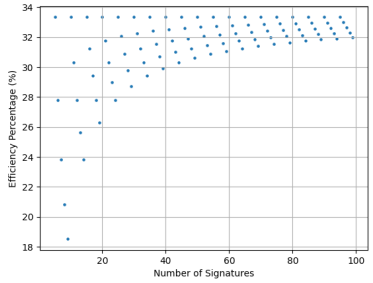

Figure 5.4: Variation of the Improvement Rate (%) Provided by the Batch Algorithm Compared to the Classical Version Depending on the Number of Signatures for Dilithium 3 Verification

Table 5.3: Variation of the Number of Multiplications Required for Batch and Classical Use of Dilithium 3 Verification, and the Improvement Rates Provided by the Batch Method According to the Number of Signatures

# of Signatures	**Batch** ($p = 5$)	**Classic**	**Improvement (%)**
5	100	150	33.33
6	130	180	27.78
7	160	210	23.81
8	190	240	20.83
9	220	270	18.52
10	200	300	33.33
33	690	990	30.30
47	960	1410	31.91
75	1500	2250	33.33
88	1790	2640	32.20
99	2020	2970	31.99
100	2000	3000	33.33

5.3.3 Batch Dilithium 5 Verification

The batch number p is selected as 4 for Batch Dilithium 5 Verification. Each matrix-matrix multiplication requires 154 multiplications if the fast commutative approach [29] is used for the $\hat{\boldsymbol{A}}^{8\times7} \cdot \hat{\boldsymbol{Z}}^{7\times4}$ operations. It takes 56 multiplications to perform the matrix-vector multiplication for each of the remaining messages.

The following formula can be used to determine the total number of multiplications needed to verify m signatures with Batch Dilithium 5 Verification:

$$154 \cdot \lfloor m/p \rfloor + 56 \cdot (m - \lfloor m/p \rfloor \cdot p)$$

Using classical Dilithium 5, $56m$ multiplications must be done to validate m signatures. The difference in the number of multiplications required for batch and classical use of Dilithium 5 verification, as well as the improvement rates offered by the batch technique dependent on the number of messages, are shown in Table 5.4.

Table 5.4: Variation of the Number of Multiplications Required for Batch and Classical Use of Dilithium 5 Verification, and the Improvement Rates Provided by Batch Method According to the Number of Signatures

# of Signatures	**Batch** ($p = 4$)	**Classic**	**Improvement (%)**
4	154	224	31.25
5	210	280	25.00
6	266	336	20.83
7	322	392	17.86
8	308	448	31.25
9	364	504	27.78
10	420	560	25.00
20	770	1120	31.25
40	1540	2240	31.25
70	2730	3920	30.36
80	3080	4480	31.25
90	3500	5040	30.56
100	3850	5600	31.25

The improvement rate is equal to 31.25% if $m \equiv 0 \mod p$, as Table 5.4 illustrates. As m increases and $m \not\equiv 0 \mod p$, the rate converges to 31.25%. The relationship between the number of signatures in Batch Dilithium 5 and Classical Dilithium 5 and the number of multiplications required is shown in Figure 5.5. Comparing the batch algorithm to the classical version, Figure 5.6 shows how the improvement rate (%) changes depending on the number of signatures.

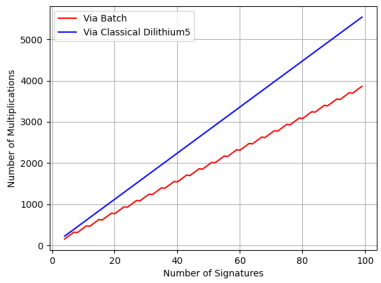

Figure 5.5: Variation of the Number of Multiplications Required According to the Number of Signatures for Batch Dilithium 5 and Classical Dilithium 5 Verification

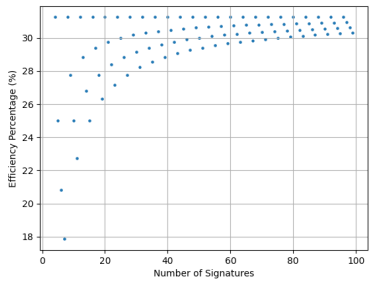

Figure 5.6: Variation of the Improvement Rate (%) Provided by the Batch Algorithm Compared to the Classical Version Depending on the Number of Signatures for Dilithium 5 Verification

5.4 Implementation Results

The proposed batch Dilithium verification algorithm is implemented to align with the structure of the classical Dilithium verification algorithm. Then, by using the Dilithium 2 signature algorithm, twenty random messages are signed, yielding twenty signatures. The reference and batch Dilithium 2, 3, and 5 are performed to validate these signatures. In Table 5.5, the CPU cycle counts obtained as a result of speed tests performed for different security levels of Dilithium, and also the improvement rates provided by batch implementation are given. In the batch implementation of Dilithium 2, Strassen's method [35] and Rosowski's method [29] are used as efficient matrix-matrix multiplication algorithms. For Dilithium 3 and 5, only Rosowski's method is performed.

Table 5.5: CPU Cycle Counts Obtained by Verifying 20 Signatures ($m = 20$) with the Reference and the Batch Implementations of Dilithium 2, Dilithium 3, and Dilithium 5 (Cycle Counts are Obtained on One Core of Intel Core i7-8700)

Dilithium Signature Verification				
Algorithm	**Batch Size**	**Reference**	**Batch**	**Imprv. (%)**
Dilithium 2	4	5549265	2781154 (using [29])	49.88
Dilithium 2	4	5549265	2850412 (using [35])	48.63
Dilithium 3	5	8935674	3877687 (using [29])	56.60
Dilithium 5	4	14988354	5833909 (using [29])	61.08

5.5 Security of the Batch Verification Algorithm

The aim of the Batch Dilithium verification algorithm is to verify multiple signatures efficiently without compromising security. Similar to the classical Dilithium verification algorithm, μ_i and c_i are generated for each message, using the relevant messages and parts of the public key. Consequently, μ_i and c_i values are unique to each message or signature and are derived in the same way as in the classical approach. Additionally, the $\hat{\boldsymbol{z}}_i$ vectors corresponding to each signature are used to form the $\hat{\boldsymbol{Z}}_i$ matrices. As a result, the security of the batch algorithm is equivalent to that of the classical Dilithium verification algorithm. The values and vectors computed in the batch algorithm are identical to those in the classical method. We enhance efficiency by utilizing alternative computational methods without modifying the structure of the algorithms. The key difference between the batch and classical algorithms lies in the fact that the batch algorithm performs the operations in a more efficient, simultaneous manner, rather than individually.

CHAPTER 6

KYBER ENCRYPTION ALGORITHM FOR BATCH OPERATIONS

This chapter explains the proposed batch version of Kyber's encryption protocol, designed for a single user's use case. This version aims to perform multiple encryption operations for a specific user in a batched manner.

6.1 Batch Kyber Encryption to a Single Recipient

Let m messages are wanted to be encrypted and sent to a user. The batch Kyber Encryption algorithm that allows it to be performed efficiently is described in Algorithm 9. The public key pk of the receiver, messages wanted to be encrypted M_i, and random coins r_i are taken as inputs where $i = 0, 1, \ldots, m-1$.

First, $\hat{\boldsymbol{t}}$ is obtained via the $Decode$ function and the public key pk. Then, ρ is derived from pk and used to generate the public matrix $\hat{\boldsymbol{A}}^T$ in the NTT domain. Next, with the help of the CBD and PRF functions, $\boldsymbol{r_j}$, $\boldsymbol{e}'_j$, and $\boldsymbol{e}''_j$ are computed to be specific to each message M_j, thanks to the unique random coins of the messages.

The matrix-vector multiplication $\hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{r}}$ in step 15 of Algorithm 5 is converted to matrix-matrix multiplication, as seen in step 18 of Algorithm 9. The matrix $\hat{\boldsymbol{R}}_i$, which is formed by groups of p of $\hat{\boldsymbol{r}}_j$'s generated uniquely for each message, is multiplied by the matrix $\hat{\boldsymbol{A}}$. This process is repeated $\lfloor m/p \rfloor$ times.

Algorithm 9: Batch Kyber Encryption for m Different Messages for a Single User

input : Public key pk, messages M_i, random coins r_i where $i = 0, 1, \ldots, m-1$

output : Ciphertexts c_i

$\hat{\boldsymbol{t}} := Decode_{12}(pk)$

$\rho = pk + 12 \cdot k \cdot n/8$

for $i = 0, 1, \ldots, m-1$ **do**

  for $j = 0, 1, \ldots, m-1$ **do**

    $\hat{\boldsymbol{A}}^T[i][j] := Parse(XOF(\rho, i, j))$ ▷ Generate $\hat{\boldsymbol{A}} \in R_q^{k \times k}$ in NTT domain.

for $j = 0, 1, \ldots, m-1$ **do**

  $N = 0$

  for $i = 0, 1, \ldots, k-1$ **do**

    $\boldsymbol{r}_j[i] := CBD_{\eta_1}(PRF(r_j, N))$

    $N = N + 1$

  for $i = 0, 1, \ldots, k-1$ **do**

    $\boldsymbol{e}'_j[i] := CBD_{\eta_2}(PRF(r_j, N))$

    $N = N + 1$

  $\boldsymbol{e}''_j := CBD_{\eta_2}(PRF(r_j, N))$

  $\hat{\boldsymbol{r}}_j := NTT(\boldsymbol{r}_j)$

for $i = 0, 1, \ldots, \lfloor m/p \rfloor - 1$ **do**

  $\hat{\boldsymbol{R}}_i :=$ $(k \times p)$ matrix whose columns are

  $NTT(\boldsymbol{r}_{pi}), NTT(\boldsymbol{r}_{pi+1}) \ldots, NTT(\boldsymbol{r}_{pi+(p-1)})$

  $\hat{\boldsymbol{K}}_i = \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{R}}_i$ whose columns are $\hat{\boldsymbol{k}}_{pi}, \hat{\boldsymbol{k}}_{pi+1}, \ldots, \hat{\boldsymbol{k}}_{pi+(p-1)}$

for $i = \lfloor m/p \rfloor \cdot p, \ldots, m-1$ **do**

  $\hat{\boldsymbol{k}}_i = \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{r}}_i$ ▷ If $0 \not\equiv (m \mod p)$

for $i = 0, 1, \ldots, m-1$ **do**

  $\boldsymbol{u}_i := NTT^{-1}(\hat{\boldsymbol{k}}_i) + \boldsymbol{e}'_i$

  $v := NTT^{-1}(\hat{\boldsymbol{t}}^T \cdot \hat{\boldsymbol{r}}_i) + e''_i + Decompress_q(Decode_1(M_i), 1)$

  $c' := Encode_{d_u}(Compress_q(\boldsymbol{u}_i, d_u))$

  $c'' := Encode_{d_v}(Compress_q(v_i, d_v))$

  $c_i = (c'_i \| c''_i)$

return c_i for all $i = 0, 1 \ldots m-1$

Example: To encrypt messages M_i where $i = 0, 1, \ldots, 21$ and $p = 4$, $\hat{\boldsymbol{K}}_i$'s are computed:

$$\hat{\boldsymbol{K}}_0 = [\hat{\boldsymbol{k}}_0 \quad \hat{\boldsymbol{k}}_1 \quad \hat{\boldsymbol{k}}_2 \quad \hat{\boldsymbol{k}}_3] = \hat{\boldsymbol{A}} \cdot [\hat{\boldsymbol{r}}_0 \quad \hat{\boldsymbol{r}}_1 \quad \hat{\boldsymbol{r}}_2 \quad \hat{\boldsymbol{r}}_3]$$
$$\hat{\boldsymbol{K}}_1 = [\hat{\boldsymbol{k}}_4 \quad \hat{\boldsymbol{k}}_5 \quad \hat{\boldsymbol{k}}_6 \quad \hat{\boldsymbol{k}}_7] = \hat{\boldsymbol{A}} \cdot [\hat{\boldsymbol{r}}_4 \quad \hat{\boldsymbol{r}}_5 \quad \hat{\boldsymbol{r}}_6 \quad \hat{\boldsymbol{r}}_7]$$
$$\hat{\boldsymbol{K}}_2 = [\hat{\boldsymbol{k}}_8 \quad \hat{\boldsymbol{k}}_9 \quad \hat{\boldsymbol{k}}_{10} \quad \hat{\boldsymbol{k}}_{11}] = \hat{\boldsymbol{A}} \cdot [\hat{\boldsymbol{r}}_8 \quad \hat{\boldsymbol{r}}_9 \quad \hat{\boldsymbol{r}}_{10} \quad \hat{\boldsymbol{r}}_{11}]$$
$$\hat{\boldsymbol{K}}_3 = [\hat{\boldsymbol{k}}_{12} \quad \hat{\boldsymbol{k}}_{13} \quad \hat{\boldsymbol{k}}_{14} \quad \hat{\boldsymbol{k}}_{15}] = \hat{\boldsymbol{A}} \cdot [\hat{\boldsymbol{r}}_{12} \quad \hat{\boldsymbol{r}}_{13} \quad \hat{\boldsymbol{r}}_{14} \quad \hat{\boldsymbol{r}}_{15}]$$
$$\hat{\boldsymbol{K}}_4 = [\hat{\boldsymbol{k}}_{16} \quad \hat{\boldsymbol{k}}_{17} \quad \hat{\boldsymbol{k}}_{18} \quad \hat{\boldsymbol{k}}_{19}] = \hat{\boldsymbol{A}} \cdot [\hat{\boldsymbol{r}}_{16} \quad \hat{\boldsymbol{r}}_{17} \quad \hat{\boldsymbol{r}}_{18} \quad \hat{\boldsymbol{r}}_{19}]$$

For the $\hat{\boldsymbol{r}}_i$'s of the remaining messages, matrix-vector multiplication is performed as in the original algorithm:

$$\hat{\boldsymbol{k}}_{\boldsymbol{20}} = \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{r}}_{20}$$
$$\hat{\boldsymbol{k}}_{\boldsymbol{21}} = \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{r}}_{21}$$

Finally, the ciphertext c_i's unique to each message M_i's are obtained through the $Encode$, $Decode$, $Compress$, and $Decompress$ functions.

6.2 Making the Batch Algorithm More Efficient Compared to the Naive Approach

Kyber's public matrix $\hat{\boldsymbol{A}}$'s sizes change according to the security level and are shown in Table 2.4. Similar to the Batch Dilithium Signing and Verification algorithms, the Batch Kyber Encryption algorithm contains matrix-matrix multiplication operation on a different ring $R'_q = \mathbb{Z}_{3329}[x]/(x^{256} + 1)$, and it can be performed efficiently with an appropriate matrix multiplication algorithm. To demonstrate it, the selected batch number and the sizes of the matrices $\hat{\boldsymbol{K}}_i$, $\hat{\boldsymbol{A}}$, and $\hat{\boldsymbol{R}}_i$ are given in Table 6.1. [35] is used in Kyber512, [29] in Kyber768. Also, both [29] and [35] algorithms are applied on Kyber 1024. Moreover, similar to the batch Dilithium algorithms, different p values and appropriate efficient matrix-matrix multiplication algorithms might be preferred in the Algorithm 9, depending on the efficiency that is needed.

Table 6.1: Batch Numbers and Matrix Sizes According to Different Security Levels of Kyber

Security Level	Algorithm	Batch Number	Size of $\hat{A}$	Size of $\hat{R}_i$	Size of $\hat{K}_i$
2	Kyber512	2	(2×2)	(2×2)	(2×2)
3	Kyber768	4	(3×3)	(3×4)	(3×4)
5	Kyber1024	4	(4×4)	(4×4)	(4×4)

Let A be the number of multiplications needed by an efficient matrix-matrix multiplication algorithm, and C be the number of multiplications needed by the matrix-vector multiplication that is done during message encryption. $C \cdot m$ multiplications are needed when the m messages are encrypted using the classical Kyber encryption technique. $[B \cdot \lfloor m/p \rfloor + C \cdot (m - \lfloor m/p \rfloor \cdot p)]$ multiplications are required if the messages are encrypted using the batch technique. 5.1 also provides the improvement rate (%) that batch encryption yields. The values C and B are determined similarly to the method outlined in Section 4.3. In these calculations, the appropriate formula from Formulas 3.1, 3.2, and 3.3 is applied based on the batch number p and the dimensions of the public matrix A for batch Kyber768 and Kyber1024. In batch Kyber512, Stassen's method is used as described in Section 3.1.1 for $p = 2$. The number of multiplications is taken as 7. In the case where $p \geq 2$, Winograd's method is applied as explained in Section 3.1.5, and the number of multiplications for batch Kyber512 are calculated with Formula 3.4.

Table 6.2: Required Number of Multiplications and Improvement Rates (%) for 20 Messages (Kyber Encryption)

	Kyber512			Kyber768			Kyber1024		
p (# of Batch)	Batch	Classical	Imprv (%)	Batch	Classical	Imprv (%)	Batch	Classical	Imprv (%)
2	70	80	12.50	-	-	-	260	320	18.75
3	74	80	7.50	144	180	20.00	248	320	22.50
4	70	80	12.50	140	180	22.22	230	320	28.13
5	68	80	15.00	132	180	26.67	224	320	30.00
6	68	80	15.00	138	180	23.33	230	320	28.13
7	70	80	12.50	144	180	20.00	248	320	22.50
8	68	80	15.00	140	180	22.22	236	320	26.25
9	66	80	17.50	132	180	26.67	224	320	30.00
10	64	80	20.00	128	180	28.89	212	320	33.75
11	71	80	11.25	150	180	16.67	260	320	18.75
12	70	80	12.50	148	180	17.78	254	320	20.63
13	69	80	13.75	144	180	20.00	248	320	22.50
14	68	80	15.00	142	180	21.11	242	320	24.38
15	67	80	16.25	138	180	23.33	236	320	26.25
16	66	80	17.50	136	180	24.44	230	320	28.13
17	65	80	18.75	132	180	26.67	224	320	30.00
18	64	80	20.00	130	180	27.78	218	320	31.88
19	63	80	21.25	126	180	30.00	212	320	33.75
20	62	80	22.50	124	180	31.11	206	320	35.63

The improvement rates (%) provided by the batch version and the number of multiplications needed for 20 messages to be encrypted using classical and batch Kyber512, Kyber768, and Kyber1024 are displayed in Table 6.2 based on the chosen p.

6.3 Arithmetic Complexity Analysis for Batch Kyber Encryption

Although the operations Dilithium and Kyber occur in different rings, R, and R', they contain the same operations, such as matrix-vector multiplication defined on the polynomial ring, and are based on the same mathematical problem. Therefore, for batch Kyber encryption, the method outlined in Section is also applicable.

6.3.1 Batch Kyber512 Encryption

As an example, the batch number p for batch Kyber512 encryption is set to 2. In this case, the matrix multiplication $\hat{\boldsymbol{A}}^{2\times 2} \cdot \hat{\boldsymbol{R}}_i^{2\times 2}$ occurs in step 18 of Algorithm 9. As an efficient matrix multiplication technique, this operation requires 7 multiplication using Strassen's approach [35] as explained in Section 3.2 and 3.1.1. With this method, $\hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{R}}_i$ matrix multiplication is performed $\lfloor m/p \rfloor$ times in order to encrypt m messages, whereas $\hat{\boldsymbol{R}}_i$ varies based on the distinct message groups of 2. The remaining messages require 4 multiplications each to be encrypted using classical matrix-vector multiplication. As a result, when $p = 2$, the following formula is used to determine how many multiplications are needed to encrypt m messages using Batch Kyber512:

$$7 \cdot \lfloor m/p \rfloor + 4 \cdot (m - \lfloor m/p \rfloor \cdot p)$$

$4m$ multiplications are needed to encrypt m messages using classical Kyber512. Table 6.3 illustrates the variation in the number of multiplications needed for batch and classical use of Kyber512 encryption, together with the improvement rates offered by the batch approach based on the number of messages.

If $m \equiv 0 \mod p$, then the improvement rate is equal to 12.50%, as Table 6.3 illustrates. The rate converges to 12.50% as long as m, the number of messages, increases and $m \not\equiv 0 \mod p$. The relationship between the number of messages in Batch Kyber512 and Classical Kyber512 and the number of multiplications required is shown

in Figure 6.1. In comparison with the classical version, Figure 6.2 shows how the improvement rate (%) available through the batch method changes depending on the number of messages.

Table 6.3: Variation of the Number of Multiplications Required for the Batch and Classical Use of Kyber512 Encryption, and the Improvement Rates Provided by the Batch Method According to the Number of Messages

# of Messages	**Batch** ($p = 2$)	**Classic**	**Improvement (%)**
2	7	8	12.50
3	11	12	8.33
4	14	16	12.50
5	18	20	10.00
6	21	24	12.50
7	25	28	10.71
8	28	32	12.50
9	32	36	11.11
10	35	40	12.50
20	70	80	12.50
40	140	160	12.50
71	249	284	12.32
85	298	340	12.35
93	326	372	12.37
100	350	400	12.50

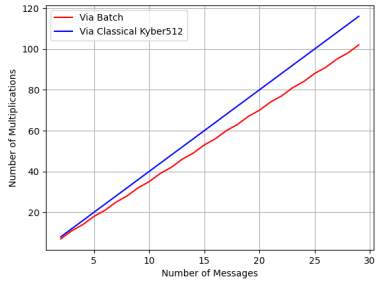

Figure 6.1: Variation of the Number of Multiplications Required According to the Number of Messages for Batch Kyber512 and Classical Kyber512 Encryption

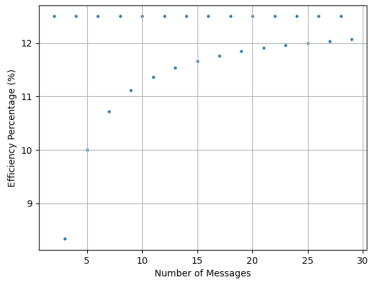

Figure 6.2: Variation of the Improvement Rate (%) Provided by the Batch Algorithm Compared to the Classical Version Depending on the Number of Messages for Kyber512 Encryption

6.3.2 Batch Kyber768 Encryption

In this case, for batch Kyber768 encryption, batch number p is assigned to 4, and matrix multiplication in Algorithm 9 becomes $\hat{\boldsymbol{A}}^{3\times 3} \cdot \hat{\boldsymbol{R}}_i^{3\times 4}$. With Rosowski's approach [29], this operation involves 28 multiplications as explained in Section 3.2 and 3.1.4. This method encrypts m messages by doing $\lfloor m/p \rfloor$ matrix multiplications of $\hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{R}}_i$, while $\hat{\boldsymbol{R}}_i$ changes due to the distinct message groups of 4. To encrypt the rest of the messages, standard matrix-vector multiplication requires 9 multiplications for each message. Hence, for $p = 4$, the number of multiplications needed to encrypt m messages using Batch Kyber768 is determined using the equation as follows:

$$28 \cdot \lfloor m/p \rfloor + 9 \cdot (m - \lfloor m/p \rfloor \cdot p)$$

For standard Kyber768, to encrypt m messages, $9m$ multiplications are required. The difference between the total number of multiplications required for batch and standard Kyber768 encryption, as well as the improvement rates provided by the batch technique depending on the number of messages, are summarized in Table 6.4.

Table 6.4: Variation of the Number of Multiplications Required for Batch and Classical Use of Kyber768 Encryption, and the Improvement Rates Provided by the Batch Method According to the Number of Messages

# of Messages	**Batch** ($p = 4$)	**Classic**	**Improvement (%)**
4	28	36	22.22
5	37	45	17.78
6	46	54	14.81
7	55	63	12.70
8	56	72	22.22
9	65	81	19.75
10	74	90	17.78
20	140	180	22.22
40	280	360	22.22
70	494	630	21.59
80	560	720	22.22
90	634	810	21.73
100	700	900	22.22

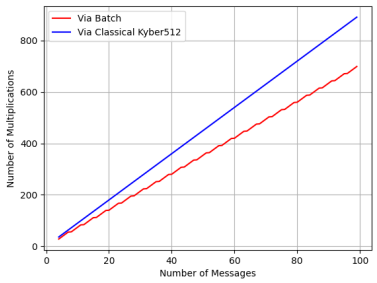

Figure 6.3: Variation of the Number of Multiplications Required According to the Number of Messages for Batch Kyber768 and Classical Kyber768 Encryption

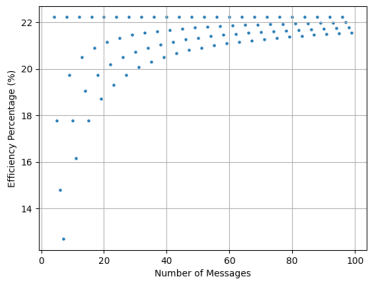

Figure 6.4: Variation of the Improvement Rate (%) Provided by the Batch Algorithm Compared to the Classical Version Depending on the Number of Messages for Kyber768 Encryption

Table 6.4 indicates that the improvement rate is equivalent to 22.22% if $m \equiv 0 \mod p$. As long as m, the number of messages, increases and $m \not\equiv 0 \mod p$, the rate converges to 22.22%. Figure 6.3 illustrates the relationship between the number of messages in the batch and the classical Kyber768 and the number of multiplications needed. Figure 6.4 illustrates how the improvement rate (%) made possible by the batch approach varies based on the number of messages when compared to the classical structure.

6.3.3 Batch Kyber1024 Encryption

The batch number p is also set to 4 when using batch Kyber1024 encryption. With respect to batch Kyber1024, the required number of multiplications to encrypt m messages can be determined via the Formula 5.2 since the matrix sizes and message grouping procedure used in the for loop are identical to the batch Dilithium2 verification. Table 5.2 displays the variance of the number of multiplications needed for batch and standard applications of Kyber2024 encryption, together with the improvement rates that the batch technique enables based on the number of messages.

The batch Kyber1024 encryption algorithm has graphs similar to those of the batch Dilithium2 verification algorithm. Figure 6.5 shows how the number of multiplications required in the batch and classical Kyber1024 differs according to the number of messages. When compared to the classical version, Figure 6.6 illustrates how the im-

provement rate (%) offered by the batch algorithm changes depending on the number of messages.

Figure 6.5: Variation of the Number of Multiplications Required According to the Number of Messages for Batch Kyber1024 and Classical Kyber1024 Encryption

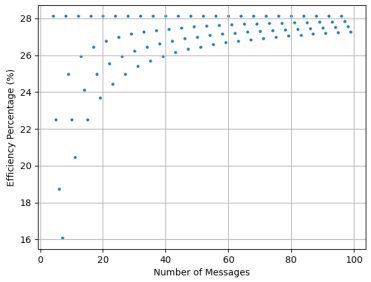

Figure 6.6: Variation of the Improvement Rate (%) Provided by the Batch Algorithm Compared to the Classical Version Depending on the Number of Messages for Kyber1024 Encryption

6.4 Implementation Results

The proposed batch Kyber encryption algorithm is designed to be compatible with the structure of the classical Kyber encryption algorithm. Then, twenty random messages are encrypted using the batch and the classical Kyber512, 768, and 1024 encryption algorithms.

Table 6.5: CPU Cycle Counts of the Multiplication Stage Obtained by Encrypting 20 Random Messages ($m = 20$) with Reference and Batch Implementations of Kyber512, Kyber768, and Kyber1024 (Cycle Counts are Obtained on One Core of Intel Core i7-8700)

Kyber Encryption (Only the Multiplication Stage)				
Algorithm	**Batch Size**	**Reference**	**Batch**	**Improvement (%)**
Kyber 512	2	322509	249408 (via [39])	22.67
Kyber 768	4	614922	443653 (via [29])	27.85
Kyber 1024	4	1069849	737093 (via [29])	31.10
Kyber 1024	4	1069849	830175 (via [35])	22.40

The CPU cycle counts derived from speed tests made for various Kyber security levels, along with the improvement rates offered by batch implementation, are given in Table 6.6. Table 6.5 gives the cycle counts of only the multiplication process of en-

crypting 20 random messages with the classical and the batch Kyber. Two efficient matrix-matrix multiplication algorithms are utilized in the batch implementation of Kyber1024: Rostowski's approach [29] and Strassen's method [35]. While [39] is preferred for Kyber512, [29] is used in Kyber768.

Table 6.6: CPU Cycle Counts Obtained by Encrypting 20 Random Messages ($m = 20$) with Reference and Batch Implementations of Kyber512, Kyber768, and Kyber1024 (Cycle Counts are Obtained on One Core of Intel Core i7-8700)

Kyber Encryption				
Algorithm	**Batch Size**	**Reference**	**Batch**	**Improvement (%)**
Kyber512	2	2706716	2102105 (via [35])	22.34
Kyber768	4	3941892	2992896 (via [29])	24.07
Kyber1024	4	5766383	4102297 (via [35])	28.86
Kyber1024	4	5766383	3988733 (via [29])	30.83

6.5 Security of the Batch Encryption Algorithm

As with the classical Kyber encryption algorithm, random coins are used as input, unique to each message. These values are employed to generate $\boldsymbol{r}_j$, $\boldsymbol{e}'_j$, and $\boldsymbol{e}''_j$ vectors, which are identical to those produced in the classical algorithm. Similar to the classical Kyber encryption system, operations are performed using these vectors in conjunction with the public matrix $\hat{\boldsymbol{A}}$. The ciphertexts obtained for a single user through the batch method are equivalent to those generated by the classical algorithm. We improve efficiency by applying various computational techniques while preserving the original structure of the algorithms. The primary distinction between the batch and classical algorithms lies in the fact that the batch algorithm executes the encryption process in a more efficient, simultaneous manner. No element critical to the security of the algorithm is altered. Consequently, the security of the batch algorithm is equivalent to that of the classical algorithm.

CHAPTER 7

PLATFORM INDEPENDENCE AND FUTURE DIRECTIONS

This work introduces a novel method for batch Dilithium signature generation, batch Dilithium signature verification coming from a single user, and Kyber encryption to a single recipient by converting matrix-vector product (MVP) operations into matrix-matrix product (MMP) operations. The main idea behind this method is that, when verifying/generating multiple signatures or encrypting multiple messages, instead of doing these operations individually, using the new batch method with efficient matrix multiplication algorithms, results in a significant decrease in computational cost. Our theoretical analysis reveals that this approach reduces computational complexity. We have validated these improvements both theoretically and through practical application in a reference implementation, with results confirming our theoretical predictions.

The main point of this work is that the computational improvements we get are independent of the specific platform or hardware architecture. The reduction in complexity is algorithmic itself—replacing multiple MVPs with a single MMP—rather than from any platform-specific optimizations. Our method's benefits can be experienced on any processor, instruction set, or architecture. Whether the algorithm is implemented on a general-purpose CPU, a GPU, or even specialized cryptographic hardware, the fundamental improvement in computational complexity will still hold.

Our reference implementation is specifically designed to showcase these computational benefits. By modifying an existing system to integrate our algorithm, we observed performance gains that aligned with our computational cost analysis. Importantly, these enhancements derive from a mathematical restructuring of the algorithm

rather than from specific hardware optimizations, demonstrating that our approach is broadly applicable across various platforms.

While we illustrate the platform independence of our method, it is important to note that implementing platform-specific optimizations, such as AVX (Advanced Vector Extensions) or other instruction sets, could lead to substantial performance gains. By customizing our approach to use these optimizations, we can further improve the algorithm's performance on specific hardware platforms.

AVX, for example, is an instruction set that allows for parallel processing of multiple data points in a single instruction, which can be highly beneficial for operations like matrix-matrix multiplication. However, the use of AVX or other hardware-specific features is not necessary to realize the core computational benefits of our method. The 25% improvement we have demonstrated is due to a reduction in the number of high-level operations and applies regardless of whether AVX or similar optimizations are employed. The implementation could be made faster by using AVX or similar instruction sets. However, this would require expertise in low-level programming and hardware-specific optimizations, which is not the focus of this work. Our contribution is in reducing high-level computational complexity, and this improvement applies to all platforms, whether or not platform-specific optimizations are used.

Future research could explore platform-specific optimizations, such as using AVX, AVX2, or AVX-512 instruction sets for matrix-matrix multiplication, as well as similar SIMD extensions on other architectures like ARM. Our algorithm's performance could be improved further by using a GPU-based implementation. This would take advantage of the highly parallel nature of matrix operations. It's important to highlight that even though these optimizations may yield added performance advantages, they do not alter the core enhancement in computational complexity brought about by our method.

In conclusion, our new method significantly reduces computational complexity by replacing multiple matrix-vector products with a single matrix-matrix product in the processes of Dilithium verification, signing, and Kyber encryption. This improvement is independent of any specific platform or hardware. While future work could explore hardware-specific optimizations like AVX, the core computational benefits

of our approach are universally applicable. We believe this research establishes a solid foundation for further platform-specific optimizations, but the primary contribution is the mathematical simplification of computational steps, which will enhance performance across all implementations.

CHAPTER 8

CONCLUSION

This study introduces the design of batch signature generation and verification algorithms for Dilithium's various security levels (Dilithium 2, 3, and 5) and a batch encryption algorithm for Kyber's security levels (Kyber512, Kyber768, and Kyber1024). The Dilithium signature generation, verification, and Kyber encryption algorithms are thoroughly analyzed and redesigned to support batch processing. Instead of executing separate operations for each message or signature, the study converts the matrix-vector multiplications in these algorithms into matrix-matrix multiplications, making the batch method more efficient. Additionally, the batch Dilithium signature generation algorithm is shown to be applicable for single signature generation as well, with efficiency and probability calculations confirming its effectiveness.

The designed batch algorithms retain the security properties of the original Dilithium and Kyber, allowing for batch processing without compromising security. The transition from matrix-vector to matrix-matrix operations enables the use of efficient non-commutative or commutative multiplication techniques. These efficiency gains are driven by mathematical improvements, independent of any particular architecture, platform, or device.

The designed batch algorithms offer the flexibility to choose different batch sizes, applying suitable matrix-matrix multiplication techniques as needed. In this regard, probability calculations are made to evaluate the performance of the Dilithium signature algorithm for both single and multiple signatures across different batch sizes, with efficiency enhancements assessed based on the selected multiplication methods. Furthermore, the efficiency improvements for batch Kyber encryption and Dilithium

verification are determined using matrix-matrix multiplication techniques customized for various batch sizes.

To validate these theoretical improvements, specific batch values and suitable multiplication methods are selected, and multiplication formulas are derived for the three security levels of both Dilithium and Kyber. These batch algorithms, along with the derived matrix-matrix multiplication formulas, are implemented in the C programming language. Comparisons between the newly developed batch algorithms and their reference counterparts highlight the efficiency improvements.

The results indicate that for Dilithium 2, 3, 5, Kyber768, and Kyber1024, using the method from [29] is more effective, while methods from [35] and [39] are more suitable for Kyber512. Batch Dilithium 2 and Kyber1024 involve the multiplication of two square matrices of size $2^n \times 2^n$, making them compatible with recursive application of techniques from [7], [35], and [40]. However, the method in [29] is non-recursive due to its requirement for commutative entries, which is incompatible with non-commutative matrix multiplications.

For batch Dilithium signature at three security levels, arithmetic complexity improvements of 28.1%, 33.3%, and 31.5% are observed. CPU cycle counts are improved by 34.22%, 17.40%, and 10.15%, respectively. Batch Dilithium verification, which uses the same multiplication formulas, shows arithmetic complexity improvements of 28.13%, 33.33%, and 31.25%, with CPU cycle reductions of 49.88%, 56.60%, and 61.08%. The batch Kyber encryption demonstrates improvements in arithmetic complexity of 12.50%, 22.22%, and 28.13%, and CPU cycle reductions of 22.34%, 24.07%, and 30.83% for its three security levels when efficient multiplication algorithms are applied.

In the batch method, the full matrix is used instead of just a single vector because matrix-matrix multiplication is employed rather than matrix-vector multiplication. This approach increases the amount of stack memory required for calculating and storing the linear combinations involved in the multiplication process, as well as when extracting matrix elements via addition and subtraction. However, with optimized implementation techniques —such as using loops for linear combinations or applying parallelization— memory usage can be minimized. The need for dynamic memory

allocation arises when performing these matrix operations, as the algorithm involves handling entire matrices rather than vectors. Naturally, this creates a trade-off between time and memory. Nonetheless, our calculations show that the impact is minor, as the matrix is built by adding only a few additional vectors.

REFERENCES

[1] C. Aguilar-Melchor, M. R. Albrecht, T. Bailleux, N. Bindel, J. Howe, A. Hülsing, D. Joseph, and M. Manzano, Batch signatures, revisited, in *Cryptographers' Track at the RSA Conference*, pp. 163–186, Springer, 2024.

[2] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, C. Miller, D. Moody, R. Peralta, et al., Status report on the third round of the nist post-quantum cryptography standardization process, US Department of Commerce, NIST, 2022.

[3] J.-P. Aumasson, D. J. Bernstein, W. Beullens, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, et al., Sphincs, Technical report, Stanford Univ., Tech. Rep, 2019.

[4] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, Crystals-kyber algorithm specifications and supporting documentation, NIST PQC Round, 2(4), pp. 1–43, 2019.

[5] M. Bellare, J. A. Garay, and T. Rabin, Fast batch verification for modular exponentiation and digital signatures, in *Advances in Cryptology—EUROCRYPT'98: International Conference on the Theory and Application of Cryptographic Techniques Espoo, Finland, May 31–June 4, 1998 Proceedings 17*, pp. 236–250, Springer, 1998.

[6] D. Benjamin, Batch signing for tls, Internet Engineering Task Force, Internet-Draft draft-davidben-tls-batch-signing-02, 2019.

[7] M. Cenk and M. A. Hasan, On the arithmetic complexity of strassen-like matrix multiplications, Journal of Symbolic Computation, 80, pp. 484–501, 2017.

[8] Y.-S. Chang, T.-C. Wu, and S.-C. Huang, Elgamal-like digital signature and multisignature schemes using self-certified public keys, Journal of Systems and Software, 50(2), pp. 99–105, 2000.

[9] J. H. Cheon, J.-S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun, Batch fully homomorphic encryption over the integers, in *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*, pp. 315–335, Springer, 2013.

[10] W. Diffie and M. Hellman, New directions in cryptography, IEEE transactions on Information Theory, 22(6), pp. 644–654, 1976.

[11] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, Crystals-dilithium algorithm specifications and supporting documentation (version 3.1), 2021.

[12] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE transactions on information theory, 31(4), pp. 469–472, 1985.

[13] A. L. Ferrara, M. Green, S. Hohenberger, and M. Ø. Pedersen, Practical short signature batch verification, in *Cryptographers' Track at the RSA Conference*, pp. 309–324, Springer, 2009.

[14] A. Fiat, Batch rsa, in *Advances in Cryptology—CRYPTO'89 Proceedings 9*, pp. 175–185, Springer, 1990.

[15] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, Z. Zhang, et al., Falcon: Fast-fourier lattice-based compact signatures over ntru, Submission to the NIST's post-quantum cryptography standardization process, 36(5), pp. 1–75, 2018.

[16] S.-J. Hwang and Y.-H. Lee, Repairing elgamal-like multi-signature schemes using self-certified public keys, Applied mathematics and computation, 156(1), pp. 73–83, 2004.

[17] D. Johnson, A. Menezes, and S. Vanstone, The elliptic curve digital signature algorithm (ecdsa), International journal of information security, 1, pp. 36–63, 2001.

[18] A. S. Kittur and A. R. Pais, Batch verification of digital signatures: approaches and challenges, Journal of information security and applications, 37, pp. 15–27, 2017.

[19] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch, PKCS #1: RSA Cryptography Specifications Version 2.2, RFC 8017, November 2016.

[20] NIST, Data encryption standard (des), FIPS PUB, pp. 46–3, 1999.

[21] NIST, Fips pub 180-4: Secure hash standard (shs), 2011.

[22] NIST, Fips pub 202: Sha-3 standard: Permutation-based hash and extendable-output functions, Technical report, 2015-08-04 00:08:00 2015.

[23] NIST, Module-lattice-based digital signature standard, Technical Report Federal Information Processing Standards Publications (FIPS PUBS) 204, August 13, 2024, U.S. Department of Commerce, Washington, D.C., 2024.

[24] NIST, Module-lattice-based key-encapsulation mechanism standard, Technical Report Federal Information Processing Standards Publications (FIPS PUBS) 203, August 13, 2024, U.S. Department of Commerce, Washington, D.C., 2024.

[25] NIST, M. J. Dworkin, M. S. Turan, and N. Mouha, Fips 197: Advanced encryption standard (aes), 2023-05-09 04:05:00 2023.

[26] C. Paquin, D. Stebila, and G. Tamvada, Benchmarking post-quantum cryptography in tls, in *Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings 11*, pp. 72–91, Springer, 2020.

[27] C. Pavlovski and C. Boyd, Efficient batch signature generation using tree structures, in *International workshop on cryptographic techniques and E-commerce, CrypTEC*, volume 99, pp. 70–77, Citeseer, 1999.

[28] R. L. Probert, On the additive complexity of matrix multiplication, SIAM Journal on Computing, 5(2), pp. 187–203, 1976.

[29] A. Rosowski, Fast commutative matrix algorithms, Journal of Symbolic Computation, 114, pp. 302–321, 2023, ISSN 0747-7171.

[30] P. Schwabe, D. Stebila, and T. Wiggers, Post-quantum tls without handshake signatures, in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1461–1480, 2020.

[31] L. L. Scientific, Enhancing cloud security based on the kyber key encapsulation mechanism, Journal Of Theoretical And Applied Information Technology, 102(4), 2024.

[32] S. Selvakumar, A. Ahilan, B. Ben Sujitha, and N. Muthukumaran, Crystals kyber cryptographic algorithm for efficient iot d2d communication, Wireless Networks, pp. 1–18, 2024.

[33] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM review, 41(2), pp. 303–332, 1999.

[34] D. Sikeridis, P. Kampanakis, and M. Devetsikiotis, Post-quantum authentication in tls 1.3: a performance study, Cryptology ePrint Archive, 2020.

[35] V. Strassen et al., Gaussian elimination is not optimal, Numerische mathematik, 13(4), pp. 354–356, 1969.

[36] S. Tanwar and A. Kumar, An efficient and secure identity based multiple signatures scheme based on rsa, Journal of Discrete Mathematical Sciences and Cryptography, 22(6), pp. 953–971, 2019.

[37] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, Fully homomorphic encryption over the integers, in *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*, pp. 24–43, Springer, 2010.

[38] J. Von Zur Gathen and J. Gerhard, *Modern computer algebra*, Cambridge university press, 2003.

[39] S. Winograd, A new algorithm for inner product, IEEE Transactions on Computers, 100(7), pp. 693–694, 1968.

[40] S. Winograd, On multiplication of 2×2 matrices, Linear Algebra and its Applications, 4(4), pp. 381–388, 1971, ISSN 0024-3795.

APPENDIX A

EFFICIENT MULTIPLICATION FORMULAS FOR DILITHIUM 2 AND KYBER1024

Let $m \geq 4$ messages be signed (or encrypted) with Dilithium 2 (or Kyber1024). Then, $\boldsymbol{A}, \boldsymbol{B} \in R_q^{4\times4}$ and $\boldsymbol{A} \cdot \boldsymbol{B} = \boldsymbol{C} \in R_q^{4\times4}$. The elements of the matrices $\boldsymbol{A}^{4\times4} \cdot \boldsymbol{B}^{4\times4} = \boldsymbol{C}^{4\times4}$ are described as follows:

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}}_{\boldsymbol{B}} = \underbrace{\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}}_{\boldsymbol{C}}$$

A.1 Strassen's Method

To compute the matrix $\boldsymbol{C}$, applying any Strassen-like [35] matrix multiplication method would provide efficiency. To obtain the matrix $\boldsymbol{C}$, firstly, unique multiplications p_i's are computed as follows:

$$\begin{aligned} p_1 &= (a_{11} + a_{33} + a_{22} + a_{44}) \cdot (b_{11} + b_{33} + b_{22} + b_{44}) \\ p_2 &= (a_{21} + a_{43} + a_{22} + a_{44}) \cdot (b_{11} + b_{33}) \\ p_3 &= (a_{11} + a_{33}) \cdot (b_{12} + b_{34} - b_{22} - b_{44}) \\ p_4 &= (a_{22} + a_{44}) \cdot (b_{21} + b_{43} - b_{11} - b_{33}) \end{aligned}$$

$$p_5 = (a_{11} + a_{33} + a_{12} + a_{34}) \cdot (b_{22} + b_{44})$$
$$p_6 = (a_{21} + a_{43} - a_{11} - a_{33}) \cdot (b_{11} + b_{33} + b_{12} + b_{34})$$
$$p_7 = (a_{12} + a_{34} - a_{22} - a_{44}) \cdot (b_{21} + b_{43} + b_{22} + b_{44})$$
$$p_8 = (a_{31} + a_{33} + a_{42} + a_{44}) \cdot (b_{11} + b_{22})$$
$$p_9 = (a_{41} + a_{43} + a_{42} + a_{44}) \cdot (b_{11})$$
$$p_{10} = (a_{31} + a_{33}) \cdot (b_{12} - b_{22})$$
$$p_{11} = (a_{42} + a_{44}) \cdot (b_{21} - b_{11})$$
$$p_{12} = (a_{31} + a_{33} + a_{32} + a_{34}) \cdot (b_{22})$$
$$p_{13} = (a_{41} + a_{43} - a_{31} - a_{33}) \cdot (b_{11} + b_{12})$$
$$p_{14} = (a_{32} + a_{34} - a_{42} - a_{44}) \cdot (b_{21} + b_{22})$$

$$p_{15} = (a_{11} + a_{22}) \cdot (b_{13} - b_{33} + b_{24} - b_{44})$$
$$p_{16} = (a_{21} + a_{22}) \cdot (b_{13} - b_{33})$$
$$p_{17} = (a_{11}) \cdot (b_{14} - b_{34} - b_{24} + b_{44})$$
$$p_{18} = (a_{22}) \cdot (b_{23} - b_{43} - b_{13} + b_{33})$$
$$p_{19} = (a_{11} + a_{12}) \cdot (b_{24} - b_{44})$$
$$p_{20} = (a_{21} - a_{11}) \cdot (b_{13} - b_{33} + b_{14} - b_{34})$$
$$p_{21} = (a_{12} - a_{22}) \cdot (b_{23} - b_{43} + b_{24} - b_{44})$$

$$p_{22} = (a_{33} + a_{44}) \cdot (b_{31} - b_{11} + b_{42} - b_{22})$$
$$p_{23} = (a_{43} + a_{44}) \cdot (b_{31} - b_{11})$$
$$p_{24} = (a_{33}) \cdot (b_{32} - b_{12} - b_{42} + b_{22})$$
$$p_{25} = (a_{44}) \cdot (b_{41} - b_{21} - b_{31} + b_{11})$$
$$p_{26} = (a_{33} + a_{34}) \cdot (b_{42} - b_{22})$$
$$p_{27} = (a_{43} - a_{33}) \cdot (b_{31} - b_{11} + b_{32} - b_{12})$$
$$p_{28} = (a_{34} - a_{44}) \cdot (b_{41} - b_{21} + b_{42} - b_{22})$$

$$p_{29} = (a_{11} + a_{13} + a_{22} + a_{24}) \cdot (b_{33} + b_{44})$$
$$p_{30} = (a_{21} + a_{23} + a_{22} + a_{24}) \cdot (b_{33})$$
$$p_{31} = (a_{11} + a_{13}) \cdot (b_{34} - b_{44})$$
$$p_{32} = (a_{22} + a_{24}) \cdot (b_{43} - b_{33})$$
$$p_{33} = (a_{11} + a_{13} + a_{12} + a_{14}) \cdot (b_{44})$$
$$p_{34} = (a_{21} + a_{23} - a_{11} - a_{13}) \cdot (b_{33} + b_{34})$$
$$p_{35} = (a_{12} + a_{14} - a_{22} - a_{24}) \cdot (b_{43} + b_{44})$$

$$p_{36} = (a_{31} - a_{11} + a_{42} - a_{22}) \cdot (b_{11} + b_{13} + b_{22} + b_{24})$$
$$p_{37} = (a_{41} - a_{21} + a_{42} - a_{22}) \cdot (b_{11} + b_{13})$$
$$p_{38} = (a_{31} - a_{11}) \cdot (b_{12} + b_{14} - b_{22} - b_{24})$$
$$p_{39} = (a_{42} - a_{22}) \cdot (b_{21} + b_{23} - b_{11} - b_{13})$$
$$p_{40} = (a_{31} - a_{11} + a_{32} - a_{12}) \cdot (b_{22} + b_{24})$$
$$p_{41} = (a_{41} - a_{21} - a_{31} + a_{11}) \cdot (b_{11} + b_{13} + b_{12} + b_{14})$$
$$p_{42} = (a_{32} - a_{12} - a_{42} + a_{22}) \cdot (b_{21} + b_{23} + b_{22} + b_{24})$$

$$p_{43} = (a_{13} - a_{33} + a_{24} - a_{44}) \cdot (b_{31} + b_{33} + b_{42} + b_{44})$$
$$p_{44} = (a_{23} - a_{43} + a_{24} - a_{44}) \cdot (b_{31} + b_{33})$$
$$p_{45} = (a_{13} - a_{33}) \cdot (b_{32} + b_{34} - b_{42} - b_{44})$$
$$p_{46} = (a_{24} - a_{44}) \cdot (b_{41} + b_{43} - b_{31} - b_{33})$$
$$p_{47} = (a_{13} - a_{33} + a_{14} - a_{34}) \cdot (b_{42} + b_{44})$$
$$p_{48} = (a_{23} - a_{43} - a_{13} + a_{33}) \cdot (b_{31} + b_{33} + b_{32} + b_{34})$$
$$p_{49} = (a_{14} - a_{34} - a_{24} + a_{44}) \cdot (b_{41} + b_{43} + b_{42} + b_{44})$$

After computing all of the linear combinations, one can obtain the elements of the matrix $\boldsymbol{C}$ as follows:

$$
\begin{aligned}
c_{11} &= p_1 + p_4 - p_5 + p_7 + p_{22} + p_{25} - p_{26} + p_{28} - p_{29} - p_{32} + p_{33} - p_{35} + p_{43} \\
&\quad + p_{46} - p_{47} + p_{49} \\
c_{12} &= p_3 + p_5 + p_{24} + p_{26} - p_{31} - p_{33} + p_{45} + p_{47} \\
c_{13} &= p_{15} + p_{18} - p_{19} + p_{21} + p_{29} + p_{32} - p_{33} + p_{35} \\
c_{14} &= p_{17} + p_{19} + p_{31} + p_{33}
\end{aligned}
$$

$$
\begin{aligned}
c_{21} &= p_2 + p_4 + p_{23} + p_{25} - p_{30} - p_{32} + p_{44} + p_{46} \\
c_{22} &= p_1 - p_2 + p_3 + p_6 + p_{22} - p_{23} + p_{24} + p_{27} - p_{29} + p_{30} - p_{31} - p_{34} + p_{43} \\
&\quad - p_{44} + p_{45} + p_{48} \\
c_{23} &= p_{16} + p_{18} + p_{30} + p_{32} \\
c_{24} &= p_{15} - p_{16} + p_{17} + p_{20} + p_{29} - p_{30} + p_{31} + p_{34}
\end{aligned}
$$

$$
\begin{aligned}
c_{31} &= p_8 + p_{11} - p_{12} + p_{14} + p_{22} + p_{25} - p_{26} + p_{28} \\
c_{32} &= p_{10} + p_{12} + p_{24} + p_{26} \\
c_{33} &= p_1 + p_4 - p_5 + p_7 - p_8 - p_{11} + p_{12} - p_{14} + p_{15} + p_{18} - p_{19} + p_{21} + p_{36} \\
&\quad + p_{39} - p_{40} + p_{42} \\
c_{34} &= p_3 + p_5 - p_{10} - p_{12} + p_{17} + p_{19} + p_{38} + p_{40}
\end{aligned}
$$

$$
\begin{aligned}
c_{41} &= p_9 + p_{11} + p_{23} + p_{25} \\
c_{42} &= p_8 - p_9 + p_{10} + p_{13} + p_{22} - p_{23} + p_{24} + p_{27} \\
c_{43} &= p_2 + p_4 - p_9 - p_{11} + p_{16} + p_{18} + p_{37} + p_{39} \\
c_{44} &= p_1 - p_2 + p_3 + p_6 - p_8 + p_9 - p_{10} - p_{13} + p_{15} - p_{16} + p_{17} + p_{20} + p_{36} \\
&\quad - p_{37} + p_{38} + p_{41}
\end{aligned}
$$

Finally, the matrix $\boldsymbol{C}$ can be obtained via 49 unique multiplications instead of 64. This method saves 15 multiplications. This means a 23.4% improvement in terms of the number of multiplications.

A.2 Fast Commutative Method

Besides the Strassen method, different efficient matrix multiplication methods can also be applied. The elements of the matrices in Dilithium also come from a commutative ring. Therefore, [29] can be used, and the following steps are applied to obtain the matrix $\boldsymbol{C}$.

Firstly, unique multiplications p_i's are obtained as follows:

$$
\begin{aligned}
p_1 &= a_{11} \cdot (b_{11} + a_{12}) & p_{21} &= a_{42} \cdot (b_{21} - a_{41}) \\
p_2 &= a_{13} \cdot (b_{31} + a_{14}) & p_{22} &= a_{44} \cdot (b_{41} - a_{43}) \\
p_3 &= a_{21} \cdot (b_{11} + a_{22}) & p_{23} &= (a_{11} + b_{22}) \cdot (a_{12} + b_{11} + b_{12}) \\
p_4 &= a_{23} \cdot (b_{31} + a_{24}) & p_{25} &= (a_{13} + b_{42}) \cdot (a_{14} + b_{31} + b_{32}) \\
p_5 &= a_{31} \cdot (b_{11} + a_{32}) & p_{24} &= (a_{11} + b_{23}) \cdot (a_{12} + b_{11} + b_{13}) \\
p_6 &= a_{33} \cdot (b_{31} + a_{34}) & p_{26} &= (a_{13} + b_{43}) \cdot (a_{14} + b_{31} + b_{33}) \\
p_7 &= a_{41} \cdot (b_{11} + a_{42}) & p_{27} &= (a_{11} + b_{24}) \cdot (a_{12} + b_{11} + b_{14}) \\
p_8 &= a_{43} \cdot (b_{31} + a_{44}) & p_{28} &= (a_{13} + b_{44}) \cdot (a_{14} + b_{31} + b_{34}) \\
p_9 &= b_{22} \cdot (b_{11} + b_{12}) & p_{29} &= (a_{21} + b_{22}) \cdot (a_{22} + b_{11} + b_{12}) \\
p_{10} &= b_{42} \cdot (b_{31} + b_{32}) & p_{30} &= (a_{23} + b_{42}) \cdot (a_{24} + b_{31} + b_{32}) \\
p_{11} &= b_{23} \cdot (b_{11} + b_{13}) & p_{31} &= (a_{21} + b_{23}) \cdot (a_{22} + b_{11} + b_{13}) \\
p_{12} &= b_{43} \cdot (b_{31} + b_{33}) & p_{32} &= (a_{23} + b_{43}) \cdot (a_{24} + b_{31} + b_{33}) \\
p_{13} &= b_{24} \cdot (b_{11} + b_{14}) & p_{33} &= (a_{21} + b_{24}) \cdot (a_{22} + b_{11} + b_{14}) \\
p_{14} &= b_{44} \cdot (b_{31} + b_{34}) & p_{34} &= (a_{23} + b_{44}) \cdot (a_{24} + b_{31} + b_{34}) \\
p_{15} &= a_{12} \cdot (b_{21} - a_{11}) & p_{35} &= (a_{31} + b_{22}) \cdot (a_{32} + b_{11} + b_{12}) \\
p_{16} &= a_{14} \cdot (b_{41} - a_{13}) & p_{36} &= (a_{33} + b_{42}) \cdot (a_{34} + b_{31} + b_{32}) \\
p_{17} &= a_{22} \cdot (b_{21} - a_{21}) & p_{37} &= (a_{31} + b_{23}) \cdot (a_{32} + b_{11} + b_{13}) \\
p_{18} &= a_{24} \cdot (b_{41} - a_{23}) & p_{38} &= (a_{33} + b_{43}) \cdot (a_{34} + b_{31} + b_{33}) \\
p_{19} &= a_{32} \cdot (b_{21} - a_{31}) & p_{39} &= (a_{31} + b_{24}) \cdot (a_{32} + b_{11} + b_{14}) \\
p_{20} &= a_{34} \cdot (b_{41} - a_{33}) & p_{40} &= (a_{33} + b_{44}) \cdot (a_{34} + b_{31} + b_{34})
\end{aligned}
$$

$$
\begin{aligned}
p_{41} &= (a_{41} + b_{22}) \cdot (a_{42} + b_{11} + b_{12}) \\
p_{42} &= (a_{43} + b_{42}) \cdot (a_{44} + b_{31} + b_{32}) \\
p_{43} &= (a_{41} + b_{23}) \cdot (a_{42} + b_{11} + b_{13}) \\
p_{44} &= (a_{43} + b_{43}) \cdot (a_{44} + b_{31} + b_{33}) \\
p_{45} &= (a_{41} + b_{24}) \cdot (a_{42} + b_{11} + b_{14}) \\
p_{46} &= (a_{43} + b_{44}) \cdot (a_{44} + b_{31} + b_{34})
\end{aligned}
$$

After computing all of the linear combinations, one can obtain the elements of the matrix $\boldsymbol{C}$ as follows:

$$
\begin{aligned}
c_{11} &= p_1 + p_2 + p_{15} + p_{16} & \qquad c_{31} &= p_5 + p_6 + p_{19} + p_{20} \\
c_{12} &= p_{23} + p_{24} - p_1 - p_2 - p_9 - p_{10} & \qquad c_{32} &= p_{35} + p_{36} - p_5 - p_6 - p_9 - p_{10} \\
c_{13} &= p_{25} + p_{26} - p_1 - p_2 - p_{11} - p_{12} & \qquad c_{33} &= p_{37} + p_{38} - p_5 - p_6 - p_{11} - p_{12} \\
c_{14} &= p_{27} + p_{28} - p_1 - p_2 - p_{13} - p_{14} & \qquad c_{34} &= p_{39} + p_{40} - p_5 - p_6 - p_{13} - p_{14}
\end{aligned}
$$

$$
\begin{aligned}
c_{21} &= p_3 + p_4 + p_{17} + p_{18} & \qquad c_{41} &= p_7 + p_8 + p_{21} + p_{22} \\
c_{22} &= p_{29} + p_{30} - p_3 - p_4 - p_9 - p_{10} & \qquad c_{42} &= p_{41} + p_{42} - p_7 - p_8 - p_9 - p_{10} \\
c_{23} &= p_{31} + p_{32} - p_3 - p_4 - p_{11} - p_{12} & \qquad c_{43} &= p_{43} + p_{44} - p_7 - p_8 - p_{11} - p_{12} \\
c_{24} &= p_{33} + p_{34} - p_3 - p_4 - p_{13} - p_{14} & \qquad c_{44} &= p_{45} + p_{46} - p_7 - p_8 - p_{13} - p_{14}
\end{aligned}
$$

Finally, the matrix $\boldsymbol{C}$ can be obtained via 46 unique multiplications instead of 64. This method saves 18 multiplications. This means a 28.1% improvement in terms of the number of multiplications.

APPENDIX B

EFFICIENT MULTIPLICATION FORMULA FOR DILITHIUM 3

Let $m \geq 5$ messages are needed to be signed with Dilithium 3. Then, $\boldsymbol{A} \in R_q^{6\times 5}$, $\boldsymbol{B} \in R_q^{5\times 5}$ and $\boldsymbol{A}\cdot\boldsymbol{B} = \boldsymbol{C} \in R_q^{6\times 5}$. The elements of the matrices $\boldsymbol{A}^{6\times 5}\cdot\boldsymbol{B}^{5\times 5} = \boldsymbol{C}^{6\times 5}$ are described as follows:

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \end{bmatrix}}_{\boldsymbol{B}} = \underbrace{\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} \\ c_{51} & c_{52} & c_{53} & c_{54} & c_{55} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} \end{bmatrix}}_{\boldsymbol{C}}$$

By [29], the matrix $\boldsymbol{C}$ can be obtained as follows:

$$p_1 = (a_{11} + b_{21}) \cdot (a_{12} + b_{12}) \qquad p_9 = (a_{31} + b_{31}) \cdot (a_{33} + b_{13})$$
$$p_2 = (a_{21} + b_{21}) \cdot (a_{22} + b_{12}) \qquad p_{10} = (a_{41} + b_{31}) \cdot (a_{43} + b_{13})$$
$$p_3 = (a_{31} + b_{21}) \cdot (a_{32} + b_{12}) \qquad p_{11} = (a_{51} + b_{31}) \cdot (a_{53} + b_{13})$$
$$p_4 = (a_{41} + b_{21}) \cdot (a_{42} + b_{12}) \qquad p_{12} = (a_{61} + b_{31}) \cdot (a_{63} + b_{13})$$
$$p_5 = (a_{51} + b_{21}) \cdot (a_{52} + b_{12}) \qquad p_{13} = (a_{12} + b_{32}) \cdot (a_{13} + b_{23})$$
$$p_6 = (a_{61} + b_{21}) \cdot (a_{62} + b_{12}) \qquad p_{14} = (a_{22} + b_{32}) \cdot (a_{23} + b_{23})$$
$$p_7 = (a_{11} + b_{31}) \cdot (a_{13} + b_{13}) \qquad p_{15} = (a_{32} + b_{32}) \cdot (a_{33} + b_{23})$$
$$p_8 = (a_{21} + b_{31}) \cdot (a_{23} + b_{13}) \qquad p_{16} = (a_{42} + b_{32}) \cdot (a_{43} + b_{23})$$

$$p_{17} = (a_{52} + b_{32}) \cdot (a_{53} + b_{23})$$

$$p_{18} = (a_{62} + b_{32}) \cdot (a_{63} + b_{23})$$

$$p_{19} = a_{11} \cdot (b_{11} - b_{12} - b_{13} - a_{12} - a_{13})$$

$$p_{20} = a_{21} \cdot (b_{11} - b_{12} - b_{13} - a_{22} - a_{23})$$

$$p_{21} = a_{31} \cdot (b_{11} - b_{12} - b_{13} - a_{32} - a_{33})$$

$$p_{22} = a_{41} \cdot (b_{11} - b_{12} - b_{13} - a_{42} - a_{43})$$

$$p_{23} = a_{51} \cdot (b_{11} - b_{12} - b_{13} - a_{52} - a_{53})$$

$$p_{24} = a_{61} \cdot (b_{11} - b_{12} - b_{13} - a_{62} - a_{63})$$

$$p_{25} = a_{12} \cdot (b_{22} - b_{21} - b_{23} - a_{11} - a_{13})$$

$$p_{26} = a_{22} \cdot (b_{22} - b_{21} - b_{23} - a_{21} - a_{23})$$

$$p_{27} = a_{32} \cdot (b_{22} - b_{21} - b_{23} - a_{31} - a_{33})$$

$$p_{28} = a_{42} \cdot (b_{22} - b_{21} - b_{23} - a_{41} - a_{43})$$

$$p_{29} = a_{52} \cdot (b_{22} - b_{21} - b_{23} - a_{51} - a_{53})$$

$$p_{30} = a_{62} \cdot (b_{22} - b_{21} - b_{23} - a_{61} - a_{63})$$

$$p_{31} = a_{13} \cdot (b_{33} - b_{31} - b_{32} - a_{11} - a_{12})$$

$$p_{32} = a_{23} \cdot (b_{33} - b_{31} - b_{32} - a_{21} - a_{22})$$

$$p_{33} = a_{33} \cdot (b_{33} - b_{31} - b_{32} - a_{31} - a_{32})$$

$$p_{34} = a_{43} \cdot (b_{33} - b_{31} - b_{32} - a_{41} - a_{42})$$

$$p_{35} = a_{53} \cdot (b_{33} - b_{31} - b_{32} - a_{51} - a_{52})$$

$$p_{36} = a_{63} \cdot (b_{33} - b_{31} - b_{32} - a_{61} - a_{62})$$

$$p_{37} = b_{12} \cdot b_{21}$$

$$p_{38} = b_{13} \cdot b_{31}$$

$$p_{39} = b_{23} \cdot b_{32}$$

$$p_{40} = (a_{11} + b_{21} - b_{24}) \cdot (-a_{12} - b_{12} + b_{14} - b_{15})$$

$$p_{41} = (a_{21} + b_{21} - b_{24}) \cdot (-a_{22} - b_{12} + b_{14} - b_{15})$$

$$p_{42} = (a_{31} + b_{21} - b_{24}) \cdot (-a_{32} - b_{12} + b_{14} - b_{15})$$

$$p_{43} = (a_{41} + b_{21} - b_{24}) \cdot (-a_{42} - b_{12} + b_{14} - b_{15})$$

$$p_{44} = (a_{51} + b_{21} - b_{24}) \cdot (-a_{52} - b_{12} + b_{14} - b_{15})$$

$$p_{45} = (a_{61} + b_{21} - b_{24}) \cdot (-a_{62} - b_{12} + b_{14} - b_{15})$$

$$p_{46} = (a_{12} + b_{32} + b_{34} - b_{35}) \cdot (-a_{13} - b_{23} + b_{25})$$

$$p_{47} = (a_{22} + b_{32} + b_{34} - b_{35}) \cdot (-a_{23} - b_{23} + b_{25})$$

$$p_{48} = (a_{32} + b_{32} + b_{34} - b_{35}) \cdot (-a_{33} - b_{23} + b_{25})$$

$$p_{49} = (a_{42} + b_{32} + b_{34} - b_{35}) \cdot (-a_{43} - b_{23} + b_{25})$$

$$p_{50} = (a_{52} + b_{32} + b_{34} - b_{35}) \cdot (-a_{53} - b_{23} + b_{25})$$

$$p_{51} = (a_{62} + b_{32} + b_{34} - b_{35}) \cdot (-a_{63} - b_{23} + b_{25})$$

$$p_{52} = (a_{11} + b_{31} - b_{34}) \cdot (-a_{13} - b_{13} + b_{15})$$

$$p_{53} = (a_{21} + b_{31} - b_{34}) \cdot (-a_{23} - b_{13} + b_{15})$$

$$p_{54} = (a_{31} + b_{31} - b_{34}) \cdot (-a_{33} - b_{13} + b_{15})$$

$$p_{55} = (a_{41} + b_{31} - b_{34}) \cdot (-a_{43} - b_{13} + b_{15})$$

$$p_{56} = (a_{51} + b_{31} - b_{34}) \cdot (-a_{53} - b_{13} + b_{15})$$

$$p_{57} = (a_{61} + b_{31} - b_{34}) \cdot (-a_{63} - b_{13} + b_{15})$$

$$p_{58} = (b_{21} - b_{24}) \cdot (-b_{12} + b_{14} - b_{15})$$

$$p_{59} = (b_{31} - b_{34}) \cdot (-b_{13} + b_{15})$$

$$p_{60} = (b_{32} + b_{34} - b_{35}) \cdot (-b_{23} + b_{25})$$

$$p_{61} = a_{14} \cdot (b_{41} + a_{15})$$

$$p_{62} = a_{24} \cdot (b_{41} + a_{25})$$

$$p_{63} = a_{34} \cdot (b_{41} + a_{35})$$

$$p_{64} = a_{44} \cdot (b_{41} + a_{45})$$

$$p_{65} = a_{54} \cdot (b_{41} + a_{55})$$

$$p_{66} = a_{64} \cdot (b_{41} + a_{65})$$

$$p_{67} = a_{15} \cdot (b_{51} - a_{14})$$

$$p_{68} = a_{25} \cdot (b_{51} - a_{24})$$

$$p_{69} = a_{35} \cdot (b_{51} - a_{34})$$

$$p_{70} = a_{45} \cdot (b_{51} - a_{44})$$

$$p_{71} = a_{55} \cdot (b_{51} - a_{54})$$

$$p_{72} = a_{65} \cdot (b_{51} - a_{64})$$

$$p_{73} = (a_{14} + b_{52}) \cdot (a_{15} + b_{41} + b_{42})$$

$$p_{74} = (a_{24} + b_{52}) \cdot (a_{25} + b_{41} + b_{42})$$

$$p_{75} = (a_{34} + b_{52}) \cdot (a_{35} + b_{41} + b_{42})$$

$$p_{76} = (a_{44} + b_{52}) \cdot (a_{45} + b_{41} + b_{42})$$

$$p_{77} = (a_{54} + b_{52}) \cdot (a_{55} + b_{41} + b_{42})$$

$$p_{78} = (a_{64} + b_{52}) \cdot (a_{65} + b_{41} + b_{42})$$

$$p_{79} = (a_{14} + b_{53}) \cdot (a_{15} + b_{41} + b_{43})$$

$$p_{80} = (a_{24} + b_{53}) \cdot (a_{25} + b_{41} + b_{43})$$

$$p_{81} = (a_{34} + b_{53}) \cdot (a_{35} + b_{41} + b_{43})$$

$$p_{82} = (a_{44} + b_{53}) \cdot (a_{45} + b_{41} + b_{43})$$

$$p_{83} = (a_{54} + b_{53}) \cdot (a_{55} + b_{41} + b_{43})$$

$$p_{84} = (a_{64} + b_{53}) \cdot (a_{65} + b_{41} + b_{43})$$

$$p_{85} = (a_{14} + b_{54}) \cdot (a_{15} + b_{41} + b_{44})$$

$$p_{86} = (a_{24} + b_{54}) \cdot (a_{25} + b_{41} + b_{44})$$

$$p_{87} = (a_{34} + b_{54}) \cdot (a_{35} + b_{41} + b_{44})$$

$$p_{88} = (a_{44} + b_{54}) \cdot (a_{45} + b_{41} + b_{44})$$

$$p_{89} = (a_{54} + b_{54}) \cdot (a_{55} + b_{41} + b_{44})$$

$$p_{90} = (a_{64} + b_{54}) \cdot (a_{65} + b_{41} + b_{44})$$

$$p_{91} = (a_{14} + b_{55}) \cdot (a_{15} + b_{41} + b_{45})$$

$$p_{92} = (a_{24} + b_{55}) \cdot (a_{25} + b_{41} + b_{45})$$

$$p_{93} = (a_{34} + b_{55}) \cdot (a_{35} + b_{41} + b_{45})$$

$$p_{94} = (a_{44} + b_{55}) \cdot (a_{45} + b_{41} + b_{45})$$

$$p_{95} = (a_{54} + b_{55}) \cdot (a_{55} + b_{41} + b_{45})$$

$$p_{96} = (a_{64} + b_{55}) \cdot (a_{65} + b_{41} + b_{45})$$

$$p_{97} = b_{52} \cdot (b_{41} + b_{42})$$

$$p_{98} = b_{53} \cdot (b_{41} + b_{43})$$

$$p_{99} = b_{54} \cdot (b_{41} + b_{44})$$

$$p_{100} = b_{55} \cdot (b_{41} + b_{45})$$

After computing all of the linear combinations, one can obtain the elements of the matrix $\boldsymbol{C}$ as follows:

$$c_{11} = p_1 + p_7 + p_{19} - p_{37} - p_{38} + p_{61} + p_{67}$$
$$c_{21} = p_2 + p_8 + p_{20} - p_{37} - p_{38} + p_{62} + p_{68}$$
$$c_{31} = p_3 + p_9 + p_{21} - p_{37} - p_{38} + p_{63} + p_{69}$$
$$c_{41} = p_4 + p_{10} + p_{22} - p_{37} - p_{38} + p_{64} + p_{70}$$
$$c_{51} = p_5 + p_{11} + p_{23} - p_{37} - p_{38} + p_{65} + p_{71}$$
$$c_{61} = p_6 + p_{12} + p_{24} - p_{37} - p_{38} + p_{66} + p_{72}$$

$$c_{12} = p_1 + p_{13} + p_{25} - p_{37} - p_{39} + p_{73} - p_{61} - p_{97}$$
$$c_{22} = p_2 + p_{14} + p_{26} - p_{37} - p_{39} + p_{74} - p_{62} - p_{97}$$
$$c_{32} = p_3 + p_{15} + p_{27} - p_{37} - p_{39} + p_{75} - p_{63} - p_{97}$$
$$c_{42} = p_4 + p_{16} + p_{28} - p_{37} - p_{39} + p_{76} - p_{64} - p_{97}$$
$$c_{52} = p_5 + p_{17} + p_{29} - p_{37} - p_{39} + p_{77} - p_{65} - p_{97}$$
$$c_{62} = p_6 + p_{18} + p_{30} - p_{37} - p_{39} + p_{78} - p_{66} - p_{97}$$

$$c_{13} = p_7 + p_{13} + p_{31} - p_{38} - p_{39} + p_{79} - p_{61} - p_{98}$$
$$c_{23} = p_8 + p_{14} + p_{32} - p_{38} - p_{39} + p_{80} - p_{62} - p_{98}$$
$$c_{33} = p_9 + p_{15} + p_{33} - p_{38} - p_{39} + p_{81} - p_{63} - p_{98}$$
$$c_{43} = p_{10} + p_{16} + p_{34} - p_{38} - p_{39} + p_{82} - p_{64} - p_{98}$$
$$c_{53} = p_{11} + p_{17} + p_{35} - p_{38} - p_{39} + p_{83} - p_{65} - p_{98}$$
$$c_{63} = p_{12} + p_{18} + p_{36} - p_{38} - p_{39} + p_{84} - p_{66} - p_{98}$$

$$c_{14} = p_1 + p_7 + p_{40} + p_{52} - p_{37} - p_{38} - p_{58} - p_{59} + p_{85} - p_{61} - p_{99}$$

$$c_{24} = p_2 + p_8 + p_{41} + p_{53} - p_{37} - p_{38} - p_{58} - p_{59} + p_{86} - p_{62} - p_{99}$$

$$c_{34} = p_3 + p_9 + p_{42} + p_{54} - p_{37} - p_{38} - p_{58} - p_{59} + p_{87} - p_{63} - p_{99}$$

$$c_{44} = p_4 + p_{10} + p_{43} + p_{55} - p_{37} - p_{38} - p_{58} - p_{59} + p_{88} - p_{64} - p_{99}$$

$$c_{54} = p_5 + p_{11} + p_{44} + p_{56} - p_{37} - p_{38} - p_{58} - p_{59} + p_{89} - p_{65} - p_{99}$$

$$c_{64} = p_6 + p_{12} + p_{45} + p_{57} - p_{37} - p_{38} - p_{58} - p_{59} + p_{90} - p_{66} - p_{99}$$

$$c_{15} = p_7 + p_{13} + p_{46} + p_{52} - p_{38} - p_{39} - p_{59} - p_{60} + p_{91} - p_{61} - p_{100}$$

$$c_{25} = p_8 + p_{14} + p_{47} + p_{53} - p_{38} - p_{39} - p_{59} - p_{60} + p_{92} - p_{62} - p_{100}$$

$$c_{35} = p_9 + p_{15} + p_{48} + p_{54} - p_{38} - p_{39} - p_{59} - p_{60} + p_{93} - p_{63} - p_{100}$$

$$c_{45} = p_{10} + p_{16} + p_{49} + p_{55} - p_{38} - p_{39} - p_{59} - p_{60} + p_{94} - p_{64} - p_{100}$$

$$c_{55} = p_{11} + p_{17} + p_{50} + p_{56} - p_{38} - p_{39} - p_{59} - p_{60} + p_{95} - p_{65} - p_{100}$$

$$c_{65} = p_{12} + p_{18} + p_{51} + p_{57} - p_{38} - p_{39} - p_{59} - p_{60} + p_{96} - p_{66} - p_{100}$$

APPENDIX C

EFFICIENT MULTIPLICATION FORMULA FOR DILITHIUM 5

Let $m \geq 4$ messages be signed with Dilithium 5. Then, $\boldsymbol{A} \in R_q^{8\times 7}$, $\boldsymbol{B} \in R_q^{7\times 4}$ and $\boldsymbol{A}\cdot\boldsymbol{B} = \boldsymbol{C} \in R_q^{8\times 4}$. The elements of the matrices $\boldsymbol{A}^{8\times 7}\cdot\boldsymbol{B}^{7\times 4} = \boldsymbol{C}^{8\times 4}$ are described as follows:

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} & a_{86} & a_{87} \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \\ b_{51} & b_{52} & b_{53} & b_{54} \\ b_{61} & b_{62} & b_{63} & b_{64} \\ b_{71} & b_{72} & b_{73} & b_{74} \end{bmatrix}}_{\boldsymbol{B}} = \underbrace{\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \\ c_{51} & c_{52} & c_{53} & c_{54} \\ c_{61} & c_{62} & c_{63} & c_{64} \\ c_{71} & c_{72} & c_{73} & c_{74} \\ c_{81} & c_{82} & c_{83} & c_{84} \end{bmatrix}}_{\boldsymbol{C}}$$

By [29], the matrix $\boldsymbol{C}$ can be obtained as follows:

$$p_1 = (a_{11} + b_{21}) \cdot (a_{12} + b_{12}) \qquad p_6 = (a_{61} + b_{21}) \cdot (a_{62} + b_{12})$$
$$p_2 = (a_{21} + b_{21}) \cdot (a_{22} + b_{12}) \qquad p_7 = (a_{71} + b_{21}) \cdot (a_{72} + b_{12})$$
$$p_3 = (a_{31} + b_{21}) \cdot (a_{32} + b_{12}) \qquad p_8 = (a_{81} + b_{21}) \cdot (a_{82} + b_{12})$$
$$p_4 = (a_{41} + b_{21}) \cdot (a_{42} + b_{12}) \qquad p_9 = (a_{11} + b_{31}) \cdot (a_{13} + b_{13})$$
$$p_5 = (a_{51} + b_{21}) \cdot (a_{52} + b_{12}) \qquad p_{10} = (a_{21} + b_{31}) \cdot (a_{23} + b_{13})$$

$$p_{11} = (a_{31} + b_{31}) \cdot (a_{33} + b_{13})$$

$$p_{12} = (a_{41} + b_{31}) \cdot (a_{43} + b_{13})$$

$$p_{13} = (a_{51} + b_{31}) \cdot (a_{53} + b_{13})$$

$$p_{14} = (a_{61} + b_{31}) \cdot (a_{63} + b_{13})$$

$$p_{15} = (a_{71} + b_{31}) \cdot (a_{73} + b_{13})$$

$$p_{16} = (a_{81} + b_{31}) \cdot (a_{83} + b_{13})$$

$$p_{17} = (a_{12} + b_{32}) \cdot (a_{13} + b_{23})$$

$$p_{18} = (a_{22} + b_{32}) \cdot (a_{23} + b_{23})$$

$$p_{19} = (a_{32} + b_{32}) \cdot (a_{33} + b_{23})$$

$$p_{20} = (a_{42} + b_{32}) \cdot (a_{43} + b_{23})$$

$$p_{21} = (a_{52} + b_{32}) \cdot (a_{53} + b_{23})$$

$$p_{22} = (a_{62} + b_{32}) \cdot (a_{63} + b_{23})$$

$$p_{23} = (a_{72} + b_{32}) \cdot (a_{73} + b_{23})$$

$$p_{24} = (a_{82} + b_{32}) \cdot (a_{83} + b_{23})$$

$$p_{25} = b_{12} \cdot b_{21}$$

$$p_{26} = b_{13} \cdot b_{31}$$

$$p_{27} = b_{23} \cdot b_{32}$$

$$p_{28} = (b_{21} - b_{24}) \cdot (-b_{12} + b_{14})$$

$$p_{29} = a_{11} \cdot (b_{11} - b_{12} - b_{13} - a_{12} - a_{13})$$

$$p_{30} = a_{21} \cdot (b_{11} - b_{12} - b_{13} - a_{22} - a_{23})$$

$$p_{31} = a_{31} \cdot (b_{11} - b_{12} - b_{13} - a_{32} - a_{33})$$

$$p_{32} = a_{41} \cdot (b_{11} - b_{12} - b_{13} - a_{42} - a_{43})$$

$$p_{33} = a_{51} \cdot (b_{11} - b_{12} - b_{13} - a_{52} - a_{53})$$

$$p_{34} = a_{61} \cdot (b_{11} - b_{12} - b_{13} - a_{62} - a_{63})$$

$$p_{35} = a_{71} \cdot (b_{11} - b_{12} - b_{13} - a_{72} - a_{73})$$

$$p_{36} = a_{81} \cdot (b_{11} - b_{12} - b_{13} - a_{82} - a_{83})$$

$$p_{37} = a_{12} \cdot (b_{22} - b_{21} - b_{23} - a_{11} - a_{13})$$

$$p_{38} = a_{22} \cdot (b_{22} - b_{21} - b_{23} - a_{21} - a_{23})$$

$$p_{39} = a_{32} \cdot (b_{22} - b_{21} - b_{23} - a_{31} - a_{33})$$
$$p_{40} = a_{42} \cdot (b_{22} - b_{21} - b_{23} - a_{41} - a_{43})$$
$$p_{41} = a_{52} \cdot (b_{22} - b_{21} - b_{23} - a_{51} - a_{53})$$
$$p_{42} = a_{62} \cdot (b_{22} - b_{21} - b_{23} - a_{61} - a_{63})$$
$$p_{43} = a_{72} \cdot (b_{22} - b_{21} - b_{23} - a_{71} - a_{73})$$
$$p_{44} = a_{82} \cdot (b_{22} - b_{21} - b_{23} - a_{81} - a_{83})$$
$$p_{45} = a_{13} \cdot (b_{33} - b_{31} - b_{32} - a_{11} - a_{12})$$
$$p_{46} = a_{23} \cdot (b_{33} - b_{31} - b_{32} - a_{21} - a_{22})$$
$$p_{47} = a_{33} \cdot (b_{33} - b_{31} - b_{32} - a_{31} - a_{32})$$
$$p_{48} = a_{43} \cdot (b_{33} - b_{31} - b_{32} - a_{41} - a_{42})$$
$$p_{49} = a_{53} \cdot (b_{33} - b_{31} - b_{32} - a_{51} - a_{52})$$
$$p_{50} = a_{63} \cdot (b_{33} - b_{31} - b_{32} - a_{61} - a_{62})$$
$$p_{51} = a_{73} \cdot (b_{33} - b_{31} - b_{32} - a_{71} - a_{72})$$
$$p_{52} = a_{83} \cdot (b_{33} - b_{31} - b_{32} - a_{81} - a_{82})$$
$$p_{53} = (a_{11} + b_{21} - b_{24}) \cdot (-a_{12} - b_{12} + b_{14})$$
$$p_{54} = (a_{21} + b_{21} - b_{24}) \cdot (-a_{22} - b_{12} + b_{14})$$
$$p_{55} = (a_{31} + b_{21} - b_{24}) \cdot (-a_{32} - b_{12} + b_{14})$$
$$p_{56} = (a_{41} + b_{21} - b_{24}) \cdot (-a_{42} - b_{12} + b_{14})$$
$$p_{57} = (a_{51} + b_{21} - b_{24}) \cdot (-a_{52} - b_{12} + b_{14})$$
$$p_{58} = (a_{61} + b_{21} - b_{24}) \cdot (-a_{62} - b_{12} + b_{14})$$
$$p_{59} = (a_{71} + b_{21} - b_{24}) \cdot (-a_{72} - b_{12} + b_{14})$$
$$p_{60} = (a_{81} + b_{21} - b_{24}) \cdot (-a_{82} - b_{12} + b_{14})$$
$$p_{61} = a_{13} \cdot b_{34}$$
$$p_{62} = a_{23} \cdot b_{34}$$
$$p_{63} = a_{33} \cdot b_{34}$$
$$p_{64} = a_{43} \cdot b_{34}$$
$$p_{65} = a_{53} \cdot b_{34}$$
$$p_{66} = a_{63} \cdot b_{34}$$
$$p_{67} = a_{73} \cdot b_{34}$$
$$p_{68} = a_{83} \cdot b_{34}$$
$$p_{69} = a_{14} \cdot (b_{41} + a_{15})$$
$$p_{70} = a_{24} \cdot (b_{41} + a_{25})$$
$$p_{71} = a_{34} \cdot (b_{41} + a_{35})$$
$$p_{72} = a_{44} \cdot (b_{41} + a_{45})$$
$$p_{73} = a_{54} \cdot (b_{41} + a_{55})$$
$$p_{74} = a_{64} \cdot (b_{41} + a_{65})$$
$$p_{75} = a_{74} \cdot (b_{41} + a_{75})$$
$$p_{76} = a_{84} \cdot (b_{41} + a_{85})$$
$$p_{77} = a_{16} \cdot (b_{61} + a_{17})$$
$$p_{78} = a_{26} \cdot (b_{61} + a_{27})$$
$$p_{79} = a_{36} \cdot (b_{61} + a_{37})$$
$$p_{80} = a_{46} \cdot (b_{61} + a_{47})$$
$$p_{81} = a_{56} \cdot (b_{61} + a_{57})$$
$$p_{82} = a_{66} \cdot (b_{61} + a_{67})$$
$$p_{83} = a_{76} \cdot (b_{61} + a_{77})$$
$$p_{84} = a_{86} \cdot (b_{61} + a_{87})$$
$$p_{85} = b_{52} \cdot (b_{41} + b_{42})$$
$$p_{86} = b_{72} \cdot (b_{61} + b_{62})$$
$$p_{87} = b_{53} \cdot (b_{41} + b_{43})$$
$$p_{88} = b_{73} \cdot (b_{61} + b_{63})$$
$$p_{89} = b_{54} \cdot (b_{41} + b_{44})$$
$$p_{90} = b_{74} \cdot (b_{61} + b_{64})$$
$$p_{91} = a_{15} \cdot (b_{51} - a_{14})$$
$$p_{92} = a_{25} \cdot (b_{51} - a_{24})$$
$$p_{93} = a_{35} \cdot (b_{51} - a_{34})$$
$$p_{94} = a_{45} \cdot (b_{51} - a_{44})$$

$$p_{95} = a_{55} \cdot (b_{51} - a_{54})$$
$$p_{96} = a_{65} \cdot (b_{51} - a_{64})$$
$$p_{97} = a_{75} \cdot (b_{51} - a_{74})$$
$$p_{98} = a_{85} \cdot (b_{51} - a_{84})$$
$$p_{99} = a_{17} \cdot (b_{71} - a_{16})$$
$$p_{100} = a_{27} \cdot (b_{71} - a_{26})$$
$$p_{101} = a_{37} \cdot (b_{71} - a_{36})$$
$$p_{102} = a_{47} \cdot (b_{71} - a_{46})$$
$$p_{103} = a_{57} \cdot (b_{71} - a_{56})$$
$$p_{104} = a_{67} \cdot (b_{71} - a_{66})$$
$$p_{105} = a_{77} \cdot (b_{71} - a_{76})$$
$$p_{106} = a_{87} \cdot (b_{71} - a_{86})$$
$$p_{107} = (a_{14} + b_{52}) \cdot (a_{15} + b_{41} + b_{42})$$
$$p_{108} = (a_{24} + b_{52}) \cdot (a_{25} + b_{41} + b_{42})$$
$$p_{109} = (a_{34} + b_{52}) \cdot (a_{35} + b_{41} + b_{42})$$
$$p_{110} = (a_{44} + b_{52}) \cdot (a_{45} + b_{41} + b_{42})$$
$$p_{111} = (a_{54} + b_{52}) \cdot (a_{55} + b_{41} + b_{42})$$
$$p_{112} = (a_{64} + b_{52}) \cdot (a_{65} + b_{41} + b_{42})$$
$$p_{113} = (a_{74} + b_{52}) \cdot (a_{75} + b_{41} + b_{42})$$
$$p_{114} = (a_{84} + b_{52}) \cdot (a_{85} + b_{41} + b_{42})$$
$$p_{115} = (a_{16} + b_{72}) \cdot (a_{17} + b_{61} + b_{62})$$
$$p_{116} = (a_{26} + b_{72}) \cdot (a_{27} + b_{61} + b_{62})$$
$$p_{117} = (a_{36} + b_{72}) \cdot (a_{37} + b_{61} + b_{62})$$
$$p_{118} = (a_{46} + b_{72}) \cdot (a_{47} + b_{61} + b_{62})$$
$$p_{119} = (a_{56} + b_{72}) \cdot (a_{57} + b_{61} + b_{62})$$
$$p_{120} = (a_{66} + b_{72}) \cdot (a_{67} + b_{61} + b_{62})$$
$$p_{121} = (a_{76} + b_{72}) \cdot (a_{77} + b_{61} + b_{62})$$
$$p_{122} = (a_{86} + b_{72}) \cdot (a_{87} + b_{61} + b_{62})$$
$$p_{123} = (a_{14} + b_{53}) \cdot (a_{15} + b_{41} + b_{43})$$
$$p_{124} = (a_{24} + b_{53}) \cdot (a_{25} + b_{41} + b_{43})$$
$$p_{125} = (a_{34} + b_{53}) \cdot (a_{35} + b_{41} + b_{43})$$
$$p_{126} = (a_{44} + b_{53}) \cdot (a_{45} + b_{41} + b_{43})$$
$$p_{127} = (a_{54} + b_{53}) \cdot (a_{55} + b_{41} + b_{43})$$
$$p_{128} = (a_{64} + b_{53}) \cdot (a_{65} + b_{41} + b_{43})$$
$$p_{129} = (a_{74} + b_{53}) \cdot (a_{75} + b_{41} + b_{43})$$
$$p_{130} = (a_{84} + b_{53}) \cdot (a_{85} + b_{41} + b_{43})$$
$$p_{131} = (a_{16} + b_{73}) \cdot (a_{17} + b_{61} + b_{63})$$
$$p_{132} = (a_{26} + b_{73}) \cdot (a_{27} + b_{61} + b_{63})$$
$$p_{133} = (a_{36} + b_{73}) \cdot (a_{37} + b_{61} + b_{63})$$
$$p_{134} = (a_{46} + b_{73}) \cdot (a_{47} + b_{61} + b_{63})$$
$$p_{135} = (a_{56} + b_{73}) \cdot (a_{57} + b_{61} + b_{63})$$
$$p_{136} = (a_{66} + b_{73}) \cdot (a_{67} + b_{61} + b_{63})$$
$$p_{137} = (a_{76} + b_{73}) \cdot (a_{77} + b_{61} + b_{63})$$
$$p_{138} = (a_{86} + b_{73}) \cdot (a_{87} + b_{61} + b_{63})$$
$$p_{139} = (a_{14} + b_{54}) \cdot (a_{15} + b_{41} + b_{44})$$
$$p_{140} = (a_{24} + b_{54}) \cdot (a_{25} + b_{41} + b_{44})$$
$$p_{141} = (a_{34} + b_{54}) \cdot (a_{35} + b_{41} + b_{44})$$
$$p_{142} = (a_{44} + b_{54}) \cdot (a_{45} + b_{41} + b_{44})$$
$$p_{143} = (a_{54} + b_{54}) \cdot (a_{55} + b_{41} + b_{44})$$
$$p_{144} = (a_{64} + b_{54}) \cdot (a_{65} + b_{41} + b_{44})$$
$$p_{145} = (a_{74} + b_{54}) \cdot (a_{75} + b_{41} + b_{44})$$
$$p_{146} = (a_{84} + b_{54}) \cdot (a_{85} + b_{41} + b_{44})$$
$$p_{147} = (a_{16} + b_{74}) \cdot (a_{17} + b_{61} + b_{64})$$
$$p_{148} = (a_{26} + b_{74}) \cdot (a_{27} + b_{61} + b_{64})$$
$$p_{149} = (a_{36} + b_{74}) \cdot (a_{37} + b_{61} + b_{64})$$
$$p_{150} = (a_{46} + b_{74}) \cdot (a_{47} + b_{61} + b_{64})$$

$$p_{151} = (a_{56} + b_{74}) \cdot (a_{57} + b_{61} + b_{64}) \qquad p_{153} = (a_{76} + b_{74}) \cdot (a_{77} + b_{61} + b_{64})$$
$$p_{152} = (a_{66} + b_{74}) \cdot (a_{67} + b_{61} + b_{64}) \qquad p_{154} = (a_{86} + b_{74}) \cdot (a_{87} + b_{61} + b_{64})$$

After computing all of the linear combinations, the elements of the matrix $\boldsymbol{C}$ are obtained as follows:

$$c_{11} = p_1 + p_9 + p_{29} - p_{25} - p_{26} + p_{69} + p_{77} + p_{91} + p_{99}$$
$$c_{21} = p_2 + p_{10} + p_{30} - p_{25} - p_{26} + p_{70} + p_{78} + p_{92} + p_{100}$$
$$c_{31} = p_3 + p_{11} + p_{31} - p_{25} - p_{26} + p_{71} + p_{79} + p_{93} + p_{101}$$
$$c_{41} = p_4 + p_{12} + p_{32} - p_{25} - p_{26} + p_{72} + p_{80} + p_{94} + p_{102}$$
$$c_{51} = p_5 + p_{13} + p_{33} - p_{25} - p_{26} + p_{73} + p_{81} + p_{95} + p_{103}$$
$$c_{61} = p_6 + p_{14} + p_{34} - p_{25} - p_{26} + p_{74} + p_{82} + p_{96} + p_{104}$$
$$c_{71} = p_7 + p_{15} + p_{35} - p_{25} - p_{26} + p_{75} + p_{83} + p_{97} + p_{105}$$
$$c_{81} = p_8 + p_{16} + p_{36} - p_{25} - p_{26} + p_{76} + p_{84} + p_{98} + p_{106}$$

$$c_{12} = p_1 + p_{17} + p_{37} - p_{25} - p_{27} + p_{107} + p_{115} - p_{69} - p_{77} - p_{85} - p_{86}$$
$$c_{22} = p_2 + p_{18} + p_{38} - p_{25} - p_{27} + p_{108} + p_{116} - p_{70} - p_{78} - p_{85} - p_{86}$$
$$c_{32} = p_3 + p_{19} + p_{39} - p_{25} - p_{27} + p_{109} + p_{117} - p_{71} - p_{79} - p_{85} - p_{86}$$
$$c_{42} = p_4 + p_{20} + p_{40} - p_{25} - p_{27} + p_{110} + p_{118} - p_{72} - p_{80} - p_{85} - p_{86}$$
$$c_{52} = p_5 + p_{21} + p_{41} - p_{25} - p_{27} + p_{111} + p_{119} - p_{73} - p_{81} - p_{85} - p_{86}$$
$$c_{62} = p_6 + p_{22} + p_{42} - p_{25} - p_{27} + p_{112} + p_{120} - p_{74} - p_{82} - p_{85} - p_{86}$$
$$c_{72} = p_7 + p_{23} + p_{43} - p_{25} - p_{27} + p_{113} + p_{121} - p_{75} - p_{83} - p_{85} - p_{86}$$
$$c_{82} = p_8 + p_{24} + p_{44} - p_{25} - p_{27} + p_{114} + p_{122} - p_{76} - p_{84} - p_{85} - p_{86}$$

$$c_{13} = p_9 + p_{17} + p_{45} - p_{26} - p_{27} + p_{123} + p_{131} - p_{69} - p_{77} - p_{87} - p_{88}$$

$$c_{23} = p_{10} + p_{18} + p_{46} - p_{26} - p_{27} + p_{124} + p_{132} - p_{70} - p_{78} - p_{87} - p_{88}$$

$$c_{33} = p_{11} + p_{19} + p_{47} - p_{26} - p_{27} + p_{125} + p_{133} - p_{71} - p_{79} - p_{87} - p_{88}$$

$$c_{43} = p_{12} + p_{20} + p_{48} - p_{26} - p_{27} + p_{126} + p_{134} - p_{72} - p_{80} - p_{87} - p_{88}$$

$$c_{53} = p_{13} + p_{21} + p_{49} - p_{26} - p_{27} + p_{127} + p_{135} - p_{73} - p_{81} - p_{87} - p_{88}$$

$$c_{63} = p_{14} + p_{22} + p_{50} - p_{26} - p_{27} + p_{128} + p_{136} - p_{74} - p_{82} - p_{87} - p_{88}$$

$$c_{73} = p_{15} + p_{23} + p_{51} - p_{26} - p_{27} + p_{129} + p_{137} - p_{75} - p_{83} - p_{87} - p_{88}$$

$$c_{83} = p_{16} + p_{24} + p_{52} - p_{26} - p_{27} + p_{130} + p_{138} - p_{76} - p_{84} - p_{87} - p_{88}$$

$$c_{14} = p_1 + p_{53} + p_{61} - p_{25} - p_{28} + p_{139} + p_{147} - p_{69} - p_{77} - p_{89} - p_{90}$$

$$c_{24} = p_2 + p_{54} + p_{62} - p_{25} - p_{28} + p_{140} + p_{148} - p_{70} - p_{78} - p_{89} - p_{90}$$

$$c_{34} = p_3 + p_{55} + p_{63} - p_{25} - p_{28} + p_{141} + p_{149} - p_{71} - p_{79} - p_{89} - p_{90}$$

$$c_{44} = p_4 + p_{56} + p_{64} - p_{25} - p_{28} + p_{142} + p_{150} - p_{72} - p_{80} - p_{89} - p_{90}$$

$$c_{54} = p_5 + p_{57} + p_{65} - p_{25} - p_{28} + p_{143} + p_{151} - p_{73} - p_{81} - p_{89} - p_{90}$$

$$c_{64} = p_6 + p_{58} + p_{66} - p_{25} - p_{28} + p_{144} + p_{152} - p_{74} - p_{82} - p_{89} - p_{90}$$

$$c_{74} = p_7 + p_{59} + p_{67} - p_{25} - p_{28} + p_{145} + p_{153} - p_{75} - p_{83} - p_{89} - p_{90}$$

$$c_{84} = p_8 + p_{60} + p_{68} - p_{25} - p_{28} + p_{146} + p_{154} - p_{76} - p_{84} - p_{89} - p_{90}$$

APPENDIX D

EFFICIENT MULTIPLICATION FORMULA FOR KYBER512

Let $m \geq 2$ messages are needed to be encrypted with Kyber512. Then, $\boldsymbol{A} \in R_q^{2\times 2}$, $\boldsymbol{B} \in R_q^{2\times 2}$ and $\boldsymbol{A}\cdot\boldsymbol{B} = \boldsymbol{C} \in R_q^{2\times 2}$. The elements of the matrices $\boldsymbol{A}^{2\times 2}\cdot\boldsymbol{B}^{2\times 2} = \boldsymbol{C}^{2\times 2}$ are described as follows:

By [35], the matrix $\boldsymbol{C}$ can be obtained as follows:

$$
\begin{aligned}
p_1 &= (a_{11} + a_{22}) \cdot (b_{11} + b_{22}) \\
p_2 &= (a_{21} + a_{22}) \cdot b_{11} \\
p_3 &= a_{11} \cdot (b_{12} - b_{22}) \\
p_4 &= a_{22} \cdot (-b_{11} + b_{21}) \\
p_5 &= (a_{11} + a_{12}) \cdot b_{22} \\
p_6 &= (-a_{11} + a_{21}) \cdot (b_{11} + b_{12}) \\
p_7 &= (a_{12} - a_{22}) \cdot (b_{21} + b_{22})
\end{aligned}
$$

After computing all of the linear combinations, one can obtain the element of the matrix $\boldsymbol{C}$ as follows:

$$
\begin{aligned}
c_{11} &= p_1 + p_4 - p_5 + p_7 & \qquad c_{21} &= p_2 + p_4 \\
c_{12} &= p_3 + p_5 & \qquad c_{22} &= p_1 - p_2 + p_3 + p_6
\end{aligned}
$$

APPENDIX E

EFFICIENT MULTIPLICATION FORMULA FOR KYBER768

Let $m \geq 4$ messages are needed to be encrypted with Kyber768. Then, $\boldsymbol{A} \in R_q^{3\times 3}$, $\boldsymbol{B} \in R_q^{3\times 4}$ and $\boldsymbol{A}\cdot\boldsymbol{B} = \boldsymbol{C} \in R_q^{3\times 4}$. The elements of the matrices $\boldsymbol{A}^{3\times 3}\cdot\boldsymbol{B}^{3\times 4} = \boldsymbol{C}^{3\times 4}$ are described as follows:

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}}_{\boldsymbol{B}} = \underbrace{\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \\ c_{51} & c_{52} & c_{53} & c_{54} \end{bmatrix}}_{\boldsymbol{C}}$$

By [29], the matrix $\boldsymbol{C}$ can be obtained as follows:

$$\begin{aligned} p_1 &= (a_{11} + b_{21}) \cdot (a_{12} - b_{12}) \\ p_2 &= (a_{21} + b_{21}) \cdot (a_{22} - b_{12}) \\ p_3 &= (a_{31} + b_{21}) \cdot (a_{32} - b_{12}) \\ p_4 &= (a_{11} + b_{31}) \cdot (a_{13} - b_{13}) \\ p_5 &= (a_{21} + b_{31}) \cdot (a_{23} - b_{13}) \\ p_6 &= (a_{31} + b_{31}) \cdot (a_{33} - b_{13}) \\ p_7 &= b_{12} \cdot b_{21} \\ p_8 &= b_{13} \cdot b_{31} \\ p_9 &= b_{23} \cdot b_{32} \end{aligned}$$

$$
\begin{aligned}
p_{10} &= (a_{12} + b_{32}) \cdot (a_{13} + b_{23}) \\
p_{11} &= (a_{22} + b_{32}) \cdot (a_{23} + b_{23}) \\
p_{12} &= (a_{32} + b_{32}) \cdot (a_{33} + b_{23}) \\
p_{13} &= (b_{21} - b_{24}) \cdot (-b_{12} + b_{14}) \\
p_{14} &= a_{11} \cdot (b_{11} - b_{12} - b_{13} - a_{12} - a_{13}) \\
p_{15} &= a_{21} \cdot (b_{11} - b_{12} - b_{13} - a_{22} - a_{23}) \\
p_{16} &= a_{31} \cdot (b_{11} - b_{12} - b_{13} - a_{32} - a_{33}) \\
p_{17} &= a_{12} \cdot (b_{22} - b_{21} - b_{23} - a_{11} - a_{13}) \\
p_{18} &= a_{22} \cdot (b_{22} - b_{21} - b_{23} - a_{21} - a_{23}) \\
p_{19} &= a_{32} \cdot (b_{22} - b_{21} - b_{23} - a_{31} - a_{33}) \\
p_{20} &= a_{13} \cdot (b_{33} - b_{31} - b_{32} - a_{11} - a_{12}) \\
p_{21} &= a_{23} \cdot (b_{33} - b_{31} - b_{32} - a_{21} - a_{22}) \\
p_{22} &= a_{33} \cdot (b_{33} - b_{31} - b_{32} - a_{31} - a_{32}) \\
p_{23} &= (a_{11} + b_{21} - b_{24}) \cdot (-a_{12} - b_{12} + b_{14}) \\
p_{24} &= (a_{21} + b_{21} - b_{24}) \cdot (-a_{22} - b_{12} + b_{14}) \\
p_{25} &= (a_{31} + b_{21} - b_{24}) \cdot (-a_{32} - b_{12} + b_{14}) \\
p_{26} &= a_{13} \cdot b_{34} \\
p_{27} &= a_{23} \cdot b_{34} \\
p_{28} &= a_{33} \cdot b_{34}
\end{aligned}
$$

After computing all of the linear combinations, one can obtain the element of the matrix $\boldsymbol{C}$ as follows:

$$
\begin{aligned}
c_{11} &= p_1 + p_4 + p_{14} - p_7 - p_8 \\
c_{12} &= p_1 + p_{10} + p_{17} - p_7 - p_9 \\
c_{13} &= p_4 + p_{10} + p_{20} - p_8 - p_9 \\
c_{14} &= p_1 + p_{23} + p_{26} - p_7 - p_{13}
\end{aligned}
$$

$$c_{21} = p_2 + p_5 + p_{15} - p_7 - p_8$$

$$c_{22} = p_2 + p_{11} + p_{18} - p_7 - p_9$$

$$c_{23} = p_5 + p_{11} + p_{21} - p_8 - p_9$$

$$c_{24} = p_2 + p_{24} + p_{27} - p_7 - p_{13}$$

$$c_{31} = p_3 + p_6 + p_{16} - p_7 - p_8$$

$$c_{32} = p_3 + p_{12} + p_{19} - p_7 - p_9$$

$$c_{33} = p_6 + p_{12} + p_{22} - p_8 - p_9$$

$$c_{34} = p_3 + p_{25} + p_{28} - p_7 - p_{13}$$

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Türe, Nazlı Deniz
Nationality: Turkish (T.C.)

EDUCATION

Degree	Institution	Year of Graduation
B.S.	Middle East Technical University - MATH	2018

PROFESSIONAL EXPERIENCE

Year	Place	Enrollment
(2020-Today)	FAME CRYPT	Cryptographic Researcher

PUBLICATIONS

International Conference Publications

- Türe, N. D. & Cenk, M. (2024). Efficient Batch Post-Quantum Signatures with Crystals Dilithium. In Proceedings of the International Workshop on the Arithmetic of Finite Fields (WAIFI 2024). Lecture Notes in Computer Science (LNCS). Cham: Springer.