Efficient Batch Algorithms for the Post-Quantum Crystals Dilithium Signature Scheme and Crystals Kyber Encryption Scheme

Nazlı Deniz Türe[1,2] and Murat Cenk[3]

[1] Middle East Technical University, Ankara, Turkey denizzzture@gmail.com
[2] FAME CRYPT, Ankara, Turkey
[3] Ripple Labs Inc. mcenk77@gmail.com

Abstract. Digital signatures ensure authenticity and secure communication. They are used to verify the integrity and authenticity of signed documents and are widely utilized in various fields such as information technologies, finance, education, and law. They are crucial in securing servers against cyberattacks and 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. Performing batch encryption, signature generation, and signature verification simultaneously and efficiently is highlighted as a beneficial approach for many systems. This work is the extended version of a conference paper that focuses on efficient batch signature generation with Dilithium, presented in 2024 at the International Workshop on the Arithmetic of Finite Fields (WAIFI 2024). We add efficient matrix multiplication methods for batch verifications of signatures from a single user using Crystals Dilithium (NIST's post-quantum digital signature standard) and batch encryption to a single receiver 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 paper, we propose to use efficient matrix multiplications of sizes greater than four to generate/verify m signatures with Dilithium and greater than two to encrypt m messages with Kyber. To this end, batch algorithms that transform the polynomial matrix-vector multiplication in Dilithium's and Kyber's structures into polynomial matrix-matrix multiplication are designed. The batch numbers and the sizes of the matrices to be multiplied based on the number of repetitions of Dilithium's signature algorithm are determined. Also, batch versions of Dilithium verification and Kyber encryption algorithms are proposed. Moreover, many efficient matrix-matrix multiplication algorithms, such as Strassen-like multiplications and commutative matrix multiplications, are analyzed to design the best algorithms that are compatible with the specified dimensions and yield improvements. Various multiplication formulas are derived for different security levels of Dilithium signature generation, verification, and Kyber encryption. 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 proposed batch Dilithium signature algorithm and the efficient multiplication algorithms are also implemented, and 34.22%, 17.40%, and 10.15% improvements on CPU cycle counts for three security levels are obtained. 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.

1 Introduction

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 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 [16] and ECDSA [14] are used as digital signing algorithms on many major servers, such as TLS servers. 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 [4] has developed a system based on ECDSA and RSA that provides multiple signings for TLS. Techniques such as ElGamal [9], ECDSA, Crystals Dilithium [8] and Falcon [12] have implemented multiple signing in previous studies ([6], [13], [1]). Fiat [11] and Tanwar et al. [27] present a batch algorithm that depends on the RSA system. Pavlovski et al. [20] propose an efficient batch signature generation via binary tree structures.

In addition to batch digital signature generation, batch verifications of signatures signed by a single user 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. A practical method of batch verification of short signatures is described in [10]. [15] 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 [11] has introduced a system where RSA is used in batch encryption. In [7], a fully homomorphic encryption scheme over the integers of van Dijk et al. [28] is extended into its batch version.

High-capacity quantum computers can be used to break many of the cryptographic algorithms in use today, as demonstrated in 1994 by P. Shor [24]. 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 is selected as the digital signing standard (NIST FIPS 204-Module-Lattice-Based Digital Signature Standard [17]), while CRYSTALS Kyber [3] is chosen as the encryption/key encapsulation standard (NIST FIPS 203-Module-Lattice-Based Key-Encapsulation Mechanism Standard [18]). Following this process, Dilithium's integrability into TLS 1.3 is demonstrated in [25]. However, research has shown that using post-quantum in the TLS server will cause performance degradation [19]. On the encryption side, Kyber is replacing the current encryption methods after it has been selected as the encryption/KEM standard. In [22], the Kyber Key Encapsulation Mechanism is integrated into the cloud architecture to improve security. Moreover, [23] suggests an effective method for securing the device-to-device connection with the use of Crystals Kyber for data transmission.

In this work, 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.

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 [26], [30], [5], and [21], in multiple signing can increase efficiency. Note that since the entries are large-size polynomials, reducing the number of multiplications contributes significantly to complexity.

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). Matrix-vector multiplication in Dilithium's and Kyber's structure is converted to matrix-matrix multiplication using the batch technique. Batch numbers for Dilithium are determined separately for each security level, according to the number of repetitions in the signing and the probability that the signature produced is valid. Suitable matrix-matrix multiplication techniques are chosen for the selected batch numbers (4, 5, and 4 for Batch Dilithium 2, Batch Dilithium 3, and Batch Dilithium 5, respectively). Batch numbers (2, 4, and 4) are chosen as an example of Kyber's three security levels due to their ease of implementation. Since Dilithium's matrix and vector entries are polynomials from the ring $R_q = \mathbb{Z}_{8380417}[x]/(x^{256}+1)$ and Kyber's are from $R'_q = \mathbb{Z}_{3329}[x]/(x^{256}+1)$, multiplication is much more expensive than addition. Reducing the number of multiplications is a priority to increase efficiency when selecting these algorithms. Therefore, matrix multiplication algorithms are chosen based on the cost metric minimizing multiplications. It is observed that using [21] for Dilithium 2, Dilithium 3, Dilithium 5, Kyber768 and Kyber1024 is more functional, while [26] and [29] 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 [5], [26], [30] to it recursively. However, note that multiplication in [21] cannot be used recursively since it requires entries to be commutative, and matrix multiplications are non-commutative. The multiplication formulas are derived from the most efficient matrix-matrix multiplication techniques based on batch numbers, which determine the size of the matrices to be multiplied. 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

The organization of the remainder of the paper is as follows: In Section 2, related notations and functions are described, and the Crystals Dilithium and Crystals Kyber algorithms are introduced. Sections 3, 4, and 5 cover the presentation of the new batch algorithms of Dilithium signature generation, verification and Kyber encryption, as well as their arithmetic complexity and implementation analyses. Section 6 summarizes this research's accomplishments. Finally, formulas derived using some efficient matrix-matrix multiplication methods in the literature are explained in Appendices A, B, C, D, and E.

2 Preliminaries

2.1 Notations and Subfunctions

Notations and functions that are used in the remaining sections are given in Table 1 and Table 2.

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
KEM	Key Encapsulation Mechanism
CCA	Chosen Ciphertext Attack
CPA	Chosen Plaintext Attack

Table 1. Notations and their definitions

Function	**Definition**
NTT	Number Theoretic Transform described in [8]
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 [8]
$\|z\|_\infty$	$\|z \mod {}^{\pm}q\|$ defined in [8]
$\|\|$	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 [3]
$Compress$	Compression function defined in [3]
$Decompress$	Decompression function defined in [3]

Table 2. Functions and their definitions

2.2 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 1, Algorithm 2, and Algorithm 3, respectively.

The mechanism for generating keys in Dilithium is described in Algorithm 1. 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 1 Dilithium Key Generation [8]

```
Output: Public key $pk$, Secret key $sk$
1: $\zeta \leftarrow \{0,1\}^{256}$
2: $(\rho, \rho', K) \in \{0,1\}^{256} \times \{0,1\}^{512} \times \{0,1\}^{256} := H(\zeta)$
3: $\hat{\boldsymbol{A}} \in R_q^{k \times l} := ExpandA(\rho)$        ▷ $\boldsymbol{A}$ is generated in NTT representation as $\hat{\boldsymbol{A}}$
4: $(\boldsymbol{s_1}, \boldsymbol{s_2}) \in S_\eta^l \times S_\eta^k := ExpandS(\rho')$
5: $\boldsymbol{t} := NTT^{-1}(\hat{\boldsymbol{A}} \cdot NTT(\boldsymbol{s_1})) + \boldsymbol{s_2}$
6: $(\boldsymbol{t_1}, \boldsymbol{t_0}) := Power2Round_q(\boldsymbol{t}, d)$
7: $tr \in \{0,1\}^{256} := H(\rho \| \boldsymbol{t_1})$
8: return $pk = (\rho, \boldsymbol{t_1})$, $sk = (\rho, K, tr, \boldsymbol{s_1}, \boldsymbol{s_2}, \boldsymbol{t_0})$
```

Algorithm 2 Dilithium Signing [8]

```
Input: Secret key $sk = (\rho, K, tr, \boldsymbol{s_1}, \boldsymbol{s_2}, \boldsymbol{t_0})$, message $M$
Output: Signature $\sigma$
1: $\hat{\boldsymbol{A}} \in R_q^{k \times l} := ExpandA(\rho)$        ▷ $\boldsymbol{A}$ is generated in NTT representation as $\hat{\boldsymbol{A}}$
2: $\mu \in \{0,1\}^{512} := H(tr \| M)$
3: $\kappa := 0$, $(\boldsymbol{z}, \boldsymbol{h}) = \perp$
4: $\rho' \in \{0,1\}^{512} := H(K \| \mu)$
5: $\hat{\boldsymbol{s}}_1 := NTT(\boldsymbol{s}_1)$
6: $\hat{\boldsymbol{s}}_2 := NTT(\boldsymbol{s}_2)$
7: $\hat{\boldsymbol{t}}_0 := NTT(\boldsymbol{t}_0)$
8: while $(\boldsymbol{z}, \boldsymbol{h}) = \perp$ do
9:     $\boldsymbol{y} \in \tilde{S}_{\gamma_1}^l := ExpandMask(\rho', \kappa)$
10:    $\boldsymbol{w} := NTT^{-1}(\hat{\boldsymbol{A}} \cdot NTT(\boldsymbol{y}))$
11:    $\boldsymbol{w_1} := HighBits_q(\boldsymbol{w}, 2\gamma_w)$
12:    $\tilde{c} \in \{0,1\}^{256} := H(\mu \| \boldsymbol{w_1})$
13:    $c \in B_\tau := SampleInBall(\tilde{c})$
14:    $\boldsymbol{z} := \boldsymbol{y} + NTT^{-1}(\hat{c}\hat{\boldsymbol{s}}_1)$        ▷ $\hat{\boldsymbol{c}} = NTT(\boldsymbol{c})$
15:    $\boldsymbol{r_0} := LowBits_q(\boldsymbol{w} - NTT^{-1}(\hat{c} \cdot \hat{\boldsymbol{s}}_2), 2\gamma_2)$
16:    if $\|\boldsymbol{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\boldsymbol{r_0}\|_\infty \geq \gamma_2 - \beta$ then
17:        $(\boldsymbol{z}, \boldsymbol{h}) := \perp$
18:    else
19:        $\boldsymbol{h} := MakeHint_q(-NTT^{-1}(\hat{c} \cdot \hat{\boldsymbol{t}}_0), \boldsymbol{w} - c\boldsymbol{s_2} + NTT^{-1}(\hat{c} \cdot \hat{\boldsymbol{t}}_0), 2\gamma_2)$
20:        if $\|c\boldsymbol{t}_0\|_\infty \geq \gamma_2$ or the # of 1's in $\boldsymbol{h}$ is greater than $\omega$ then
21:            $(\boldsymbol{z}, \boldsymbol{h}) := \perp$
22:    $\kappa := \kappa + l$
23: return $\sigma = (\tilde{c}, \boldsymbol{z}, \boldsymbol{h})$
```

Algorithm 2 provides a description of the Dilithium signature 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 [8], 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 3.

Algorithm 3 Dilithium Verification [8]

```
Input: Public key $pk = (\rho, \boldsymbol{t_1})$, message $M$, signature $\sigma$
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 [$\tilde{c} = H(\mu \parallel \boldsymbol{w_1'})$] and [# of 1's in $\boldsymbol{h}$ is $\leq \omega$]
```

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

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

Table 3. Parameter Sets for Dilithium [8]

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 3, the matrix-vector multiplications in the 5th step of the Algorithm 1, the 10th step of the Algorithm 2, and the 4th step of the Algorithm 3 are one of the most expensive operations of Dilithium. Consider an example where $\boldsymbol{A}$ belongs to $R_q^{6\times 5}$ 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 Section **??**, the proposed algorithm, which guarantees that several messages can be signed simultaneously and more efficiently, is described.

2.3 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 4 and Algorithm 5, respectively.

Kyber CPAPKE encryption details are given in Algorithm 4. 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 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 5, 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 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 4 Kyber CPAPKE Encryption [3]

```
Input: Public key $pk$, message $m$, random coin $r$
Output: Ciphertext $c$
1: $N := 0$
2: $\hat{\boldsymbol{t}} := Decode_{12}(pk)$
3: $\rho := pk + 12 \cdot k \cdot n/8$
4: for $i = 0, 1, \dots, k-1$ do        ▷ $\boldsymbol{A}$ is generated in NTT representation as $\hat{\boldsymbol{A}}$
5:     for $j = 0, 1, \dots, k-1$ do
6:         $\hat{\boldsymbol{A}}^T[i][j] := Parse(XOF(\rho, i, j))$
7: for $i = 0, 1, \dots, k-1$ do
8:     $\boldsymbol{r}[i] := CBD_{\eta_1}(PRF(r, N))$
9:     $N := N + 1$
10: for $i = 0, 1, \dots, k-1$ do
11:     $\boldsymbol{e_1}[i] := CBD_{\eta_2}(PRF(r, N))$
12:     $N := N + 1$
13: $e_2 := CBD_{\eta_2}(PRF(r, N))$
14: $\hat{\boldsymbol{r}} := NTT(\boldsymbol{r})$
15: $\boldsymbol{u} := NTT^{-1}(\hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{r}}) + \boldsymbol{e_1}$
16: $v := NTT^{-1}(\hat{\boldsymbol{t}}^T \cdot \hat{\boldsymbol{r}}) + e_2 + Decompress_q(Decode_1(m), 1)$
17: $c_1 := Encode_{d_u}(Compress_q(\boldsymbol{u}, d_u))$
18: $c_2 := Encode_{d_v}(Compress_q(v, d_v))$
19: return $c = (c_1 \| c_2)$
```

Algorithm 5 Kyber CCAKEM Encapsulation [3]

```
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 $c = (c, K)$
```

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)

Table 4. Parameter sets for Kyber

The matrix-vector multiplication in the 15^{th} step of the Algorithm 4 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 using the Kyber algorithm is described in Section 5.

3 Efficient Batch Dilithium Signing

This section explains the proposed batch version of Dilithium's signing algorithm. Algorithm 6 describes the batch Dilithium Signature Algorithm, which is based on the Dilithium Signature Algorithm given in Algorithm 2 and enables signing multiple messages at once. As an example, the suggested strategy chooses batch numbers 4, 5, and 4 for Dilithium 2, 3, and 5, respectively. A detailed explanation of batch number selection is provided in Section 3.2.

The user's secret key $sk = (\rho, K, tr, s_1, s_2, t_0)$ and m distinct messages are the inputs of the batch signing algorithm. The indices for these messages are $M_0, M_1, \ldots, M_{m-1}$. First, ρ is used to create the matrix $\hat{\boldsymbol{A}}$, using Algorithm 2. The values of μ_i, κ_i, and ρ'_i for each message M_i are then calculated. The starting value of $(\boldsymbol{z}'_i, \boldsymbol{h}'_i)$ is set to $\perp$, and κ_i values begin at 0. There is defined a $waitList = 0, 1, 2, \ldots, m-1$ list that contains the indices of messages that are awaiting to be signed. The messages that are simultaneously in the signing phase are indicated by the first p indices in the list. The indices of the messages with the proper signatures are removed from this list at the end of the while loop. The secret key elements, $\boldsymbol{s_1}$, $\boldsymbol{s_2}$, and $\boldsymbol{t_0}$, are converted to their NTT formats, $\hat{\boldsymbol{s_2}}$, $\hat{\boldsymbol{s_2}}$, and $\hat{\boldsymbol{t_0}}$, just before the signature phase begins. The signing loop starts after the preliminary setup is finished. The signature candidates that are generated need to fulfill specific requirements. $status_i$ determines the status of these requirements.

Algorithm 6 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 $\sigma_i$, where $i = 0, 1, \ldots, m-1$
1: $\hat{\boldsymbol{A}} \in R_q^{k \times l} := ExpandA(\rho)$        ▷ $\boldsymbol{A}$ is generated in NTT representation as $\hat{\boldsymbol{A}}$
2: for $i = 0, 1, \ldots, m-1$ do
3:     $\mu_i \in \{0,1\}^{512} := H(tr \| M_i)$
4:     $\kappa_i := 0$, $(\boldsymbol{z_i'}, \boldsymbol{h_i'}) = \perp$
5:     $\rho_i' \in \{0,1\}^{512} := H(K \| \mu_i)$
6: $waitList_i = i \quad \forall i = 0, 1, 2, \ldots, m-1$
7: $\hat{\boldsymbol{s}}_1 := NTT(\boldsymbol{s}_1)$
8: $\hat{\boldsymbol{s}}_2 := NTT(\boldsymbol{s}_2)$
9: $\hat{\boldsymbol{t}}_0 := NTT(\boldsymbol{t}_0)$
10: while Length of $waitList \geq p$ do        ▷ Batch numbers $p = 4, 5, 4$ for Dilithium 2, 3, 5.
11:     $status_i = 0 \quad \forall i = 0, 1, 2, \ldots, m-1$
12:     for $i = 0, 1, \ldots, p-1$ do
13:         $\boldsymbol{y}_i \in \tilde{S}_{\gamma_1}^l := ExpandMask(\rho'_{waitList_i}, \kappa_{waitList_i})$
14:     $\hat{\boldsymbol{Y}}$:= $(l \times p)$ matrix, whose columns are $NTT(\boldsymbol{y}_i)$'s, where $i = 0, 1, 2, \ldots, p-1$
15:     $\hat{\boldsymbol{W}} := \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{Y}}$,   $\hat{\boldsymbol{W}} : (k \times p)$ matrix, whose columns are $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$'s, where $i = 0, 1, 2, \ldots, p-1$
16:     $\boldsymbol{W}$ := $(k \times p)$ matrix, whose columns are $NTT^{-1}(\hat{\boldsymbol{w}}_{\boldsymbol{i}})$'s, where $i = 0, 1, 2, \ldots, p-1$
17:     for $i = 0, 1, \ldots, p-1$ do
18:         $\boldsymbol{w_i'} := HighBits_q(\boldsymbol{i}, 2\gamma_2)$
19:         $\tilde{c}_{waitList_i} \in \{0,1\}^{256} := H(\mu_{waitList_i} \| \boldsymbol{w_i'})$
20:         $c_{waitList_i} \in B_\tau := SampleInBall(\tilde{c}_{waitList_i})$
21:         $\hat{c}_{waitList_i} = NTT(c_{waitList_i})$
22:         $\boldsymbol{z_i'} := \boldsymbol{y}_i + NTT^{-1}(\hat{c}_{waitList_i} \cdot \hat{\boldsymbol{s}}_{\boldsymbol{1}})$
23:         $\boldsymbol{r_i} := LowBits_q(\boldsymbol{w}_i - NTT^{-1}(\hat{c}_{waitList_i} \cdot \hat{\boldsymbol{s}}_{\boldsymbol{2}}), 2\gamma_2)$
24:         if $\|\boldsymbol{z_i'}\|_\infty \geq \gamma_1 - \beta$ then
25:             $status_i = status_i + 1$
26:         if $\|\boldsymbol{r}_i\|_\infty \geq \gamma_2 - \beta$ then
27:             $status_i = status_i + 1$
28:         $\boldsymbol{h_i'} := MakeHint_q(-NTT^{-1}(\hat{c}_{waitList_i} \cdot \hat{\boldsymbol{t}}_{\boldsymbol{0}}), \boldsymbol{w}_i - c_{waitList_i} \cdot \boldsymbol{s_2} + NTT^{-1}(\hat{c}_{waitList_i} \cdot \hat{\boldsymbol{t}}_{\boldsymbol{0}}), 2\gamma_2)$
29:         if $\|c\boldsymbol{t}_0\|_\infty \geq \gamma_2$ then
30:             $status_i = status_i + 1$
31:         if The # of 1's in $\boldsymbol{h_i'}$ is greater than $\omega$ then
32:             $status_i = status_i + 1$
33:         $\kappa_{waitList_i} = \kappa_{waitList_i} + l$
34:     for $i = 0, 1, \ldots, p-1$ do
35:         if !$status_i$ then
36:             $\boldsymbol{z}_{waitList_i} = \boldsymbol{z_i'}$
37:             $\boldsymbol{h}_{waitList_i} = \boldsymbol{h_i'}$
38:             Delete $waitList_i$ from the list
39: return $\sigma_i = (\tilde{c}_i, \boldsymbol{z}_i, \boldsymbol{h}_i)$ where $i = 0, 1, \ldots, m-1$
```

The user's secret key $sk = (\rho, K, tr, s_1, s_2, t_0)$ and m distinct messages are the inputs of the batch signing algorithm. The indices for these messages are $M_0, M_1, \ldots, M_{m-1}$. First, ρ is used to create the matrix $\hat{\boldsymbol{A}}$, using Algorithm 2. The values of μ_i, κ_i, and ρ_i' for each message M_i are then calculated. The starting value of $(\boldsymbol{z_i'}, \boldsymbol{h_i'})$ is set to $\perp$, and κ_i values begin at 0. There is defined a $waitList = 0, 1, 2, \ldots, m-1$ list that contains the indices of messages that are awaiting to be signed. The messages that are simultaneously in the signing phase are indicated by the first p indices in the list. The indices of the messages with the proper signatures are removed from this list at the end of the while loop. The secret key elements, $\boldsymbol{s_1}$, $\boldsymbol{s_2}$, and $\boldsymbol{t_0}$, are converted to their NTT formats, $\hat{\boldsymbol{s_2}}$, $\hat{\boldsymbol{s_2}}$, and $\hat{\boldsymbol{t_0}}$, just before the signature phase begins. The signing loop starts after the preliminary setup is finished. The signature candidates that are generated need to fulfill specific requirements. $status_i$ determines the status of these requirements.

Note that the y_i values are calculated and converted to their NTT format utilizing the ρ'_i and κ_i values of the relevant messages. The columns of the $\hat{\boldsymbol{Y}}$ matrix are formed by the generated $\hat{\boldsymbol{y_i}}$. In the 15th step, the algorithm performs its batch operation. Matrix-matrix multiplication is used in place of the matrix-vector multiplication in this phase. NTT^{-1} is used to transform each column of the matrix $\hat{\boldsymbol{W}}$, that is obtained from multiplication, into its normal form. In the signing phase, these are used for p messages and are referred to as w_i's. The operations carried out for a single message in the original method are applied to p messages in Steps 17–33. The $status_i$'s are updated after "if" statements are used to check the candidate signatures produced for every message. In steps 34 through 38, candidate signatures that satisfy all requirements are recognized as valid signatures, and the *waitList* is cleared of the indices of correctly signed messages. With new κ_i values (which are updated in step 33), p messages that are in the queue in the *waitList* enter the loop. Until less than p elements remain in the *waitList*, the loop is repeated. The messages that are not able to be signed using Algorithm 6 are signed individually using Algorithm 2. Therefore, all messages are signed correctly.

Example: Let $m = 8$, the messages M_0, M_1, M_2, M_3, M_4, M_5, M_6, M_7 to be signed using Dilithium 2 and let $\kappa = [\kappa_0, \kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa_6, \kappa_7] = [0, 0, 0, 0, 0, 0, 0]$ and the batch number p is set to be 4 in Algorithm 6, step 4. In the while loop, the signatures are created and examined to see if they fulfill the necessary requirements. Assume that the following generated signatures are valid in the correct order: $(\tilde{c}_3, \boldsymbol{z}_3, \boldsymbol{h}_3)$, $(\tilde{c}_1, \boldsymbol{z}_1, \boldsymbol{h}_1)$, $(\tilde{c}_5, \boldsymbol{z}_5, \boldsymbol{h}_5)$, $(\tilde{c}_6, \boldsymbol{z}_6, \boldsymbol{h}_6)$, and $(\tilde{c}_0, \boldsymbol{z}_0, \boldsymbol{h}_0)$. Table 5, for instance, illustrates the change of κ and *waitList* based on the number of loops and valid signatures produced.

# 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]

Table 5. Change of κ and *waitList* according to the number of loops and valid signatures generated.

First, Algorithm 6 generates the matrix $\hat{\boldsymbol{A}}$, and for $i = 0, 1, 2, \ldots, 7$, μ_i and ρ'_i are computed. Since $p = 4$, four messages are handled simultaneously. The first four elements of *waitList* determine which four messages are processed. $waitList = [0, 1, 2, 3, 4, 5, 6, 7]$ at the beginning. The first while loop attempts to generate signatures for the messages M_0, M_1, M_2, and M_3 since the first four components of the *waitList* are $[0, 1, 2, 3]$. The signature candidates generated by the first loop are $(\tilde{c}_0, \boldsymbol{z}_0, \boldsymbol{h}_0)$, $(\tilde{c}_1, \boldsymbol{z}_1, \boldsymbol{h}_1)$, $(\tilde{c}_2, \boldsymbol{z}_2, \boldsymbol{h}_2)$, and $(\tilde{c}_3, \boldsymbol{z}_3, \boldsymbol{h}_3)$, as shown by Table 5. Of these four signature candidates, only $\sigma_3 = (\tilde{c}_3, \boldsymbol{z}_3, \boldsymbol{h}_3)$ satisfies the requirements and is considered to be valid. The

κ values are adjusted to [4,4,4,4,0,0,0,0]. Three is eliminated from the *waitList* and *waitList* $= [0, 1, 2, 4, 5, 6, 7]$ since the signature created for M_3 is valid. In the second loop, signature candidates are generated for M_0, M_1, M_2, and M_4. Consider that among the four signatures produced at the end of the second round, only $(\tilde{c}_1, \boldsymbol{z}_1, \boldsymbol{h}_1)$ is correct. The list then becomes *waitList* $= [0, 2, 4, 5, 6, 7]$ after 1 is removed from the list and $\kappa = [8, 8, 8, 4, 4, 0, 0, 0]$. The process repeats until *waitList* has fewer elements than p. This implies that σ_3, σ_1, σ_5, σ_7, and σ_0 are generated, in that order, at the end of the Batch Dilithium 2 Signing algorithm.

When the batch algorithm is unable to sign the messages M_2, M_4, and M_7, each of them is signed separately using the Dilithium 2 Signing Algorithm (Algorithm 2). Going a step further, since some κ values are tried for these messages and no proper results are obtained, it is possible to initialize κ values of the original Dilithium 2 signature method to the values obtained at the end of the batch algorithm. Finally, all messages are signed by combining the batch and the classical Dilithium signing algorithms.

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

To make sure that the batch method is more efficient than the naive one, some techniques should be applied to the operation in step 15 of Algorithm 6. The operation becomes matrix-matrix multiplication instead of matrix-vector multiplication, which is performed independently for every message in the classical algorithm. The elements of these matrices are polynomials of degree 255 rather than integers because of the structure of Dilithium. Due to this, the multiplication of these matrices is one of the most costly phases of the algorithm, even though the matrix sizes are small. The batch version does not appear to offer an advantage over the original version if these multiplications are performed using the schoolbook approach. However, depending on the dimensions and characteristics of the matrices, there are several types of matrix-matrix multiplication algorithms in the literature that can be applied. The number of polynomial multiplications needed for this process can be decreased by selecting proper methods, and generating several signatures at once is more efficient than producing each one separately.

The Importance of Commutativity Property and Choosing The Proper Efficient Matrix-Matrix Multiplication Algorithms Table 6 shows the dimensions of the matrices to be multiplied in the 15^{th} step of Algorithm 6 based on the determined batch numbers and various security levels of Dilithium signing.

Security Level	Algorithm	Batch Number	Size of $\hat{\boldsymbol{A}}$	Size of $\hat{\boldsymbol{Y}}$	Size of $\hat{\boldsymbol{W}}$
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)

Table 6. Batch numbers and matrix sizes according to different security levels.

Many matrix-matrix multiplication techniques can be applied when taking Dilithium's matrix dimensions and structure into account. Dilithium's ring, $R = \mathbb{Z}_{8380417}[X]/(X^{256}+1)$, is commutative.Therefore, the multiplication of polynomials, which constitute the entries of matrices, is commutative. This characteristic yields the equality $a_{ij} \cdot y_{kl} = y_{kl} \cdot a_{ij}$ for any product of two entries, a_{ij} and y_{kl}. A fast commutative matrix-matrix multiplication technique described in [21] can be applied for all security levels by utilizing this property. Compared to the non-commutative approaches in the literature or the Strassen [26] method, this algorithm works more efficiently. For larger-size matrices, non-commutative approaches may also be chosen when matrix multiplication is done recursively.

The most effective recursive matrix-matrix multiplications are known to be Strassen-like multiplications, such as the Strassen technique, Cenk&Hassan's approach [5], and Winograd's Multiplication [30].

In this work, the Batch Dilithium 2 algorithm is constructed, and Strassen's approach and the fast commutative approach are used to obtain $(4\times 4)\cdot(4\times 4)$ matrix-matrix multiplication formulas. Using the fast commutative approach, matrix multiplication formulas for $(6\times 5)\cdot(5\times 5)$ and $(8\times 7)\cdot(7\times 4)$ are obtained for Batch Dilithium 3 and Dilithium 5.

3.2 Probability Computations and Choosing the Batch Sizes

Signature candidates have to satisfy certain requirements in order to be accepted by Dilithium's signing algorithm. "If statements" are used by the signature algorithm to confirm these requirements. These requirements are included in lines 16 and 20 in the classical algorithm (Algorithm 2) and in lines 24, 26, 29, and 31 in the batch algorithm (Algorithm 6). In Algorithm 2, Step 16 is more dominant in the condition checks. As a result, for step 13, the following formula—which is defined in the [8]—is utilized to calculate the probability that the generated signature is proper:

$$\approx e^{-256\cdot\beta(l/\gamma_1+k/\gamma_2)}. \tag{1}$$

Table 7 provides the values of the variables used in the formula along with the probability that a generated signature candidate fulfills the requirements depending on the different Dilithium security levels. The while loop reproduces signatures that don't meet the requirements.

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

Table 7. Variables required to compute the probability

The probability that at least one of the p signatures generated by Dilithium 2, Dilithium 3, and Dilithium 5 is proper is determined by the batch number and is $1-(1-0.24)^p$, $1-(1-0.196)^p$, and $1-(1-0.26)^p$, respectively.

As the batch number increases, the probability that at least one of the p signatures is correct also increases. Two scenarios must be considered in order to determine the batch number p. The first is the probability that at least one of the signatures of p messages is correct when they are all signed. The second is the improvement rate that the selected p provides. The user can choose a suitable batch number and decide the priority based on these two circumstances. Furthermore, the formulas to be derived based on the matrix sizes and potential matrix-matrix multiplication algorithms should be taken into account while selecting the batch number.

The probability that at least one of the p signatures is proper based on certain eligible batch numbers (p) is given for Dilithium 2, 3, and 5 in Table 8.

	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

Table 8. Probabilities of at least one of p signatures being correct according to batch number (p) for Dilithium 2, 3, and 5.

For Dilithium 2, 3, and 5, the probability of at least one of the p signatures being correct according to the batch number p is displayed in Figures 1, 2, and 3, respectively. The probability converges to 1 as the number of batches increases, as demonstrated by Table 8 and Figures 1, 2, and 3.

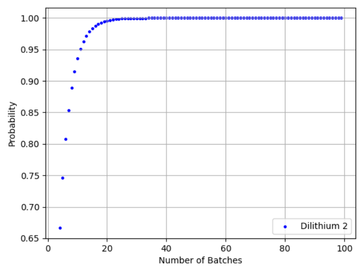

Fig. 1. Change in probability of at least one of p signatures generated with Dilithium 2 being correct according to batch number (p).

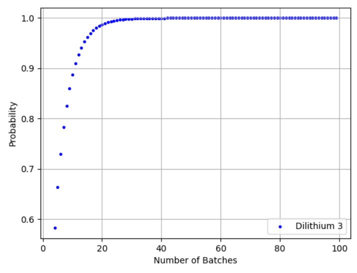

Fig. 2. Change in probability of at least one of p signatures generated with Dilithium 3 being correct according to batch number (p).

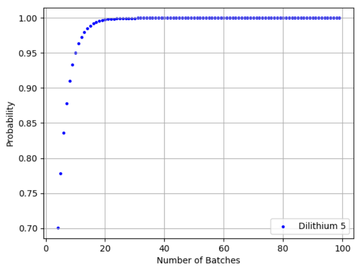

Fig. 3. Change in probability of at least one of p signatures generated with Dilithium 5 being correct according to batch number (p).

Table 9 lists the number of polynomial multiplication operations that must be performed based on the batch number that has been chosen. [21] is selected as the efficient matrix-matrix multiplication method for the batch method.

Using [21], the number of multiplication operations needed to multiply the matrices $\hat{\boldsymbol{A}}^{k\times l}$ and $\hat{\boldsymbol{Y}}^{l\times p}$ can be computed as $l(kp+k+p-1)/2$, where $k=4$ and $l=4$. Similarly, for Dilithium 3 and 5, where $k=6$, $l=5$ for Dilithium 3, and $k=8$, $l=7$ for Dilithium 5, the number of multiplication operations needed can be found as $(l(kp+k+p-1)+k-1)/2$ if p is even and $l(kp+k+p-1)/2$ for p is odd.

	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

Table 9. 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)	**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

Table 10. Improvement Rates (%) for 20 Signatures

For illustration, suppose that Batch Dilithium is used to sign 20 messages. Table 10 provides the improvement rates that will be attained by reducing the number of multiplications. As the table shows, choosing p equal to 5 yields the best improvement rate. Based on probability estimates and improvement percentages, p can be chosen by considering the available effective matrix-matrix multiplication methods. Let m be the number of messages to be signed, p be the number of batches, C be the number of multiplications needed for the classical technique, and B be the number of multiplications needed for the effective matrix-matrix multiplication algorithm used in the batch method. The following formula is used to determine the improvement rate:

$$(C\cdot m-[B\cdot(m-(p-1))+C\cdot(p-1)])100/(C\cdot m) \tag{2}$$

The probability of a signature being valid for Dilithium 2, 3, and 5 are approximately 1/4, 1/5, and 1/4, respectively, as shown in Table 7. Therefore, there is a high probability that one of the four signatures generated by Dilithium 2, one of the five for Dilithium 3, and one of the four for Dilithium 5 will be correct. Consequently, $1-(1-0.24)^4 \approx 0.67 = 67\%$ is the probability that at least one of the four signatures generated by Dilithium 2 is correct.

3.3 Results for Batch Dilithium Signing

The results of the batch application are examined in two sections: Improvements in the arithmetic complexity and improvements observed via the implementations.

Matrix Multiplications and Arithmetic Complexity Analysis For the batch method to be efficient, it needs to be combined with effective matrix-matrix multiplication algorithms. For this reason, the method outlined in Section 3.2 is utilized to identify the batch numbers to be used in the Batch Dilithium Algorithm for each security level. Table 6 provides the dimensions of the matrices in the 15^{th} step of Algorithm 6 for each security level of Dilithium. Reducing the number of multiplications is the goal since the matrices' entries are polynomials, and their multiplications are far more expensive than their additions. Recursively applying the Strassen-like matrix-matrix multiplications yields the smallest number of multiplications. However, the technique described in this work does not require recursions since the matrices are small in size. Since the entries are polynomials from the commutative ring R_q, the commutative matrix multiplication techniques that have better multiplicative complexity than the non-commutative multiplication algorithms can also be used for our purposes. For this reason, applying the commutative matrix-matrix multiplication approach as presented in [21] is recommended.

Let the product of a matrix of size $(n_1 \times n_2)$ and $(n_2 \times n_3)$ be (n_1, n_2, n_3).$(4, 4, 4)$, $(6, 5, 5)$ and, $(8, 7, 4)$ efficient matrix multiplication methods are required, for Dilithium2, 3, and 5, respectively. The commutative matrix multiplication method by [21] with 46 multiplications (Appendix A.2) and the Strassen-like recursive multiplications [26], [30], [5] with 49 multiplications (Appendix A.1) appear to be the best options for the $(4, 4, 4)$ multiplications. [21] may be used to do $(6, 5, 5)$ matrix multiplication in batch Dilithium 3 with 100 multiplications (Appendix B) and $(8, 7, 4)$ matrix multiplication in batch Dilithium 5 with 154 multiplications (Appendix C). All explicit formulas are derived for this work to implement efficient batch algorithms.

Batch Dilithium 2 Signing: Assume that the classical method is used to sign m messages independently using Dilithium 2. It takes 16 multiplications for each message. Nonetheless, $16 \cdot 4 = 64$ multiplications are required on average to sign the message because of the algorithm's failure probability. For m messages, a total of $64m$ multiplications must be performed.

Step 15 is performed for four signatures in Algorithm 6's batch approach. Considering that there is a 0.24 probability that a signature candidate for Dilithium 2 is valid, one of the four signatures can be taken as proper. When $m - (p - 1)$ messages are signed, this procedure ends. As a result, $46 \cdot (m - (p - 1)) = 46 \cdot (m - 3)$ multiplication operations are needed for the batch operation. Using classical Dilithium 2, each of the remaining $p - 1 = 3$ messages from the batch procedure is signed separately. To sign each message, step 7 of the Algorithm 2 needs 16 multiplications. The amount of multiplications needed for a message is $16 \cdot 4 = 64$ if we assume that the loop is repeated four times in order to produce a valid signature. Therefore, $3 \cdot 64 = 192$ is the number of multiplications needed for three messages, and approximately $46 \cdot (m - 3) + 192 = 46m + 54$ is the number of multiplications needed for batch Dilithium 2. Table 11 shows the number of multiplications required for the batch and classical approaches, as well as the improvement rates attained with batch Dilithium 2 for the chosen number of messages up to 100. The improvement rate is obtained as $(64m - (46m + 54)) \cdot 100/64m$, and it converges to 28.1% as the number of messages increases.

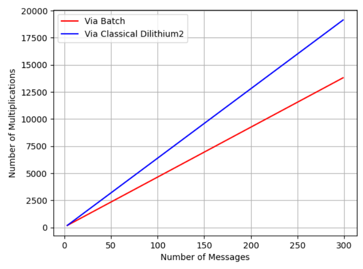

Fig. 4. Variation of the number of multiplications required according to the number of messages for Batch Dilithium 2 and Classical Dilithium 2

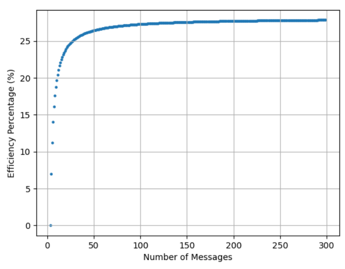

Fig. 5. Variation of the improvement rate (%) provided by the batch algorithm compared to the classical version depending on the number of messages for Dilithium 2

Figure 4 illustrates how the number of multiplications needed varies based on the number of messages in Batch Dilithium 2 and Classical Dilithium 2. Figure 5 demonstrates how the improvement rate (%) offered by the batch algorithm changes based on the number of messages when compared to the classical version.

# 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

Table 11. Variation of the number of multiplications required for batch and classical use of Dilithium 2 signing, and the improvement rates provided by batch method according to the number of messages

Batch Dilithium 3 Signing: m message requires 30 multiplications to be signed using classical Dilithium 3. The average number of multiplications is $30 \cdot 5 = 150$ due to the failure rate. A total of $150m$ multiplications are expected for m messages.

In Algorithm 6, step 15, the batch signature generation is carried out for five signatures. With a probability of approximately 0.196 for a signature candidate to be correct for Dilithium 3, it is assumed that at least one of the five signatures is valid. A total of $100 \cdot (m - 4)$ multiplication operations are needed for the batch operation as a result of computations similar to those in batch Dilithium 2. In order to sign the final four messages using the standard method, a total of $100m+200$ multiplications are needed for batch Dilithium 3. Table 12 provides details about the number of messages, the number of multiplications needed for the batch or classical technique, and the improvement rates achieved with batch Dilithium 3. The formula for calculating the improvement rate is $(150m-(100m+200))\cdot 100/150m$. This rate converges to 33.3% as the number of messages increases. Figure 6 illustrates how the number of multiplications needed varies based on the number of messages in Batch Dilithium 3 and Classical Dilithium 3. Figure 7 illustrates how the improvement rate (%) offered by the batch algorithm differs based on the number of messages when compared to the classical version.

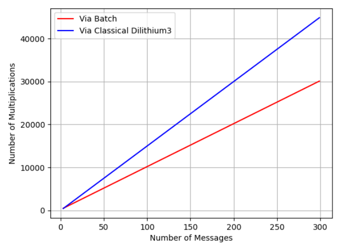

Fig. 6. Variation of the number of multiplications required according to the number of messages for Batch Dilithium 3 and Classical Dilithium 3

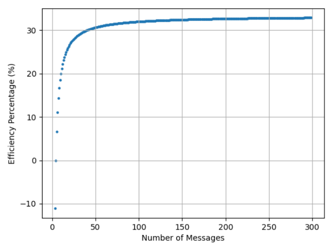

Fig. 7. Variation of the improvement rate (%) provided by the batch algorithm compared to the classical version depending on the number of messages for Dilithium 3

# 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

Table 12. Variation of the number of multiplications required for batch and classical use of Dilithium 3 signing, and the improvement rates provided by batch method according to the number of messages

Batch Dilithium 5 Signing: Assume that Dilithium 5 is used to sign m messages individually using the standard approach. 56 multiplications are required to sing for each message. $56\cdot 4=224$ multiplications are carried out with the assumption that a valid signature needs four repetitions. There are $224m$ multiplications needed for m messages.

The 15^{th} phase of Algorithm 6 is carried out for four signatures, the same as in Batch Dilithium 2. At least one of the four signatures is expected to be valid because the probability that a signature candidate is valid for Dilithium 5 is roughly 0.26. $154\cdot(m-3)$ multiplication operations are needed for the batch operation as a result of computations equivalent to those in batch Dilithium 2 and 3. In order to sign all three remaining messages using the standard method, $154m+210$ multiplications are required for batch Dilithium 5. Table 13 shows the number of messages, the number of multiplications needed for the batch or classical technique, and the improvement rates attained with batch Dilithium 5.

# 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

Table 13. Variation of the number of multiplications required for batch and classical use of Dilithium 5 signing, and the improvement rates provided by batch method according to the number of messages

The formula for calculating the improvement rate is $(224m - (154m + 210)) \cdot 100/224m$. This rate converges to 31.5% as the number of messages increases. Figure 8 illustrates how the number of multiplications required changes according to the number of messages in Batch Dilithium 5 and Classical Dilithium 5. Figure 9 displays how the improvement rate (%) offered by the batch algorithm differs depending on the number of messages when compared to the classical version.

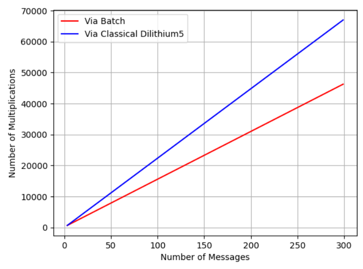

Fig. 8. Variation of the number of multiplications required according to the number of messages for Batch Dilithium 5 and Classical Dilithium 5

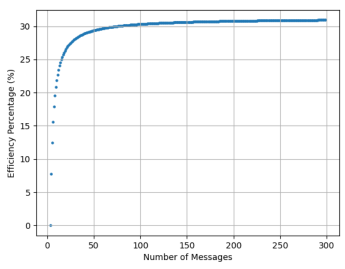

Fig. 9. Variation of the improvement rate (%) provided by the batch algorithm compared to the classical version depending on the number of messages for Dilithium 5

Implementation Results In order to compare the batch implementations with the reference, twenty random messages are generated. The CPU cycle counting reference implementation is used to sign each message separately. Following that, similar speed tests are performed, and the same messages are signed using the batch method. Cycle counts are collected using the single core of the Intel Core i7-8700 processor.

The cycle counts for only the multiplication operation of signing 20 random messages using Classical Dilithium and Batch Dilithium are provided in Table 14. 17 messages are signed using Batch Dilithium, and the amount of the CPU cycles is obtained. Classical Dilithium is used to sign the remaining three messages. Classical Dilithium is also used to sign the same 20 messages. The cycle counts of signing 17 messages for Classical Dilithium are measured in order to provide an exact comparison.

The test results for Dilithium's three security levels are displayed in Table 15. Strassen's algorithm and Rosowski's commutative algorithm are used in the implementation of Dilithium 2. The latter produces better results because of its better multiplicative complexity. Only Rosowski's approach is used in the implementation of Dilithium 3 and Dilithium 5.

Dilithium Signature Generation (Only the Multiplication Stage)				
Algorithm	**Batch Size**	**Reference**	**Batch**	**Improvement (%)**
Dilithium 2	4	2555202	1532297 (via [21])	40.03
Dilithium 2	4	2555202	1809040 (via [26])	29.20
Dilithium 3	5	4723348	3331311 (via [21])	29.47
Dilithium 5	4	5715468	4914677 (via [21])	14.01

Table 14. 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				
Algorithm	**Batch Size**	**Reference**	**Batch**	**Improvement (%)**
Dilithium 2	4	31382991	20645253 (via [21])	34.22
Dilithium 2	4	31382991	20991482 (via [26])	33.11
Dilithium 3	5	50128969	41407391 (via [21])	17.40
Dilithium 5	4	46078440	41833217 (via [21])	10.15

Table 15. 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.

4 Efficient Batch Dilithium Verification From a Single User

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

Algorithm 7 provides the specifics of the Batch Dilithium Verification algorithm, which is based on Algorithm 3. 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 public key of the signer. As a result, it calculates whether the signatures are valid or not.

Algorithm 7 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 $\sigma_i$ are valid or not, where $i = 0, 1, \ldots, m-1$
1: $\hat{\boldsymbol{A}} \in R_q^{k \times l} := ExpandA(\rho)$        ▷ $\boldsymbol{A}$ is generated in NTT representation as $\hat{\boldsymbol{A}}$
2: for $i = 0, 1, \ldots, m-1$ do
3:     $\mu_i \in \{0,1\}^{512} := H(H(\rho \| \boldsymbol{t}_1) \| M_i)$
4:     $c_i := SampleInBall(\tilde{c}_i)$
5: for $i = 0, 1, \ldots, \lfloor m/p \rfloor - 1$ do
6:     $\hat{\boldsymbol{Z}}_i :=$ $(l \times p)$ matrix whose columns are $NTT(\boldsymbol{z}_{pi}), NTT(\boldsymbol{z}_{pi+1}) \ldots, NTT(\boldsymbol{z}_{pi+(p-1)})$
7:     $\hat{\boldsymbol{K}}_i = \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{Z}}_i$ whose columns are $\hat{\boldsymbol{k}}_{pi}, \hat{\boldsymbol{k}}_{pi+1}, \ldots, \hat{\boldsymbol{k}}_{pi+(p-1)}$
8: for $i = \lfloor m/p \rfloor, \ldots, m-1$ do        ▷ If $0 \not\equiv (m \mod p)$
9:     $\hat{\boldsymbol{k}}_i = \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{z}}_i$
10: $status_i = 1$ where $i = 0, 1, \ldots, m-1$
11: for $i = 0, 1, \ldots, m-1$ do
12:     $\boldsymbol{w}_i' := UseHint_q(\boldsymbol{h}_i, NTT^{-1}(\hat{\boldsymbol{k}}_i - NTT(\boldsymbol{c}_i) \cdot NTT(\boldsymbol{t_1} \cdot \boldsymbol{2^d})), 2\gamma_2)$
13:     $status_i =$ [$\| \boldsymbol{z}_i \|_\infty < \gamma_1 - \beta$] and [$\tilde{c}_i = H(\mu_i \| \boldsymbol{w_i'})$] and [# of 1's in $\boldsymbol{h}_i$ is $\leq \omega$]
14: return $status_i$ where $i = 0, 1, \ldots, m-1$
```

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_is and $\tilde{c}_i$s are used, μ_is and c_is 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 3 into matrix-matrix multiplication as seen in step 7 of Algorithm 7. 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 m/p many matrix-matrix multiplications are performed. For the remaining messages, matrix-vector multiplication is performed as in the original algorithm. Thus, the $\hat{\boldsymbol{k}}?i$s are obtained. Then, $\boldsymbol{w_i'}$s and $status_i$s, which are unique for each signature, are calculated. Finally, the $status_i$s are given as output. They contain information about whether the signatures of each message are valid or not.

4.1 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 3.1, in order for the matrix-matrix multiplication contained in 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 6, 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 [21] and [26] 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.

Depending on the efficiency wanted to be achieved, different p values and also suitable, efficient matrix-matrix multiplication algorithms can be preferred in Algorithm 7. 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:

$$(C \cdot m - [B \cdot \lfloor m/p \rfloor + C \cdot (m - \lfloor m/p \rfloor \cdot p)]) \cdot 100/(C \cdot m) \tag{3}$$

Table 16 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.

	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

Table 16. Required number of multiplications and improvement rates (%) for 20 signatures.

4.2 Results for Batch Dilithium Verification

The results of the batch approach for Dilithium verification are explained in this section.

Matrix Multiplications and Arithmetic Complexity Analysis Due to the fact that they have similar matrix sizes and ring properties, the approach described in Section 3.3 for the batch Dilithium signing algorithm is also applied for batch Dilithium verification.

Batch Dilithium 2 Verification: Batch number p is selected as 4 for Batch Dilithium 2 as an example. In this case, in Algorithm 7 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 [21] 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:

$$46 \cdot \lfloor m/p \rfloor + 16 \cdot (m - \lfloor m/p \rfloor \cdot p) \tag{4}$$

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 17.

# 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

Table 17. Variation of the number of multiplications required for batch and classical use of Dilithium 2 verification (Kyber1024 encryption), and the improvement rates provided by batch method according to the number of signatures (messages)

As it can be observed in Table 17, 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 10 illustrates how the number of multiplications needed varies based on the number of signatures in Batch Dilithium 2 and Classical Dilithium 2. Figure 11 illustrates how the improvement rate (%) provided by the batch algorithm varies based on the number of signatures when compared to the classical version.

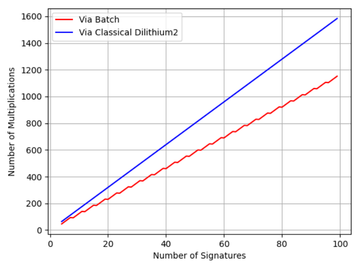

Fig. 10. Variation of the number of multiplications required according to the number of signatures for Batch Dilithium 2 and Classical Dilithium 2 Verification

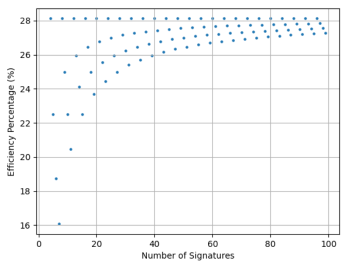

Fig. 11. Variation of the improvement rate (%) provided by the batch algorithm compared to the classical version depending on the number of signatures for Dilithium 2 Verification

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, [21] 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:

$$100 \cdot \lfloor m/p \rfloor + 30 \cdot (m - \lfloor m/p \rfloor \cdot p) \tag{5}$$

$30m$ multiplications are needed to validate a m signature using classical Dilithium 3. Table 18 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 18 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 12 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 13 shows how the improvement rate (%) obtainable via the batch algorithm changes depending on 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

Table 18. Variation of the number of multiplications required for batch and classical use of Dilithium 3 verification, and the improvement rates provided by batch method according to the number of signatures

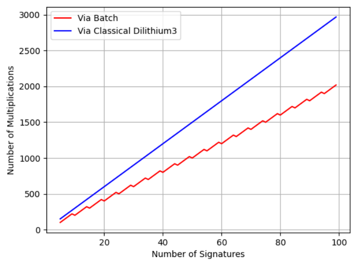

Fig. 12. Variation of the number of multiplications required according to the number of signatures for Batch Dilithium 3 and Classical Dilithium 3 Verification

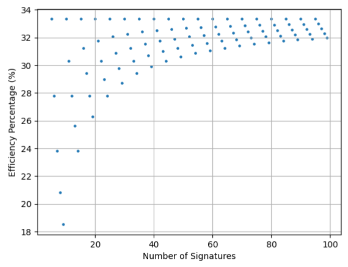

Fig. 13. Variation of the improvement rate (%) provided by the batch algorithm compared to the classical version depending on the number of signatures for Dilithium 3 Verification

Table 18 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 12 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 13 shows how the improvement rate (%) obtainable via the batch algorithm changes depending on the number of signatures.

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 [21] is used for the $\hat{\boldsymbol{A}}_{(8\times 7)} \cdot \hat{\boldsymbol{Z}}_{(7\times 4)}$ 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:

$$154 \cdot \lfloor m/p \rfloor + 56 \cdot (m - \lfloor m/p \rfloor \cdot p) \tag{6}$$

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 19.

# 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

Table 19. 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

The improvement rate is equal to 31.25% if $m \equiv 0 \mod p$, as Table 19 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 14. Comparing the batch algorithm to the classical version, Figure 15 shows how the improvement rate (%) changes depending on the number of signatures.

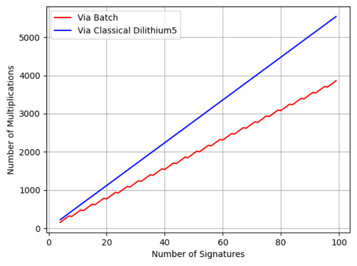

Fig. 14. Variation of the number of multiplications required according to the number of signatures for Batch Dilithium 5 and Classical Dilithium 5 Verification

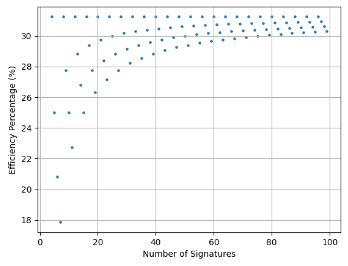

Fig. 15. Variation of the improvement rate (%) provided by the batch algorithm compared to the classical version depending on the number of signatures for Dilithium 5 Verification

Implementation Results 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 20, 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 [26] and Rostowski's method [21] are used as efficient matrix-matrix multiplication algorithms. For Dilithium 3 and 5, only Rosowski's method is performed.

Dilithium Signature Verification from a Single User				
Algorithm	**Batch Size**	**Reference**	**Batch**	**Improvement (%)**
Dilithium 2	4	5549265	2781154 (using [21])	49.88
Dilithium 2	4	5549265	2850412 (using [26])	48.63
Dilithium 3	5	8935674	3877687 (using [21])	56.60
Dilithium 5	4	14988354	5833909 (using [21])	61.08

Table 20. CPU Cycle counts obtained by verifying 20 signatures ($m = 20$) 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.

5 Efficient Batch Kyber Encryption to a Single User

This section explains the proposed batch version of Kybers's encryption and encapsulation for a single user.

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 8. 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 4 is converted to matrix-matrix multiplication, as seen in step 17 of Algorithm 8. 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 8 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 $c_i$
1: $\hat{\boldsymbol{t}} := Decode_{12}(pk)$
2: $\rho = pk + 12 \cdot k \cdot n/8$
3: for $i = 0, 1, \ldots, m-1$ do
4:     for $j = 0, 1, \ldots, m-1$ do
5:         $\hat{\boldsymbol{A}}^T[i][j] := Parse(XOF(\rho, i, j))$        ▷ Generate $\hat{\boldsymbol{A}} \in R_q^{k \times k}$ in NTT domain.
6: for $j = 0, 1, \ldots, m-1$ do
7:     $N = 0$
8:     for $i = 0, 1, \ldots, k-1$ do
9:         $\boldsymbol{r}_j[i] := CBD_{\eta_1}(PRF(r_j, N))$
10:        $N = N + 1$
11:    for $i = 0, 1, \ldots, k-1$ do
12:        $\boldsymbol{e}'_j[i] := CBD_{\eta_2}(PRF(r_j, N))$  $N = N + 1$
13:    $e''_j := CBD_{\eta_2}(PRF(r_j, N))$
14:    $\hat{\boldsymbol{r}}_j := NTT(\boldsymbol{r}_j)$
15: for $i = 0, 1, \ldots, \lfloor m/p \rfloor - 1$ do
16:    $\hat{\boldsymbol{R}}_i :=$ $(k \times p)$ matrix whose columns are $NTT(\boldsymbol{r}_{pi}), NTT(\boldsymbol{r}_{pi+1}) \ldots, NTT(\boldsymbol{r}_{pi+(p-1)})$
17:    $\hat{\boldsymbol{K}}_i = \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{R}}_i$ whose columns are $\hat{\boldsymbol{k}}_{pi}, \hat{\boldsymbol{k}}_{pi+1}, \ldots, \hat{\boldsymbol{k}}_{pi+(p-1)}$
18: for $i = \lfloor m/p \rfloor \cdot p, \ldots, m-1$ do        ▷ If $0 \not\equiv (m \mod p)$
19:    $\hat{\boldsymbol{k}}_i = \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{r}}_i$
20: for $i = 0, 1, \ldots, m-1$ do
21:    $\boldsymbol{u}_i := NTT^{-1}(\hat{\boldsymbol{k}}_i) + \boldsymbol{e}'_i$
22:    $v := NTT^{-1}(\hat{\boldsymbol{t}}^T \cdot \hat{\boldsymbol{r}}_i) + e''_i + Decompress_q(Decode_1(M_i), 1)$
23:    $c' := Encode_{d_u}(Compress_q(\boldsymbol{u}_i, d_u))$
24:    $c'' := Encode_{d_v}(Compress_q(v_i, d_v))$
25:    $c_i = (c'_i \| c''_i)$
26: 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:

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

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

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

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

5.1 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 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 21. [26] is used in Kyber512, [21] in Kyber768. Also, both [21] and [26] 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 8, depending on the efficiency that is needed.

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. 3 provides the improvement rate (%) that batch verification yields, similarly.

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 22 based on the chosen p. [29] or [26] are used in Kyber512 calculations with batch number $p = 2$. For $p > 2$, the number of multiplications for the batch method are obtained via [29].

Security Level	Algorithm	Batch Number	Size of $\hat{\boldsymbol{A}}$	Size of $\hat{\boldsymbol{R}}_i$	Size of $\hat{\boldsymbol{K}}_i$
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)

Table 21. Batch numbers and matrix sizes according to different security levels.

	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

Table 22. Required number of multiplications and improvement rates (%) for 20 messages.

5.2 Results for Batch Kyber Encryption

Matrix Multiplications and Arithmetic Complexity Analysis 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 3.3 is also applicable.

Batch Kyber512 Encryption: As an example, the batch number p for batch Kyber512 encryption is set to 2. In this case, the matrix multiplication $\hat{\boldsymbol{A}}_{(2\times2)} \cdot \hat{\boldsymbol{R}}_{i(2\times2)}$ occurs in step 17 of Algorithm 8. As an efficient matrix multiplication technique, this operation requires 7 multiplications (Appendix D) using Strassen's approach [26]. 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:

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

$4m$ multiplications are needed to encrypt m messages using classical Kyber512. Table 23 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.

# 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

Table 23. Variation of the number of multiplications required for batch and classical use of Kyber512 encryption, and the improvement rates provided by batch method according to the number of messages

If $m \equiv 0 \mod p$, then the improvement rate is equal to 12.50%, as Table 23 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 16. In comparison with the classical version, Figure 17 shows how the improvement rate (%) available through the batch method changes depending on the number of messages.

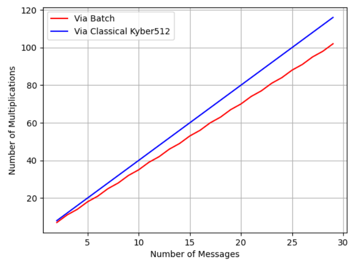

Fig. 16. Variation of the number of multiplications required according to the number of messages for Batch Kyber512 and Classical Kyber512 Encryption

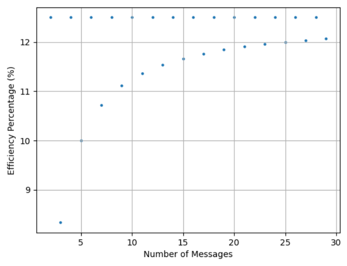

Fig. 17. Variation of the improvement rate (%) provided by the batch algorithm compared to the classical version depending on the number of messages for Kyber512 Encryption

Batch Kyber768 Encryption: In this case, for batch Kyber768 encryption, batch number p is assigned to 4, and matrix multiplication in Algorithm 8 becomes $\hat{\boldsymbol{A}}_{(3\times3)} \cdot \hat{\boldsymbol{R}}_{i\,(3\times4)}$. With Rosowski's approach [21], this operation involves 28 multiplications (Appendix E). 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:

$$28 \cdot \lfloor m/p \rfloor + 9 \cdot (m - \lfloor m/p \rfloor \cdot p) \tag{8}$$

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 24.

# 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

Table 24. Variation of the number of multiplications required for batch and classical use of Kyber768 encryption, and the improvement rates provided by batch method according to the number of messages

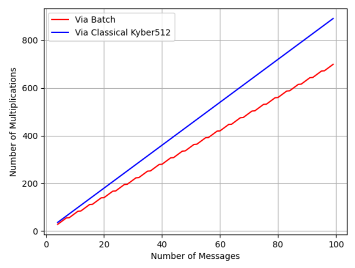

Fig. 18. Variation of the number of multiplications required according to the number of messages for Batch Kyber512 and Classical Kyber768 Encryption

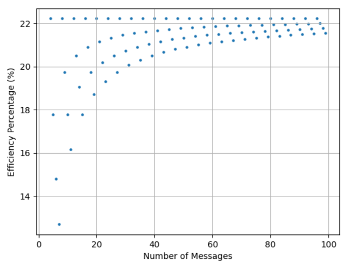

Fig. 19. Variation of the improvement rate (%) provided by the batch algorithm compared to the classical version depending on the number of messages for Kyber768 Encryption

Table 24 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 18 illustrates the relationship between the number of messages in the batch and the classical Kyber768 and the number of multiplications needed. Figure 19 illustrates how the improvement rate (%) made possible by the batch approach varies based on the number of messages when compared to the classical structure.

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 (4) since the matrix sizes and message grouping procedure used in the for loop are identical to the batch Dilithium2 verification. Table 17 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 20 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 21 illustrates how the improvement rate (%) offered by the batch algorithm changes depending on the number of messages.

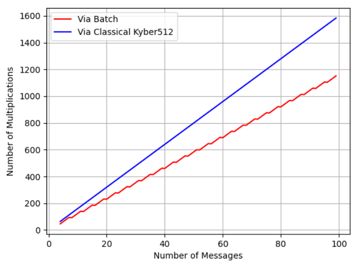

Fig. 20. Variation of the number of multiplications required according to the number of messages for Batch Kyber512 and Classical Kyber1024 Encryption

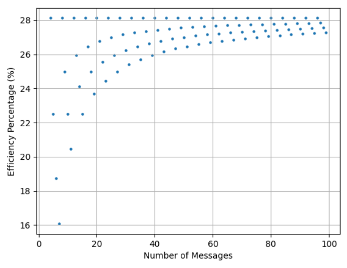

Fig. 21. Variation of the improvement rate (%) provided by the batch algorithm compared to the classical version depending on the number of messages for Kyber1024 Encryption

Implementation Results Twenty random messages are encrypted using the batch and the classical Kyber512, 768, and 1024 encryption algorithms. 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 26. Table 25 gives the cycle counts of only the multiplication process of encrypting 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 [21] and Strassen's method [26]. While [29] is preferred for Kyber512, [21] is used in Kyber768.

Kyber Encryption (Only the Multiplication Stage)				
Algorithm	**Batch Size**	**Reference**	**Batch**	**Improvement (%)**
Kyber 512	2	322509	249408 (via [29])	22.67
Kyber 768	4	614922	443653 (via [21])	27.85
Kyber 1024	4	1069849	737093 (via [21])	31.10
Kyber 1024	4	1069849	830175 (via [26])	22.40

Table 25. 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				
Algorithm	**Batch Size**	**Reference**	**Batch**	**Improvement (%)**
Kyber512	2	2706716	2102105 (via [26])	22.34
Kyber768	4	3941892	2992896 (via [21])	24.07
Kyber1024	4	5766383	4102297 (via [26])	28.86
Kyber1024	4	5766383	3988733 (via [21])	30.83

Table 26. 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.

6 Conclusion

This study proposes efficient batch signature generation and verification with the Dilithium algorithm, as well as batch encryption for a single user with Kyber. The batch Dilithium signing algorithm is designed to enable many messages to be signed simultaneously. According to the repetition numbers of Dilithium 2, Dilithium 3, and Dilithium 5 signing algorithms, the column sizes of the first matrix (i.e., batch numbers) are determined as 4, 5, and 4, respectively. The Batch Dilithium algorithm allows users to efficiently and simultaneously verify multiple signatures from a single user. The same batch numbers with the batch Dilithium signature are preferred for the implementation of batch Dilithium verification. Batch Kyber encryption is another method that is proposed in this paper, and it is used to encrypt several messages in groups for a user. Batch numbers are chosen as 2, 4, and 4, for illustration, in batch Kyber encryption. Dilithium's and Kyber's matrix-vector multiplications have been converted to matrix-matrix multiplications. These transformations allow us to employ efficient matrix-matrix multiplications for many signature generation, signature verification, and message encryption. The matrix dimensions that provide improvements are determined, and efficient multiplication methods, such as commutative matrix multiplication by Rosowski and Strassen's multiplication algorithms, are integrated into Dilithium and Kyber. As a result of those multiplications, the arithmetic complexities of generating many signatures are enhanced up to 28.1% for Dilithium 2, 33.3% while verifying multiple signatures with this method provides 28.13%, 33.33%, and 31.25% for Dilithium 3, and 31.5% for Dilithium 5. Moreover, we implement the proposed batch signature generation by signing 20 messages using the efficient matrix-matrix multiplication algorithms for three security levels and obtain improvements in terms of CPU cycle counts, which are 34.22% for Dilithium 2, 17.40% for Dilithium 3, and 10.15% for Dilithium 5. Using batch Dilithium, these 20 signatures are verified, and 49.88%, 56.60%, and 61.08% improvements are observed regarding CPU cycle counts. 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 noted for three security levels when Kyber Batch Encryption is implemented using efficient multiplication algorithms.

The full matrix is represented rather than just one vector since the batch method employs matrix-matrix multiplication rather than matrix-vector multiplication. The amount of stack memory used is increased while calculating and storing the linear combinations contained in the multiplication method's structure and when extracting the matrix elements through addition and subtraction. With proper implementation strategies (calculating linear combinations with loops, parallelization techniques, etc.), it can be minimized. Using matrices for the multiplication operations instead of vectors requires dynamic memory allocations for the operations to be calculated properly in the algorithm represented by the entire function. As expected, this results in a trade-off between time and memory. Our calculations indicate that it is not significant since the matrix is generated by including a few extra vectors. Also, the proposed method can be implemented using SIMD instructions such as AVX2. As expected, this results in a trade-off between time and memory. Our calculations indicate that it is not significant since the matrix is generated by including a few extra vectors. It should be mentioned that SIMD instructions such as AVX2 can be used to achieve the suggested strategy. This method can significantly enhance performance. Therefore, it might be considered for future work.

Availability of the software: All source codes are available at https://github.com/denizzzture/Efficient-Batch-Dilithium and https://github.com/denizzzture/Efficient-Batch-Kyber.

Acknowledgements We would like to extend our heartfelt gratitude to Sevim Seda Odacıoğlu for her invaluable assistance in the implementation phase of this paper.

References

1. Aguilar-Melchor, C., Albrecht, M.R., Bailleux, T., Bindel, N., Howe, J., Hülsing, A., Joseph, D., Manzano, M.: Batch signatures, revisited. In: Cryptographers' Track at the RSA Conference. pp. 163–186. Springer (2024)
2. Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Miller, C., Moody, D., Peralta, R., et al.: Status report on the third round of the nist post-quantum cryptography standardization process. US Department of Commerce, NIST (2022)
3. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-kyber algorithm specifications and supporting documentation. NIST PQC Round **2**(4), 1–43 (2019)
4. Benjamin, D.: Batch signing for tls. Internet Engineering Task Force, Internet-Draft draft-davidben-tls-batch-signing-02 (2019)
5. Cenk, M., Hasan, M.A.: On the arithmetic complexity of strassen-like matrix multiplications. Journal of Symbolic Computation **80**, 484–501 (2017)
6. Chang, Y.S., Wu, T.C., Huang, S.C.: Elgamal-like digital signature and multisignature schemes using self-certified public keys. Journal of Systems and Software **50**(2), 99–105 (2000)

7. Cheon, J.H., Coron, J.S., Kim, J., Lee, M.S., Lepoint, T., Tibouchi, M., Yun, A.: 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)
8. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-dilithium algorithm specifications and supporting documentation (version 3.1) (2021)
9. Elgamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**(4), 469–472 (1985). https://doi.org/10.1109/TIT.1985.1057074
10. Ferrara, A.L., Green, M., Hohenberger, S., Pedersen, M.Ø.: Practical short signature batch verification. In: Cryptographers' Track at the RSA Conference. pp. 309–324. Springer (2009)
11. Fiat, A.: Batch rsa. In: Advances in Cryptology—CRYPTO'89 Proceedings 9. pp. 175–185. Springer (1990)
12. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z., et al.: Falcon: Fast-fourier lattice-based compact signatures over ntru specification v1.2. Submission to the NIST's post-quantum cryptography standardization process (2020)
13. Hwang, S.J., Lee, Y.H.: Repairing elgamal-like multi-signature schemes using self-certified public keys. Applied mathematics and computation **156**(1), 73–83 (2004)
14. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ecdsa). International journal of information security **1**, 36–63 (2001)
15. Kittur, A.S., Pais, A.R.: Batch verification of digital signatures: approaches and challenges. Journal of information security and applications **37**, 15–27 (2017)
16. Moriarty, K., Kaliski, B., Jonsson, J., Rusch, A.: PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017 (Nov 2016). https://doi.org/10.17487/RFC8017, https://www.rfc-editor.org/info/rfc8017
17. NIST: Module-lattice-based digital signature standard. Tech. Rep. Federal Information Processing Standards Publications (FIPS PUBS) 204, August 13, 2024, U.S. Department of Commerce, Washington, D.C. (2024). https://doi.org/10.6028/NIST.FIPS.204
18. NIST: Module-lattice-based key-encapsulation mechanism standard. Tech. Rep. Federal Information Processing Standards Publications (FIPS PUBS) 203, August 13, 2024, U.S. Department of Commerce, Washington, D.C. (2024). https://doi.org/10.6028/NIST.FIPS.203
19. Paquin, C., Stebila, D., Tamvada, G.: 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)
20. Pavlovski, C., Boyd, C.: Efficient batch signature generation using tree structures. In: International workshop on cryptographic techniques and E-commerce, CrypTEC. vol. 99, pp. 70–77. Citeseer (1999)
21. Rosowski, A.: Fast commutative matrix algorithms. Journal of Symbolic Computation **114**, 302–321 (2023). https://doi.org/https://doi.org/10.1016/j.jsc.2022.05.002, https://www.sciencedirect.com/science/article/pii/S0747717122000499
22. Scientific, L.L.: Enhancing cloud security based on the kyber key encapsulation mechanism. Journal Of Theoretical And Applied Information Technology **102**(4) (2024)
23. Selvakumar, S., Ahilan, A., Ben Sujitha, B., Muthukumaran, N.: Crystals kyber cryptographic algorithm for efficient iot d2d communication. Wireless Networks pp. 1–18 (2024)
24. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review **41**(2), 303–332 (1999)
25. Sikeridis, D., Kampanakis, P., Devetsikiotis, M.: Post-quantum authentication in tls 1.3: a performance study. Cryptology ePrint Archive (2020)
26. Strassen, V., et al.: Gaussian elimination is not optimal. Numerische mathematik **13**(4), 354–356 (1969)
27. Tanwar, S., Kumar, A.: An efficient and secure identity based multiple signatures scheme based on rsa. Journal of Discrete Mathematical Sciences and Cryptography **22**(6), 953–971 (2019)
28. Van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: 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)

29. Winograd, S.: A new algorithm for inner product. IEEE Transactions on Computers **100**(7), 693–694 (1968)
30. Winograd, S.: On multiplication of 2× 2 matrices. Linear algebra and its applications **4**(4), 381–388 (1971)

A Efficient Multiplication Formulas for Dilithium 2 and Kyber1024

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

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

A.1 Strassen-Method

To compute the matrix C, applying any Strassen-like [26] matrix multiplication method would provide efficiency. To obtain the matrix C, firstly, unique multiplications p_i are computed as follows:

$$\begin{aligned}
p_1 &= (a_{11} + a_{33} + a_{22} + a_{44}) \cdot (b_{11} + b_{33} + b_{22} + b_{44}) \\
p_2 &= (a_{21} + a_{43} + a_{22} + a_{44}) \cdot (b_{11} + b_{33}) \\
p_3 &= (a_{11} + a_{33}) \cdot (b_{12} + b_{34} - b_{22} - b_{44}) \\
p_4 &= (a_{22} + a_{44}) \cdot (b_{21} + b_{43} - b_{11} - b_{33}) \\
p_5 &= (a_{11} + a_{33} + a_{12} + a_{34}) \cdot (b_{22} + b_{44}) \\
p_6 &= (a_{21} + a_{43} - a_{11} - a_{33}) \cdot (b_{11} + b_{33} + b_{12} + b_{34}) \\
p_7 &= (a_{12} + a_{34} - a_{22} - a_{44}) \cdot (b_{21} + b_{43} + b_{22} + b_{44})
\end{aligned}$$

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

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

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

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

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

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

After computing all of the linear combinations, one can obtain the element of the matrix C as follows:

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

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

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

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

Finally, the matrix 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 and Kyber belong to different commutative rings. Therefore, we derive the following by using [21] to obtain the matrix C.

Firstly, unique multiplications p_i are obtained as follows:

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

$$p_{21} = a_{42} \cdot (b_{21} - a_{41})$$
$$p_{22} = a_{44} \cdot (b_{41} - a_{43})$$
$$p_{23} = (a_{11} + b_{22}) \cdot (a_{12} + b_{11} + b_{12})$$
$$p_{25} = (a_{13} + b_{42}) \cdot (a_{14} + b_{31} + b_{32})$$
$$p_{24} = (a_{11} + b_{23}) \cdot (a_{12} + b_{11} + b_{13})$$
$$p_{26} = (a_{13} + b_{43}) \cdot (a_{14} + b_{31} + b_{33})$$
$$p_{27} = (a_{11} + b_{24}) \cdot (a_{12} + b_{11} + b_{14})$$
$$p_{28} = (a_{13} + b_{44}) \cdot (a_{14} + b_{31} + b_{34})$$
$$p_{29} = (a_{21} + b_{22}) \cdot (a_{22} + b_{11} + b_{12})$$
$$p_{30} = (a_{23} + b_{42}) \cdot (a_{24} + b_{31} + b_{32})$$
$$p_{31} = (a_{21} + b_{23}) \cdot (a_{22} + b_{11} + b_{13})$$
$$p_{32} = (a_{23} + b_{43}) \cdot (a_{24} + b_{31} + b_{33})$$
$$p_{33} = (a_{21} + b_{24}) \cdot (a_{22} + b_{11} + b_{14})$$
$$p_{34} = (a_{23} + b_{44}) \cdot (a_{24} + b_{31} + b_{34})$$
$$p_{35} = (a_{31} + b_{22}) \cdot (a_{32} + b_{11} + b_{12})$$
$$p_{36} = (a_{33} + b_{42}) \cdot (a_{34} + b_{31} + b_{32})$$
$$p_{37} = (a_{31} + b_{23}) \cdot (a_{32} + b_{11} + b_{13})$$
$$p_{38} = (a_{33} + b_{43}) \cdot (a_{34} + b_{31} + b_{33})$$
$$p_{39} = (a_{31} + b_{24}) \cdot (a_{32} + b_{11} + b_{14})$$
$$p_{40} = (a_{33} + b_{44}) \cdot (a_{34} + b_{31} + b_{34})$$
$$p_{41} = (a_{41} + b_{22}) \cdot (a_{42} + b_{11} + b_{12})$$
$$p_{42} = (a_{43} + b_{42}) \cdot (a_{44} + b_{31} + b_{32})$$
$$p_{43} = (a_{41} + b_{23}) \cdot (a_{42} + b_{11} + b_{13})$$
$$p_{44} = (a_{43} + b_{43}) \cdot (a_{44} + b_{31} + b_{33})$$
$$p_{45} = (a_{41} + b_{24}) \cdot (a_{42} + b_{11} + b_{14})$$
$$p_{46} = (a_{43} + b_{44}) \cdot (a_{44} + b_{31} + b_{34})$$

After computing all of the linear combinations, one can obtain the element of the matrix C as follows:

$$c_{11} = p_1 + p_2 + p_{15} + p_{16}$$
$$c_{12} = p_{23} + p_{24} - p_1 - p_2 - p_9 - p_{10}$$
$$c_{13} = p_{25} + p_{26} - p_1 - p_2 - p_{11} - p_{12}$$
$$c_{14} = p_{27} + p_{28} - p_1 - p_2 - p_{13} - p_{14}$$

$$c_{21} = p_3 + p_4 + p_{17} + p_{18}$$
$$c_{22} = p_{29} + p_{30} - p_3 - p_4 - p_9 - p_{10}$$
$$c_{23} = p_{31} + p_{32} - p_3 - p_4 - p_{11} - p_{12}$$
$$c_{24} = p_{33} + p_{34} - p_3 - p_4 - p_{13} - p_{14}$$

$$c_{31} = p_5 + p_6 + p_{19} + p_{20}$$
$$c_{32} = p_{35} + p_{36} - p_5 - p_6 - p_9 - p_{10}$$
$$c_{33} = p_{37} + p_{38} - p_5 - p_6 - p_{11} - p_{12}$$
$$c_{34} = p_{39} + p_{40} - p_5 - p_6 - p_{13} - p_{14}$$

$$c_{41} = p_7 + p_8 + p_{21} + p_{22}$$
$$c_{42} = p_{41} + p_{42} - p_7 - p_8 - p_9 - p_{10}$$
$$c_{43} = p_{43} + p_{44} - p_7 - p_8 - p_{11} - p_{12}$$
$$c_{44} = p_{45} + p_{46} - p_7 - p_8 - p_{13} - p_{14}$$

Finally, the matrix 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.

B Efficient Multiplication Formula for Dilithium 3

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

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

By [21], the matrix C can be obtained as follows:

$$\begin{aligned}
p_1 &= (a_{11} + b_{21}) \cdot (a_{12} + b_{12}) \\
p_2 &= (a_{21} + b_{21}) \cdot (a_{22} + b_{12}) \\
p_3 &= (a_{31} + b_{21}) \cdot (a_{32} + b_{12}) \\
p_4 &= (a_{41} + b_{21}) \cdot (a_{42} + b_{12}) \\
p_5 &= (a_{51} + b_{21}) \cdot (a_{52} + b_{12}) \\
p_6 &= (a_{61} + b_{21}) \cdot (a_{62} + b_{12}) \\
p_7 &= (a_{11} + b_{31}) \cdot (a_{13} + b_{13}) \\
p_8 &= (a_{21} + b_{31}) \cdot (a_{23} + b_{13}) \\
p_9 &= (a_{31} + b_{31}) \cdot (a_{33} + b_{13}) \\
p_{10} &= (a_{41} + b_{31}) \cdot (a_{43} + b_{13}) \\
p_{11} &= (a_{51} + b_{31}) \cdot (a_{53} + b_{13}) \\
p_{12} &= (a_{61} + b_{31}) \cdot (a_{63} + b_{13}) \\
p_{13} &= (a_{12} + b_{32}) \cdot (a_{13} + b_{23}) \\
p_{14} &= (a_{22} + b_{32}) \cdot (a_{23} + b_{23}) \\
p_{15} &= (a_{32} + b_{32}) \cdot (a_{33} + b_{23}) \\
p_{16} &= (a_{42} + b_{32}) \cdot (a_{43} + b_{23}) \\
p_{17} &= (a_{52} + b_{32}) \cdot (a_{53} + b_{23}) \\
p_{18} &= (a_{62} + b_{32}) \cdot (a_{63} + b_{23}) \\
p_{19} &= a_{11} \cdot (b_{11} - b_{12} - b_{13} - a_{12} - a_{13}) \\
p_{20} &= a_{21} \cdot (b_{11} - b_{12} - b_{13} - a_{22} - a_{23}) \\
p_{21} &= a_{31} \cdot (b_{11} - b_{12} - b_{13} - a_{32} - a_{33}) \\
p_{22} &= a_{41} \cdot (b_{11} - b_{12} - b_{13} - a_{42} - a_{43}) \\
p_{23} &= a_{51} \cdot (b_{11} - b_{12} - b_{13} - a_{52} - a_{53}) \\
p_{24} &= a_{61} \cdot (b_{11} - b_{12} - b_{13} - a_{62} - a_{63}) \\
p_{25} &= a_{12} \cdot (b_{22} - b_{21} - b_{23} - a_{11} - a_{13}) \\
p_{26} &= a_{22} \cdot (b_{22} - b_{21} - b_{23} - a_{21} - a_{23}) \\
p_{27} &= a_{32} \cdot (b_{22} - b_{21} - b_{23} - a_{31} - a_{33}) \\
p_{28} &= a_{42} \cdot (b_{22} - b_{21} - b_{23} - a_{41} - a_{43}) \\
p_{29} &= a_{52} \cdot (b_{22} - b_{21} - b_{23} - a_{51} - a_{53}) \\
p_{30} &= a_{62} \cdot (b_{22} - b_{21} - b_{23} - a_{61} - a_{63})
\end{aligned}$$

$$\begin{aligned}
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})
\end{aligned}$$

$$
\begin{aligned}
p_{61} &= a_{14} \cdot (b_{41} + a_{15}) & p_{81} &= (a_{34} + b_{53}) \cdot (a_{35} + b_{41} + b_{43}) \\
p_{62} &= a_{24} \cdot (b_{41} + a_{25}) & p_{82} &= (a_{44} + b_{53}) \cdot (a_{45} + b_{41} + b_{43}) \\
p_{63} &= a_{34} \cdot (b_{41} + a_{35}) & p_{83} &= (a_{54} + b_{53}) \cdot (a_{55} + b_{41} + b_{43}) \\
p_{64} &= a_{44} \cdot (b_{41} + a_{45}) & p_{84} &= (a_{64} + b_{53}) \cdot (a_{65} + b_{41} + b_{43}) \\
p_{65} &= a_{54} \cdot (b_{41} + a_{55}) & p_{85} &= (a_{14} + b_{54}) \cdot (a_{15} + b_{41} + b_{44}) \\
p_{66} &= a_{64} \cdot (b_{41} + a_{65}) & p_{86} &= (a_{24} + b_{54}) \cdot (a_{25} + b_{41} + b_{44}) \\
p_{67} &= a_{15} \cdot (b_{51} - a_{14}) & p_{87} &= (a_{34} + b_{54}) \cdot (a_{35} + b_{41} + b_{44}) \\
p_{68} &= a_{25} \cdot (b_{51} - a_{24}) & p_{88} &= (a_{44} + b_{54}) \cdot (a_{45} + b_{41} + b_{44}) \\
p_{69} &= a_{35} \cdot (b_{51} - a_{34}) & p_{89} &= (a_{54} + b_{54}) \cdot (a_{55} + b_{41} + b_{44}) \\
p_{70} &= a_{45} \cdot (b_{51} - a_{44}) & p_{90} &= (a_{64} + b_{54}) \cdot (a_{65} + b_{41} + b_{44}) \\
p_{71} &= a_{55} \cdot (b_{51} - a_{54}) & p_{91} &= (a_{14} + b_{55}) \cdot (a_{15} + b_{41} + b_{45}) \\
p_{72} &= a_{65} \cdot (b_{51} - a_{64}) & p_{92} &= (a_{24} + b_{55}) \cdot (a_{25} + b_{41} + b_{45}) \\
p_{73} &= (a_{14} + b_{52}) \cdot (a_{15} + b_{41} + b_{42}) & p_{93} &= (a_{34} + b_{55}) \cdot (a_{35} + b_{41} + b_{45}) \\
p_{74} &= (a_{24} + b_{52}) \cdot (a_{25} + b_{41} + b_{42}) & p_{94} &= (a_{44} + b_{55}) \cdot (a_{45} + b_{41} + b_{45}) \\
p_{75} &= (a_{34} + b_{52}) \cdot (a_{35} + b_{41} + b_{42}) & p_{95} &= (a_{54} + b_{55}) \cdot (a_{55} + b_{41} + b_{45}) \\
p_{76} &= (a_{44} + b_{52}) \cdot (a_{45} + b_{41} + b_{42}) & p_{96} &= (a_{64} + b_{55}) \cdot (a_{65} + b_{41} + b_{45}) \\
p_{77} &= (a_{54} + b_{52}) \cdot (a_{55} + b_{41} + b_{42}) & p_{97} &= b_{52} \cdot (b_{41} + b_{42}) \\
p_{78} &= (a_{64} + b_{52}) \cdot (a_{65} + b_{41} + b_{42}) & p_{98} &= b_{53} \cdot (b_{41} + b_{43}) \\
p_{79} &= (a_{14} + b_{53}) \cdot (a_{15} + b_{41} + b_{43}) & p_{99} &= b_{54} \cdot (b_{41} + b_{44}) \\
p_{80} &= (a_{24} + b_{53}) \cdot (a_{25} + b_{41} + b_{43}) & p_{100} &= b_{55} \cdot (b_{41} + b_{45})
\end{aligned}
$$

After computing all of the linear combinations, one can obtain the element of the matrix C as follows:

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

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

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

$$c_{14} = p_1 + p_7 + p_{40} + p_{52} - p_{37} - p_{38} - p_{58} - p_{59} + p_{85} - p_{61} - p_{99}$$
$$c_{24} = p_2 + p_8 + p_{41} + p_{53} - p_{37} - p_{38} - p_{58} - p_{59} + p_{86} - p_{62} - p_{99}$$
$$c_{34} = p_3 + p_9 + p_{42} + p_{54} - p_{37} - p_{38} - p_{58} - p_{59} + p_{87} - p_{63} - p_{99}$$
$$c_{44} = p_4 + p_{10} + p_{43} + p_{55} - p_{37} - p_{38} - p_{58} - p_{59} + p_{88} - p_{64} - p_{99}$$
$$c_{54} = p_5 + p_{11} + p_{44} + p_{56} - p_{37} - p_{38} - p_{58} - p_{59} + p_{89} - p_{65} - p_{99}$$
$$c_{64} = p_6 + p_{12} + p_{45} + p_{57} - p_{37} - p_{38} - p_{58} - p_{59} + p_{90} - p_{66} - p_{99}$$

$$c_{15} = p_7 + p_{13} + p_{46} + p_{52} - p_{38} - p_{39} - p_{59} - p_{60} + p_{91} - p_{61} - p_{100}$$
$$c_{25} = p_8 + p_{14} + p_{47} + p_{53} - p_{38} - p_{39} - p_{59} - p_{60} + p_{92} - p_{62} - p_{100}$$
$$c_{35} = p_9 + p_{15} + p_{48} + p_{54} - p_{38} - p_{39} - p_{59} - p_{60} + p_{93} - p_{63} - p_{100}$$
$$c_{45} = p_{10} + p_{16} + p_{49} + p_{55} - p_{38} - p_{39} - p_{59} - p_{60} + p_{94} - p_{64} - p_{100}$$
$$c_{55} = p_{11} + p_{17} + p_{50} + p_{56} - p_{38} - p_{39} - p_{59} - p_{60} + p_{95} - p_{65} - p_{100}$$
$$c_{65} = p_{12} + p_{18} + p_{51} + p_{57} - p_{38} - p_{39} - p_{59} - p_{60} + p_{96} - p_{66} - p_{100}$$

C Efficient Multiplication Formula for Dilithium 5

Let $m \geq 4$ messages be signed with Dilithium 5. Then, $A \in R_q^{8\times7}$, $B \in R_q^{7\times4}$ and $A \cdot B = C \in R_q^{8\times4}$. The elements of the matrices $A^{8\times7} \cdot B^{7\times4} = C^{8\times4}$ are described as follows:

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

By [21], the matrix C can be obtained as follows:

$$\begin{aligned}
p_1 &= (a_{11} + b_{21}) \cdot (a_{12} + b_{12}) & p_{29} &= a_{11} \cdot (b_{11} - b_{12} - b_{13} - a_{12} - a_{13}) \\
p_2 &= (a_{21} + b_{21}) \cdot (a_{22} + b_{12}) & p_{30} &= a_{21} \cdot (b_{11} - b_{12} - b_{13} - a_{22} - a_{23}) \\
p_3 &= (a_{31} + b_{21}) \cdot (a_{32} + b_{12}) & p_{31} &= a_{31} \cdot (b_{11} - b_{12} - b_{13} - a_{32} - a_{33}) \\
p_4 &= (a_{41} + b_{21}) \cdot (a_{42} + b_{12}) & p_{32} &= a_{41} \cdot (b_{11} - b_{12} - b_{13} - a_{42} - a_{43}) \\
p_5 &= (a_{51} + b_{21}) \cdot (a_{52} + b_{12}) & p_{33} &= a_{51} \cdot (b_{11} - b_{12} - b_{13} - a_{52} - a_{53}) \\
p_6 &= (a_{61} + b_{21}) \cdot (a_{62} + b_{12}) & p_{34} &= a_{61} \cdot (b_{11} - b_{12} - b_{13} - a_{62} - a_{63}) \\
p_7 &= (a_{71} + b_{21}) \cdot (a_{72} + b_{12}) & p_{35} &= a_{71} \cdot (b_{11} - b_{12} - b_{13} - a_{72} - a_{73}) \\
p_8 &= (a_{81} + b_{21}) \cdot (a_{82} + b_{12}) & p_{36} &= a_{81} \cdot (b_{11} - b_{12} - b_{13} - a_{82} - a_{83}) \\
p_9 &= (a_{11} + b_{31}) \cdot (a_{13} + b_{13}) & p_{37} &= a_{12} \cdot (b_{22} - b_{21} - b_{23} - a_{11} - a_{13}) \\
p_{10} &= (a_{21} + b_{31}) \cdot (a_{23} + b_{13}) & p_{38} &= a_{22} \cdot (b_{22} - b_{21} - b_{23} - a_{21} - a_{23}) \\
p_{11} &= (a_{31} + b_{31}) \cdot (a_{33} + b_{13}) & p_{39} &= a_{32} \cdot (b_{22} - b_{21} - b_{23} - a_{31} - a_{33}) \\
p_{12} &= (a_{41} + b_{31}) \cdot (a_{43} + b_{13}) & p_{40} &= a_{42} \cdot (b_{22} - b_{21} - b_{23} - a_{41} - a_{43}) \\
p_{13} &= (a_{51} + b_{31}) \cdot (a_{53} + b_{13}) & p_{41} &= a_{52} \cdot (b_{22} - b_{21} - b_{23} - a_{51} - a_{53}) \\
p_{14} &= (a_{61} + b_{31}) \cdot (a_{63} + b_{13}) & p_{42} &= a_{62} \cdot (b_{22} - b_{21} - b_{23} - a_{61} - a_{63}) \\
p_{15} &= (a_{71} + b_{31}) \cdot (a_{73} + b_{13}) & p_{43} &= a_{72} \cdot (b_{22} - b_{21} - b_{23} - a_{71} - a_{73}) \\
p_{16} &= (a_{81} + b_{31}) \cdot (a_{83} + b_{13}) & p_{44} &= a_{82} \cdot (b_{22} - b_{21} - b_{23} - a_{81} - a_{83}) \\
p_{17} &= (a_{12} + b_{32}) \cdot (a_{13} + b_{23}) & p_{45} &= a_{13} \cdot (b_{33} - b_{31} - b_{32} - a_{11} - a_{12}) \\
p_{18} &= (a_{22} + b_{32}) \cdot (a_{23} + b_{23}) & p_{46} &= a_{23} \cdot (b_{33} - b_{31} - b_{32} - a_{21} - a_{22}) \\
p_{19} &= (a_{32} + b_{32}) \cdot (a_{33} + b_{23}) & p_{47} &= a_{33} \cdot (b_{33} - b_{31} - b_{32} - a_{31} - a_{32}) \\
p_{20} &= (a_{42} + b_{32}) \cdot (a_{43} + b_{23}) & p_{48} &= a_{43} \cdot (b_{33} - b_{31} - b_{32} - a_{41} - a_{42}) \\
p_{21} &= (a_{52} + b_{32}) \cdot (a_{53} + b_{23}) & p_{49} &= a_{53} \cdot (b_{33} - b_{31} - b_{32} - a_{51} - a_{52}) \\
p_{22} &= (a_{62} + b_{32}) \cdot (a_{63} + b_{23}) & p_{50} &= a_{63} \cdot (b_{33} - b_{31} - b_{32} - a_{61} - a_{62}) \\
p_{23} &= (a_{72} + b_{32}) \cdot (a_{73} + b_{23}) & p_{51} &= a_{73} \cdot (b_{33} - b_{31} - b_{32} - a_{71} - a_{72}) \\
p_{24} &= (a_{82} + b_{32}) \cdot (a_{83} + b_{23}) & p_{52} &= a_{83} \cdot (b_{33} - b_{31} - b_{32} - a_{81} - a_{82}) \\
p_{25} &= b_{12} \cdot b_{21} & p_{53} &= (a_{11} + b_{21} - b_{24}) \cdot (-a_{12} - b_{12} + b_{14}) \\
p_{26} &= b_{13} \cdot b_{31} & p_{54} &= (a_{21} + b_{21} - b_{24}) \cdot (-a_{22} - b_{12} + b_{14}) \\
p_{27} &= b_{23} \cdot b_{32} & p_{55} &= (a_{31} + b_{21} - b_{24}) \cdot (-a_{32} - b_{12} + b_{14}) \\
p_{28} &= (b_{21} - b_{24}) \cdot (-b_{12} + b_{14}) & p_{56} &= (a_{41} + b_{21} - b_{24}) \cdot (-a_{42} - b_{12} + b_{14})
\end{aligned}$$

$$p_{57} = (a_{51} + b_{21} - b_{24}) \cdot (-a_{52} - b_{12} + b_{14})$$
$$p_{58} = (a_{61} + b_{21} - b_{24}) \cdot (-a_{62} - b_{12} + b_{14})$$
$$p_{59} = (a_{71} + b_{21} - b_{24}) \cdot (-a_{72} - b_{12} + b_{14})$$
$$p_{60} = (a_{81} + b_{21} - b_{24}) \cdot (-a_{82} - b_{12} + b_{14})$$

$$p_{61} = a_{13} \cdot b_{34}$$
$$p_{62} = a_{23} \cdot b_{34}$$
$$p_{63} = a_{33} \cdot b_{34}$$
$$p_{64} = a_{43} \cdot b_{34}$$
$$p_{65} = a_{53} \cdot b_{34}$$
$$p_{66} = a_{63} \cdot b_{34}$$
$$p_{67} = a_{73} \cdot b_{34}$$
$$p_{68} = a_{83} \cdot b_{34}$$
$$p_{69} = a_{14} \cdot (b_{41} + a_{15})$$
$$p_{70} = a_{24} \cdot (b_{41} + a_{25})$$
$$p_{71} = a_{34} \cdot (b_{41} + a_{35})$$
$$p_{72} = a_{44} \cdot (b_{41} + a_{45})$$
$$p_{73} = a_{54} \cdot (b_{41} + a_{55})$$
$$p_{74} = a_{64} \cdot (b_{41} + a_{65})$$
$$p_{75} = a_{74} \cdot (b_{41} + a_{75})$$
$$p_{76} = a_{84} \cdot (b_{41} + a_{85})$$
$$p_{77} = a_{16} \cdot (b_{61} + a_{17})$$
$$p_{78} = a_{26} \cdot (b_{61} + a_{27})$$
$$p_{79} = a_{36} \cdot (b_{61} + a_{37})$$
$$p_{80} = a_{46} \cdot (b_{61} + a_{47})$$
$$p_{81} = a_{56} \cdot (b_{61} + a_{57})$$
$$p_{82} = a_{66} \cdot (b_{61} + a_{67})$$
$$p_{83} = a_{76} \cdot (b_{61} + a_{77})$$
$$p_{84} = a_{86} \cdot (b_{61} + a_{87})$$
$$p_{85} = b_{52} \cdot (b_{41} + b_{42})$$
$$p_{86} = b_{72} \cdot (b_{61} + b_{62})$$
$$p_{87} = b_{53} \cdot (b_{41} + b_{43})$$
$$p_{88} = b_{73} \cdot (b_{61} + b_{63})$$
$$p_{89} = b_{54} \cdot (b_{41} + b_{44})$$
$$p_{90} = b_{74} \cdot (b_{61} + b_{64})$$
$$p_{91} = a_{15} \cdot (b_{51} - a_{14})$$
$$p_{92} = a_{25} \cdot (b_{51} - a_{24})$$
$$p_{93} = a_{35} \cdot (b_{51} - a_{34})$$
$$p_{94} = a_{45} \cdot (b_{51} - a_{44})$$
$$p_{95} = a_{55} \cdot (b_{51} - a_{54})$$
$$p_{96} = a_{65} \cdot (b_{51} - a_{64})$$
$$p_{97} = a_{75} \cdot (b_{51} - a_{74})$$
$$p_{98} = a_{85} \cdot (b_{51} - a_{84})$$

$$p_{99} = a_{17} \cdot (b_{71} - a_{16})$$
$$p_{100} = a_{27} \cdot (b_{71} - a_{26})$$
$$p_{101} = a_{37} \cdot (b_{71} - a_{36})$$
$$p_{102} = a_{47} \cdot (b_{71} - a_{46})$$
$$p_{103} = a_{57} \cdot (b_{71} - a_{56})$$
$$p_{104} = a_{67} \cdot (b_{71} - a_{66})$$
$$p_{105} = a_{77} \cdot (b_{71} - a_{76})$$
$$p_{106} = a_{87} \cdot (b_{71} - a_{86})$$
$$p_{107} = (a_{14} + b_{52}) \cdot (a_{15} + b_{41} + b_{42})$$
$$p_{108} = (a_{24} + b_{52}) \cdot (a_{25} + b_{41} + b_{42})$$
$$p_{109} = (a_{34} + b_{52}) \cdot (a_{35} + b_{41} + b_{42})$$
$$p_{110} = (a_{44} + b_{52}) \cdot (a_{45} + b_{41} + b_{42})$$
$$p_{111} = (a_{54} + b_{52}) \cdot (a_{55} + b_{41} + b_{42})$$
$$p_{112} = (a_{64} + b_{52}) \cdot (a_{65} + b_{41} + b_{42})$$
$$p_{113} = (a_{74} + b_{52}) \cdot (a_{75} + b_{41} + b_{42})$$
$$p_{114} = (a_{84} + b_{52}) \cdot (a_{85} + b_{41} + b_{42})$$
$$p_{115} = (a_{16} + b_{72}) \cdot (a_{17} + b_{61} + b_{62})$$
$$p_{116} = (a_{26} + b_{72}) \cdot (a_{27} + b_{61} + b_{62})$$
$$p_{117} = (a_{36} + b_{72}) \cdot (a_{37} + b_{61} + b_{62})$$
$$p_{118} = (a_{46} + b_{72}) \cdot (a_{47} + b_{61} + b_{62})$$
$$p_{119} = (a_{56} + b_{72}) \cdot (a_{57} + b_{61} + b_{62})$$
$$p_{120} = (a_{66} + b_{72}) \cdot (a_{67} + b_{61} + b_{62})$$
$$p_{121} = (a_{76} + b_{72}) \cdot (a_{77} + b_{61} + b_{62})$$
$$p_{122} = (a_{86} + b_{72}) \cdot (a_{87} + b_{61} + b_{62})$$
$$p_{123} = (a_{14} + b_{53}) \cdot (a_{15} + b_{41} + b_{43})$$
$$p_{124} = (a_{24} + b_{53}) \cdot (a_{25} + b_{41} + b_{43})$$
$$p_{125} = (a_{34} + b_{53}) \cdot (a_{35} + b_{41} + b_{43})$$
$$p_{126} = (a_{44} + b_{53}) \cdot (a_{45} + b_{41} + b_{43})$$
$$p_{127} = (a_{54} + b_{53}) \cdot (a_{55} + b_{41} + b_{43})$$
$$p_{128} = (a_{64} + b_{53}) \cdot (a_{65} + b_{41} + b_{43})$$
$$p_{129} = (a_{74} + b_{53}) \cdot (a_{75} + b_{41} + b_{43})$$
$$p_{130} = (a_{84} + b_{53}) \cdot (a_{85} + b_{41} + b_{43})$$
$$p_{131} = (a_{16} + b_{73}) \cdot (a_{17} + b_{61} + b_{63})$$
$$p_{132} = (a_{26} + b_{73}) \cdot (a_{27} + b_{61} + b_{63})$$
$$p_{133} = (a_{36} + b_{73}) \cdot (a_{37} + b_{61} + b_{63})$$
$$p_{134} = (a_{46} + b_{73}) \cdot (a_{47} + b_{61} + b_{63})$$
$$p_{135} = (a_{56} + b_{73}) \cdot (a_{57} + b_{61} + b_{63})$$
$$p_{136} = (a_{66} + b_{73}) \cdot (a_{67} + b_{61} + b_{63})$$

$$p_{137} = (a_{76} + b_{73}) \cdot (a_{77} + b_{61} + b_{63}) \qquad p_{146} = (a_{84} + b_{54}) \cdot (a_{85} + b_{41} + b_{44})$$
$$p_{138} = (a_{86} + b_{73}) \cdot (a_{87} + b_{61} + b_{63}) \qquad p_{147} = (a_{16} + b_{74}) \cdot (a_{17} + b_{61} + b_{64})$$
$$p_{139} = (a_{14} + b_{54}) \cdot (a_{15} + b_{41} + b_{44}) \qquad p_{148} = (a_{26} + b_{74}) \cdot (a_{27} + b_{61} + b_{64})$$
$$p_{140} = (a_{24} + b_{54}) \cdot (a_{25} + b_{41} + b_{44}) \qquad p_{149} = (a_{36} + b_{74}) \cdot (a_{37} + b_{61} + b_{64})$$
$$p_{141} = (a_{34} + b_{54}) \cdot (a_{35} + b_{41} + b_{44}) \qquad p_{150} = (a_{46} + b_{74}) \cdot (a_{47} + b_{61} + b_{64})$$
$$p_{142} = (a_{44} + b_{54}) \cdot (a_{45} + b_{41} + b_{44}) \qquad p_{151} = (a_{56} + b_{74}) \cdot (a_{57} + b_{61} + b_{64})$$
$$p_{143} = (a_{54} + b_{54}) \cdot (a_{55} + b_{41} + b_{44}) \qquad p_{152} = (a_{66} + b_{74}) \cdot (a_{67} + b_{61} + b_{64})$$
$$p_{144} = (a_{64} + b_{54}) \cdot (a_{65} + b_{41} + b_{44}) \qquad p_{153} = (a_{76} + b_{74}) \cdot (a_{77} + b_{61} + b_{64})$$
$$p_{145} = (a_{74} + b_{54}) \cdot (a_{75} + b_{41} + b_{44}) \qquad p_{154} = (a_{86} + b_{74}) \cdot (a_{87} + b_{61} + b_{64})$$

After computing all of the linear combinations, one can obtain the element of the matrix C as follows:

$$c_{11} = p_1 + p_9 + p_{29} - p_{25} - p_{26} + p_{69} + p_{77} + p_{91} + p_{99}$$
$$c_{21} = p_2 + p_{10} + p_{30} - p_{25} - p_{26} + p_{70} + p_{78} + p_{92} + p_{100}$$
$$c_{31} = p_3 + p_{11} + p_{31} - p_{25} - p_{26} + p_{71} + p_{79} + p_{93} + p_{101}$$
$$c_{41} = p_4 + p_{12} + p_{32} - p_{25} - p_{26} + p_{72} + p_{80} + p_{94} + p_{102}$$
$$c_{51} = p_5 + p_{13} + p_{33} - p_{25} - p_{26} + p_{73} + p_{81} + p_{95} + p_{103}$$
$$c_{61} = p_6 + p_{14} + p_{34} - p_{25} - p_{26} + p_{74} + p_{82} + p_{96} + p_{104}$$
$$c_{71} = p_7 + p_{15} + p_{35} - p_{25} - p_{26} + p_{75} + p_{83} + p_{97} + p_{105}$$
$$c_{81} = p_8 + p_{16} + p_{36} - p_{25} - p_{26} + p_{76} + p_{84} + p_{98} + p_{106}$$

$$c_{12} = p_1 + p_{17} + p_{37} - p_{25} - p_{27} + p_{107} + p_{115} - p_{69} - p_{77} - p_{85} - p_{86}$$
$$c_{22} = p_2 + p_{18} + p_{38} - p_{25} - p_{27} + p_{108} + p_{116} - p_{70} - p_{78} - p_{85} - p_{86}$$
$$c_{32} = p_3 + p_{19} + p_{39} - p_{25} - p_{27} + p_{109} + p_{117} - p_{71} - p_{79} - p_{85} - p_{86}$$
$$c_{42} = p_4 + p_{20} + p_{40} - p_{25} - p_{27} + p_{110} + p_{118} - p_{72} - p_{80} - p_{85} - p_{86}$$
$$c_{52} = p_5 + p_{21} + p_{41} - p_{25} - p_{27} + p_{111} + p_{119} - p_{73} - p_{81} - p_{85} - p_{86}$$
$$c_{62} = p_6 + p_{22} + p_{42} - p_{25} - p_{27} + p_{112} + p_{120} - p_{74} - p_{82} - p_{85} - p_{86}$$
$$c_{72} = p_7 + p_{23} + p_{43} - p_{25} - p_{27} + p_{113} + p_{121} - p_{75} - p_{83} - p_{85} - p_{86}$$
$$c_{82} = p_8 + p_{24} + p_{44} - p_{25} - p_{27} + p_{114} + p_{122} - p_{76} - p_{84} - p_{85} - p_{86}$$

$$
\begin{aligned}
c_{13} &= p_9 + p_{17} + p_{45} - p_{26} - p_{27} + p_{123} + p_{131} - p_{69} - p_{77} - p_{87} - p_{88} \\
c_{23} &= p_{10} + p_{18} + p_{46} - p_{26} - p_{27} + p_{124} + p_{132} - p_{70} - p_{78} - p_{87} - p_{88} \\
c_{33} &= p_{11} + p_{19} + p_{47} - p_{26} - p_{27} + p_{125} + p_{133} - p_{71} - p_{79} - p_{87} - p_{88} \\
c_{43} &= p_{12} + p_{20} + p_{48} - p_{26} - p_{27} + p_{126} + p_{134} - p_{72} - p_{80} - p_{87} - p_{88} \\
c_{53} &= p_{13} + p_{21} + p_{49} - p_{26} - p_{27} + p_{127} + p_{135} - p_{73} - p_{81} - p_{87} - p_{88} \\
c_{63} &= p_{14} + p_{22} + p_{50} - p_{26} - p_{27} + p_{128} + p_{136} - p_{74} - p_{82} - p_{87} - p_{88} \\
c_{73} &= p_{15} + p_{23} + p_{51} - p_{26} - p_{27} + p_{129} + p_{137} - p_{75} - p_{83} - p_{87} - p_{88} \\
c_{83} &= p_{16} + p_{24} + p_{52} - p_{26} - p_{27} + p_{130} + p_{138} - p_{76} - p_{84} - p_{87} - p_{88}
\end{aligned}
$$

$$
\begin{aligned}
c_{14} &= p_1 + p_{53} + p_{61} - p_{25} - p_{28} + p_{139} + p_{147} - p_{69} - p_{77} - p_{89} - p_{90} \\
c_{24} &= p_2 + p_{54} + p_{62} - p_{25} - p_{28} + p_{140} + p_{148} - p_{70} - p_{78} - p_{89} - p_{90} \\
c_{34} &= p_3 + p_{55} + p_{63} - p_{25} - p_{28} + p_{141} + p_{149} - p_{71} - p_{79} - p_{89} - p_{90} \\
c_{44} &= p_4 + p_{56} + p_{64} - p_{25} - p_{28} + p_{142} + p_{150} - p_{72} - p_{80} - p_{89} - p_{90} \\
c_{54} &= p_5 + p_{57} + p_{65} - p_{25} - p_{28} + p_{143} + p_{151} - p_{73} - p_{81} - p_{89} - p_{90} \\
c_{64} &= p_6 + p_{58} + p_{66} - p_{25} - p_{28} + p_{144} + p_{152} - p_{74} - p_{82} - p_{89} - p_{90} \\
c_{74} &= p_7 + p_{59} + p_{67} - p_{25} - p_{28} + p_{145} + p_{153} - p_{75} - p_{83} - p_{89} - p_{90} \\
c_{84} &= p_8 + p_{60} + p_{68} - p_{25} - p_{28} + p_{146} + p_{154} - p_{76} - p_{84} - p_{89} - p_{90}
\end{aligned}
$$

D Efficient Multiplication Formula for Kyber512

Let $m \geq 2$ messages are needed to be encrypted with Kyber512. Then, $A \in R_q^{2\times 2}$, $B \in R_q^{2\times 2}$ and $A \cdot B = C \in R_q^{2\times 2}$. The elements of the matrices $A^{2\times 2} \cdot B^{2\times 2} = C^{2\times 2}$ are described as follows:

By Strassen (1969 Gaussian), the matrix 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 C as follows:

$$\begin{aligned}
c_{11} &= p_1 + p_4 - p_5 + p_7 & c_{21} &= p_2 + p_4 \\
c_{12} &= p_3 + p_5 & c_{22} &= p_1 - p_2 + p_3 + p_6
\end{aligned}$$

E Efficient Multiplication Formula for Kyber768

Let $m \geq 4$ messages are needed to be encrypted with Kyber768. Then, $A \in R_q^{3\times 3}$, $B \in R_q^{3\times 4}$ and $A \cdot B = C \in R_q^{3\times 4}$. The elements of the matrices $A^{3\times 3} \cdot B^{3\times 4} = C^{3\times 4}$ are described as follows:

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \\ c_{51} & c_{52} & c_{53} & c_{54} \end{bmatrix}}_{C}$$

By [21], the matrix C can be obtained as follows:

$$\begin{aligned}
p_1 &= (a_{11} + b_{21}) \cdot (a_{12} - b_{12}) \\
p_2 &= (a_{21} + b_{21}) \cdot (a_{22} - b_{12}) \\
p_3 &= (a_{31} + b_{21}) \cdot (a_{32} - b_{12}) \\
p_4 &= (a_{11} + b_{31}) \cdot (a_{13} - b_{13}) \\
p_5 &= (a_{21} + b_{31}) \cdot (a_{23} - b_{13}) \\
p_6 &= (a_{31} + b_{31}) \cdot (a_{33} - b_{13}) \\
p_7 &= b_{12} \cdot b_{21} \\
p_8 &= b_{13} \cdot b_{31} \\
p_9 &= b_{23} \cdot b_{32} \\
p_{10} &= (a_{12} + b_{32}) \cdot (a_{13} + b_{23}) \\
p_{11} &= (a_{22} + b_{32}) \cdot (a_{23} + b_{23}) \\
p_{12} &= (a_{32} + b_{32}) \cdot (a_{33} + b_{23}) \\
p_{13} &= (b_{21} - b_{24}) \cdot (-b_{12} + b_{14}) \\
p_{14} &= a_{11} \cdot (b_{11} - b_{12} - b_{13} - a_{12} - a_{13} \\
p_{15} &= a_{21} \cdot (b_{11} - b_{12} - b_{13} - a_{22} - a_{23} \\
p_{16} &= a_{31} \cdot (b_{11} - b_{12} - b_{13} - a_{32} - a_{33} \\
p_{17} &= a_{12} \cdot (b_{22} - b_{21} - b_{23} - a_{11} - a_{13} \\
p_{18} &= a_{22} \cdot (b_{22} - b_{21} - b_{23} - a_{21} - a_{23} \\
p_{19} &= a_{32} \cdot (b_{22} - b_{21} - b_{23} - a_{31} - a_{33} \\
p_{20} &= a_{13} \cdot (b_{33} - b_{31} - b_{32} - a_{11} - a_{12}
\end{aligned}$$

$$
\begin{aligned}
p_{21} &= a_{23} \cdot (b_{33} - b_{31} - b_{32} - a_{21} - a_{22} \\
p_{22} &= a_{33} \cdot (b_{33} - b_{31} - b_{32} - a_{31} - a_{32} \\
p_{23} &= (a_{11} + b_{21} - b_{24}) \cdot (-a_{12} - b_{12} + b_{14} \\
p_{24} &= (a_{21} + b_{21} - b_{24}) \cdot (-a_{22} - b_{12} + b_{14} \\
p_{25} &= (a_{31} + b_{21} - b_{24}) \cdot (-a_{32} - b_{12} + b_{14} \\
p_{26} &= a_{13} \cdot b_{34} \\
p_{27} &= a_{23} \cdot b_{34} \\
p_{28} &= a_{33} \cdot b_{34}
\end{aligned}
$$

After computing all of the linear combinations, one can obtain the element of the matrix C as follows:

$$
\begin{aligned}
c_{11} &= p_1 + p_4 + p_{14} - p_7 - p_8 \\
c_{12} &= p_1 + p_{10} + p_{17} - p_7 - p_9 \\
c_{13} &= p_4 + p_{10} + p_{20} - p_8 - p_9 \\
c_{14} &= p_1 + p_{23} + p_{26} - p_7 - p_{13}
\end{aligned}
$$

$$
\begin{aligned}
c_{21} &= p_2 + p_5 + p_{15} - p_7 - p_8 \\
c_{22} &= p_2 + p_{11} + p_{18} - p_7 - p_9 \\
c_{23} &= p_5 + p_{11} + p_{21} - p_8 - p_9 \\
c_{24} &= p_2 + p_{24} + p_{27} - p_7 - p_{13}
\end{aligned}
$$

$$
\begin{aligned}
c_{31} &= p_3 + p_6 + p_{16} - p_7 - p_8 \\
c_{32} &= p_3 + p_{12} + p_{19} - p_7 - p_9 \\
c_{33} &= p_6 + p_{12} + p_{22} - p_8 - p_9 \\
c_{34} &= p_3 + p_{25} + p_{28} - p_7 - p_{13}
\end{aligned}
$$