ANALYZES OF BLOCK RECOMBINATION AND LAZY INTERPOLATION METHODS AND THEIR APPLICATIONS TO SABER

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

BY

BERKİN AKSOY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CRYPTOGRAPHY

FEBRUARY 2022

Approval of the thesis:

ANALYZES OF BLOCK RECOMBINATION AND LAZY INTERPOLATION METHODS AND THEIR APPLICATIONS TO SABER

submitted by **BERKİN AKSOY** in partial fulfillment of the requirements for the degree of **Master of Science in Cryptography Department, Middle East Technical University** by,

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

Prof. Dr. Ferruh Özbudak
Head of Department, **Cryptography** _____

Assoc. Prof. Dr. Murat Cenk
Supervisor, **Cryptography, IAM, METU** _____

Examining Committee Members:

Prof. Dr. Ferruh Özbudak
Department of Mathematics, METU _____

Assoc. Prof. Dr. Murat Cenk
Cryptography, IAM, METU _____

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

Date: _____

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

Name, Last Name: BERKİN AKSOY

Signature :

ABSTRACT

ANALYZES OF BLOCK RECOMBINATION AND LAZY INTERPOLATION METHODS AND THEIR APPLICATIONS TO SABER

Aksoy, Berkin
M.S., Department of Cryptography
Supervisor : Assoc. Prof. Dr. Murat Cenk

February 2022, 61 pages

Since the beginning of the National Institute of Standards and Technology (NIST), The Post-Quantum Cryptography (PQC) Standardization Process, efficient implementations of lattice-based algorithms have been studied extensively. Lattice-based NIST PQC finalists use polynomial or matrix-vector multiplications on the ring with type $\mathbb{Z}_q[x]/f(x)$. For convenient ring types, Number Theoretic Transform (NTT) can be used to perform multiplications as done in Crystals-KYBER among the finalists of the NIST PQC Standardization Process. On the other hand, if the q value of the scheme is a power of 2, as in NTRU and Saber, which are among the other lattice-based finalists, NTT can not be used explicitly. Hence multiplications are performed by the combination of Toom-Cook and Karatsuba algorithms. Recently, a novel technique called lazy interpolation has been introduced to increase the performance of

Toom-Cook and Karatsuba algorithms. This thesis shows that the block recombination method is equivalent to lazy interpolation and can be used efficiently on multiplication algorithms. On the practical side, we compare different hybrid multiplication algorithms, then implement the block recombination method for Saber. Performance results are given in cycle values on general-purpose Intel processors with C implementation. Our work speeds up key generation, encapsulation, and decapsulation parts of Saber than the previous C implementations in the literature with a rate of between $10\% - 13\%$.

Keywords: Post-Quantum Cryptography, Lattice-Based Cryptography, Saber, Efficient Algorithm Implementations, Block Recombination, Lazy Interpolation

ÖZ

BLOCK RECOMBİNATİON VE LAZY INTERPOLATİON METODLARININ ANALİZİ VE SABER ÜZERİNDE UYGULAMALARI

Aksoy, Berkin
Yüksek Lisans, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Murat Cenk

Şubat 2022, 61 sayfa

Amerikan Ulusal Standartlar Enstitüsü (NIST) tarafından başlatılan kuantum sonrası kriptografi standardizasyon süreci ile birlikte akademik camia tarafından kafes tabanlı algoritmaların verimli gerçeklemeleri üzerine çalışmalar hız kazanarak devam etmektedir. NIST yarışmasında finale kalan kafes tabanlı algoritmalar, $\mathbb{Z}_q[x]/f(x)$ halka yapıları üzerinde matris-vektör ve polinom çarpımları gerçekleştirmektedirler. Literatürde yer alan NTT (Number Theoretical Transform) algoritması, uyumlu halka yapılarına sahip algoritmalarda polinom çarpım algoritması olarak kullanılabilmektedir. NIST PQC yarışmasında finale kalan adaylar arasında yer alan Crystals-KYBER'de NTT kullanılabilmekte iken NTRU ve Saber algoritmalarında kullanılan halka yapısında tanımlı q değerleri 2 ve üssü katlarında tanımlı olduğu için NTT

kullanımı mümkün olmamaktadır. Bu sebeple Toom-Cook ve Karatsuba algoritmaları ve kombinasyonları kullanılarak polinom çarpımları gerçekleştirilmektedir. Son günlerde lazy interpolation olarak adlandırılan yeni bir metod tanıtılmış ve bu metod sayesinde Toom-Cook ve Karatsuba algoritmalarının performansında iyileştirme sağlandığı gösterilmiştir. Bu tezde, block recombination metodunun çarpım algoritmalarına uygulanabileceğini ve lazy interpolation metoduna denk olduğunu gösteriyoruz. Gerçekleme tarafında ise genel amaçlı işlemcilerde C dilinde yaptığımız çalışma ile Saber'da hibrit çarpım algoritmalarını karşılaştırıyoruz ve block recombination metodunu uyguluyoruz. Bu sonuçlar doğrultusunda, Saber'da block recombination metodunu içeren çalışmamız ile literatürde yer alan C dilinde yapılan diğer gerçekleme sonuçlarına göre anahtar üretimi, anahtar kapsülleme ve dekapsülleme aşamalarında $\%10 - \%13$ arasında performans artışı gözlemliyoruz.

Anahtar Kelimeler: Kuantum Sonrası Kriptografi, Kafes Tabanlı Kriptografi, Saber, Verimli Algoritma Gerçeklemeleri, Block Recombination, Lazy Interpolation

to my family

ACKNOWLEDGMENTS

I would like to express my great appreciation to my thesis supervisor Assoc. Prof. Dr. Murat Cenk for his patient guidance, enthusiastic encouragement, and valuable advice during the development and preparation of this thesis.

I would like to thank Aselsan Inc. for supporting the academic studies of its employees.

My deepest thanks belong to my wife Elif Sağlam Aksoy, for encouraging me to a bright future with his support and love.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

National Institute of Standards and Technology (NIST) organized a post-quantum cryptography standardization process in 2017 and announced Round 3 finalists [1] in July 2020. Three of four PKE/KEM schemes, Saber[13], NTRU[8] and Kyber[3], are based on lattice-based cryptography, and NIST also stated that one of these algorithms would be standard. In these schemes, arithmetic operations are defined on the rings of the form $R_q = \mathbb{Z}_q[x]/f(x)$ where $f(x) \in \mathbb{Z}$. Multiplications and additions of polynomials on these rings affect the performance and efficiency of these schemes with different n and q values. Kyber[3] supports NTT (number theoretic transform) [10] which has complexity $O(n \log n)$. However, constraints on the n and q parameters limit the security levels of the schemes which can use NTT. To compute $A(x)B(x) = C(x)$ in R_q by using NTT, the forward and inverse transforms are used and $NTT^{-1}(NTT(\overline{A}) \times NTT(\overline{B})$ where $\overline{A}$ and $\overline{B}$ consist of the coefficient of $A(x), B(x)$, is computed. First, coefficients of $A(x)$ and $B(x)$ are transformed to the NTT domain by the forward NTT function. Second, multiplications are done and, the result is found in the NTT domain. Therefore, the result of multiplications has to be reversed from the NTT domain by reversing the NTT function in the final stage.

Due to constraints of the ring structure ($q = 2^k$), Saber and NTRU schemes cannot use NTT multiplication. Therefore, Toom Cook [27], [9] and Karatsuba [19] algorithms and combinations are used to perform polynomial multiplications. Toom-Cook and Karatsuba based multiplication algorithms need pre-and post-processing operations to perform polynomial multiplication with smaller degrees than the original multiplicands. The pre-and post-processing process of the algorithms is improved using different mathematical approaches. In [7], the block recombination method has been introduced and, its advantages are given to decrease the time complexity with optimized post-processing operations of Toom-Cook and Karatsuba algorithms. Recently in [23], the lazy interpolation method has also been presented for the Toom-Cook and Karatsuba algorithms, and techniques similar to those used in the block recombination method are used to decrease the time complexity of multiplication algorithms.

In this thesis, we give the block recombination and lazy interpolation methods' analysis in terms of their computational cost and the techniques they use to show the equivalence of the two methods. On practical aspects, we work on the efficient implementation of matrix-vector and vector-dot products on Saber. Selecting the efficient hybrid multiplication strategy, a combination of Toom Cook and Karatsuba algorithms depends on the implementation platform or the ring types in which the polynomials are multiplied. By examining different hybrid multiplication algorithms, we show the best combination to perform matrix-vector and vector-dot products in Saber. Then, we apply the block recombination application to Saber and get better results than the lazy interpolation application given in [23].

On general-purpose Intel processors, we compare the performance results in terms of clock cycles with previous C implementations [13, 14, 5, 23, 12] and our work have

the best results among the others with improvement rates up to 13% on key generation, encapsulation, and decapsulation stages of Saber. All source codes available at `https://github.com/berkinaksoy/saber`

1.1 Outline of the Thesis

In Chapter 2, post-quantum cryptography, lattice-basics, Saber crypto-scheme, and polynomial multiplication algorithms are presented. Analysis of block recombination and lazy interpolation methods are given and compared in Chapter 3. In Chapter 4, we apply the block recombination method to Saber and compare our work with lazy interpolation application in [23]. In Chapter 5, we implement different hybrid multiplication algorithms, block recombination methods to Saber, and compare the performance results in C implementation. In Chapter 6, the results of our work are summarized, and we mention the ideas and strategies that are open to developing in the future.

CHAPTER 2

PRELIMINARY TO THE SUBJECT

Some essential definitions and preliminaries will be introduced in this chapter to offer the reader enough theoretical knowledge of our work and analysis.

This chapter is divided into four sections. The first section will give brief information about Post-Quantum Cryptography. Secondly, lattice-basics are the background of PQC algorithms that will be summarized. Thirdly, Saber crypto-scheme, one of the candidates of the NIST standardization process in the final round, will be explained by giving the main primitives and notations. Due to its crucial role in our analyzes, Toom-Cook and Karatsuba algorithms and their applications will be given in the last section.

2.1 Post-Quantum Cryptography

With the development of quantum computers, RSA (Rivest–Shamir–Adleman), DSA (Digital Signature Algorithm), ECDSA (Elliptic Curve Digital Signature Algorithm), and ECDH (Elliptic Curve Diffie-Hellman) are currently used public-key algorithms based on integer factorization and discrete logarithm problems. They will be broken

and solved in polynomial time by Shor algorithm [26]. Micheal Mosca stated [24] that "By 2026, I believe there is a 1/7 possibility of breaking RSA-2048 and a 1/2 chance by 2031. Many firms, including Google and IBM, work on the quantum computers development process. Hence, intensive academic research has been done on developing new public key crypto schemes resistant to both classical and quantum attacks. There are five types of algorithms with security against classical and quantum computers. These algorithms consist of lattice-based, hash-based, code-based, isogeny-based, and multivariate polynomials. It is called post-quantum cryptography.

NIST organized the standardization process in 2017. In the first round of the process, 82 algorithms were submitted, only 69 were accepted and evaluated. In 2019, 26 algorithms were selected as second-round candidates, and finally, seven were introduced as final-round candidates in 2020. NIST stated that selection criteria consist of security, sizes, and performance. Therefore, studies on efficient algorithm implementations that are both performance-oriented and compact size will continue in the future. Moreover, NIST also highlights that multiple algorithms based on different problems will be standard instead of a single algorithm that will be standard at the end of the competition. Thereby, various working groups will continue to work on the development of algorithms based on different problems.

2.2 Lattice-Basics

Let $\{b_1, b_2, \ldots, b_n\}$ is linearly independent n-dimensional vector over ring R^n. The lattice L can be defined, $m_1b_1 + \ldots + m_nb_n$ where $m_i \in \mathbb{Z}$ as a set of vectors. L can be generated from set of vectors ($\{b_1, b_2, \ldots, b_n\}$), which called basis of lattice. There

are infinitely many bases, that can generate the lattice L. Length of the bases is in the form $||x|| = (x_1^2 + \ldots + x_n^2)^{1/2}$, where set of vectors defined as $x = (x_1, \ldots, x_n)$. Finding shortest non-zero vector $||x||$ is hard to solve and called shortest vector problem in lattice-based cryptography.

The LLL algorithm by Lenstra, Lenstra, and Lovash [25] is the most efficient algorithm to find reduced bases in the lattice. It runs in with complexity $O(n^6 \log^3 B)$. When the n dimensional vector space increases, the founded short vector deviates from the shortest vector of the lattice. Let $\{b_1, b_2, \ldots, b_n\}$ is linearly independent n-dimensional vector over ring R^n, then determinant of the lattice calculated as $D = |det(b_1, \ldots, b_n)|$. λ be the shortest nonzero vector in lattice L. The output of the LLL algorithm gives a basis that meets the following three conditions:

1. $||b_1^*|| \leq 2^{(n-1)/4} D^{1/4}$,
2. $||b_1^*|| \leq 2^{(n-1)/2} \lambda$,
3. $||b_1^*||||b_2^*|| \cdots ||b_n^*|| \leq 2^{(n-1)/4} D$.

Lattice-based PQC algorithms are based on problems, Learning with Error Problem (LWE), Ring Learning with Error (RLWE), Module Learning with Error (MLWE), and Module Learning with Rounding (MLWR) are some of them and as explained follows:

It is easy to solve the system with linear equations. If the noise (an error) is added to the system, the problem becomes too hard to solve. LWE [2] is based on adding noise to the system with the specified error distribution. For modulus $q \geq 2$, $n \geq 1$ and error probability distribution χ on Z_q, samples of the error distribution in normal-form are calculated as $(\mathbf{a}, \mathbf{b} = \frac{1}{q}\langle \mathbf{a}, \mathbf{s} \rangle + e \bmod 1)$. Vector $a \in Z_q^n$ is selected uniformly

at random. Secret vector, $s \in Z_q^n$, and error e are chose from the distribution χ. Distinguishing error distribution from a uniform distribution is named the decision LWE problem. Given random samples from LWE distribution, finding the secret vector s with high probability is hard to solve the search LWE problem.

RLWE is defined as a special case of LWE problem in [22]. Coefficients of polynomials are defined in $\mathbb{R}_q[x]$ instead of $\mathbb{Z}_q[x]$. Error, e, is a small polynomial and sampled from the distribution χ. A publicly known polynomial, a, is selected randomly in $\mathbb{R}_q[x]$. Secret, s, is a small polynomial and sampled from the distribution χ. Polynomial b is calculated as $a.s + e$, hence, finding secret polynomial s is hard to solve search problem.

LWE-based cryptosystems are easily scalable but have less performance, while RLWE based cryptosystems have advantageous on speed and size criteria but are more vulnerable to attacks. Therefore, MLWE problem [29, 20], is introduced to solve the trade-off between LWE and RLWE offers an interpolation between LWE and RLWE. Ring elements a and s in RLWE are replaced with module elements over the same ring. MLWE problem is a special case of RLWE with rank 1. Publicly known matrix, a, and secret vector, s are sampled in $\mathbb{R}_q^d$. Given d is the rank of the module. As in the LWE-RLWE problem, polynomial b is constructed as $a.s + e$, and finding secret polynomial s is hard to solve the search problem.

Learning with Rounding (LWR) problem is similar to LWE. However, this time deterministic error/noise is added by reducing with modulus q and p. MLWR problem and MLWE are module versions of LWE/LWR, offering performance improvements.

2.3 Saber Scheme

Saber [13, 14, 5] is a cryptographic primitive family consisting of the SaberPKE and SaberKEM that are IND-CPA secure encryption scheme and IND-CCA secure key encapsulation mechanism, respectively. Saber is one of the NIST PQC standardization process' lattice-based finalists, and its security is based on the hardness of the Module Learning with Rounding (Mod-LWR) problem [21, 4]. SaberPKE consist of key generation, encryption and decryption schemes that are given in Algorithm 1, 2, 3 respectively. By using Fujisaki-Okamoto transform [15], [17] public-key encryption scheme is transformed to Key Encapsulation, Encapsulation and Decapsulation stages which are given for SaberKEM in Algorithm 4, 5, 6, respectively. SaberKEM algorithm has three different parameter sets as shown in Table 2.1. Rings defined on Saber are not suitable for using NTT. Hence, Toom-Cook, Karatsuba, and schoolbook methods and their combinations are used as polynomial multiplication algorithms.

2.3.1 Saber Notation

In Table 2.1 and Algorithms 1, 2, 3, 4, 5, 6, given parameters are explained as follows:

- R_q is quotient ring and equal to $\mathbb{Z}_q[x]/f(x)$ where $f(x) = x^n + 1$ with $n = 256$.
- R^{lxk} represents the ring of lxk matrices over ring R. l value determines the dimension of the matrices and security levels of the algorithm. Increasing l values increase the security level of the algorithm.
- The coefficient of the secret vector s sampled by centered binomial distribution $\beta_\mu(R_q^{l\times 1})$ where parameter $\mu < p$ and samples are in the interval $[-\mu/2, \mu/2]$.

- p,q,T are powers of 2. Choosing high p and T values increase correctness while decrease security level. $q = 2^{\epsilon_q}$, $p = 2^{\epsilon_p}$, $T = 2^{\epsilon_T}$ with $\epsilon_q > \epsilon_p > \epsilon_T$ and hence $T|p|q$.

- $\mathcal{U}$ denotes the uniform distribution. SHAKE-128 is used to generate $seed$ value. $\mathcal{F}$, $\mathcal{H}$ are implemented by SHA-256 and $\mathcal{G}$ is implemented by SHA-512.

- $h \in R_q^{lx1}$ is a constant vector, $h_1, h_2 \in R_q$ are constant polynomials.

Table 2.1: SaberKEM Parameter Sets

Parameter Set	LIGHTSABER	SABER	FIRESABER
Size of n, q, p	256, 2^{13}, 2^{10}	256, 2^{13}, 2^{10}	256, 2^{13}, 2^{10}
l, T, μ	2, 2^3, 10	3, 2^4, 8	4, 2^6, 6
R_q	$\mathbb{Z}_{8192}[x]/(X^{256}+1)$	$\mathbb{Z}_{8192}[x]/(X^{256}+1)$	$\mathbb{Z}_{8192}[x]/(X^{256}+1)$
Fail probability	2^{-120}	2^{-136}	2^{-165}
Quantum bit security	114	185	257
NIST security level	1	3	5

Algorithm 1 SaberPKE.KeyGen

$seed_{\mathbf{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$

$A = gen(seed_{\mathbf{A}}) \in R_q^{l \times l}$

$r = \mathcal{U}(\{0,1\}^{256})$

$\mathbf{s} = \beta_\mu(R_q^{l \times 1}; r)$

$\mathbf{b} = ((\mathbf{A}^T\mathbf{s} + \mathbf{h})\ mod\ q) \gg (\epsilon_q - \epsilon_p)$

return $(pk := (seed_{\mathbf{A}}, \mathbf{b}), sk := (\mathbf{s}))$

Algorithm 2 SaberPKE.Encryption

$A = gen(seed_{\mathbf{A}}) \in R_q^{l \times l}$

if r is not specified **then**

$r = \mathcal{U}(\{0,1\}^{256})$

$\mathbf{s}' = \beta_\mu(R_q^{l \times 1}; r)$

$\mathbf{b}' = ((\mathbf{A}\mathbf{s}' + \mathbf{h})\ mod\ q) \gg (\epsilon_q - \epsilon_p) \in R_p^{l \times 1}$

$v' = \mathbf{b}^T(\mathbf{s}'\ mod\ p) \in R_p$

$c_m = (v' + h_1 - 2^{\epsilon_p - 1} m\ mod\ p) \gg (\epsilon_p - \epsilon_T) \in R_T$

return $c := (c_m, \mathbf{b}')$

Algorithm 3 SaberPKE.Decryption

$v = \mathbf{b}'^T(\mathbf{s}\ mod\ p) \in R_p$

$m' = ((v - 2^{\epsilon_p - \epsilon_T} c_m + h_2)\ mod\ p) \gg (\epsilon_p - 1) \in R_2$

return m'

Algorithm 4 SaberKEM KeyGeneration

$(seed_{\mathbf{A}}, \mathbf{b}, \mathbf{s}) = SaberPKE.KeyGen()$

$pk = (seed_{\mathbf{A}}, \mathbf{b})$

$pkh = \mathcal{F}(pk)$

$z = \mathcal{U}(\{0,1\}^{256})$

return $(pk := (seed_{\mathbf{A}}, \mathbf{b}),\ sk := (\mathbf{s}, z, pkh))$

Algorithm 5 SaberKEM Encapsulation

$m \leftarrow \mathcal{U}(\{0,1\}^{256})$

$(\hat{K}, r) = \mathcal{G}(\mathcal{F}(pk), m)$

$c = SaberPKE_Encryption(pk, m; r)$

$K = \mathcal{H}(\hat{K}, c)$

return (c, K)

Algorithm 6 SaberKEM Decapsulation $(sk = (\mathbf{s}, z, pkh),\ pk = (seed_{\mathbf{A}}, \mathbf{b}), c)$

$m' = SaberPKE.Decryption(\mathbf{s}, c)$

$(\hat{K}', r') = \mathcal{G}(pkh, m')$

$c' = SaberPKE.Encryption(pk, m'; r')$

if $c = c'$ **then**

return $K = \mathcal{H}(\hat{K}', c)$

else

return $K = \mathcal{H}(z, c)$

2.4 Multiplication Algorithms

Fundamentals of the polynomial multiplication algorithms are given in this section. Firstly, the naive method as schoolbook is presented, then Toom-Cook and Karatsuba algorithms and their use-cases are explained with their advantages.

2.4.1 Schoolbook Method

Schoolbook method is the naive algorithm to multiply polynomials with $A(x)$ and $B(x)$ with the degree of $n-1$. Firstly, polynomial $A(x)$ are splitted into 2 parts, $A(x) = A_0 + A_1 y$, where $A_0 = a_0 + a_1 + ... + a_{\frac{n}{2}-1}x^{\frac{n}{2}-1}$, $A_1 = a_{\frac{n}{2}} + a_{\frac{n}{2}+1}x + ... + a_{n-1}x^{\frac{n}{2}-1}$ and $y = x^{\frac{n}{2}}$. By the same way, B(x) are splitted into 2 parts. Then the multiplication of $A(x)$ and $B(x)$ are calculated as:

$$A(x)B(x) = A_0B_0 + y[(A_1B_0) + (A_0 + B_1)] + y^2(A_1B_1) \tag{2.1}$$

Result of polynomial multiplication $C(x)$ can be found by 4 polynomial multiplications with degree of $(n/2)$ and some additions. If $M(n)$ is the complexity of $A(x)B(x)$ multiplication then $M(n) = 4M(n/2) + c.n$. By master theorem [11],

complexity of school-book method can be found $O(n^{\log_2 4}) = O(n^2)$.

2.4.2 Karatsuba 2-way Algorithm

Karatsuba Multiplication [19] use divide and conquer strategy that improve quadratic complexity of the classic schoolbook multiplication. To multiply polynomials with $A(x)$ and $B(x)$ with the degree of $n-1$, firstly, polynomial $A(x)$ are splitted into 2 parts, $A(x) = A_0 + A_1 y$, where $A_0 = a_0 + a_1 + ... + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$, $A_1 = a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + ... + a_{n-1} x^{\frac{n}{2}-1}$ and $y = x^{\frac{n}{2}}$. By the same way, B(x) are splitted into 2 parts.Then the multiplication of $A(x)$ and $B(x)$ are calculated as:

$$A(x)B(x) = A_0B_0 + y[(A_0 + A_1)(B_0 + B_1) - A_0B_0 - A_1B_1] + A_1B_1y^{\frac{n}{2}} \quad (2.2)$$

Using Karatsuba 2-way method, result of polynomial multiplication $C(x)$ can be found by 3 polynomial multiplications with degree of $(n/2)$ and some additions. If $M(n)$ is the complexity of $A(x)B(x)$ multiplication then $M(n) = 3M(n/2) + c.n$. By master theorem [11], complexity of Karatsuba 2-way method can be found $O(n^{\log_2 3}) = O(n^{1.58})$. The total complexity of the Karatsuba 2-way algorithm can be calculated from block complexities as follows:

Component Polynomial Formation (CPF):

$$\begin{aligned} CPF(A) &= [(A_0), (A_1), (A_0 + A_1)] \\ CPF(B) &= [(B_0), (B_1), (B_0 + B_1)] \end{aligned} \quad (2.3)$$

Totally $6CPF(n/4)$ block required where $CPF(n) = 3CPF(n/2) + n/2$

Component multiplication (CM):

$$\begin{aligned} CM(A,B) &= CPF(A) \times CPF(B) \\ CPF(A) * CPF(B) &= \underbrace{(A_0B_0)}_{P_0}, \underbrace{(A_1B_1)}_{P_1}, \underbrace{(A_0+A_1)(B_0+B_1)}_{P_2} \end{aligned} \tag{2.4}$$

After CM part, we have three recursive products of polynomials with size $n/2$, i.e $3CM(n/2)$ required.

Then Karatsuba 2-way reconstruction algorithm applied to get result C in the following expressions in terms of $P_i, i = 0, 1, 2$

$C = (P_0 + (P_2 - P_1 - P_0)x^{n/2} + P_1x^n)$

To reconstruct resulting polynomial, we require $3R(n/2)$ blocks.

Finally, complexity of the Karatsuba 2-way formula can be evaluated as follows:

= $6CPF(n/2) + 3CM(n/2) + 3R(n/2)$ where complexity of CPF, CM and R blocks converted in Equation 2.5.

$$\begin{aligned} CPF(n/2) &= \frac{1}{3}CPF(n) \\ CM(n/2) &= \frac{1}{3}CM(n) \\ R(n/2) &= \frac{1}{3}R(n) \end{aligned} \tag{2.5}$$

Complexity of CPF, CM, R blocks are shown in Table 2.2 and complexity of optimized reconstruction part ($R(n) = \frac{7}{2}n^{\log_2 3} - 5n + \frac{3}{2}$) as calculated in [7]. Therefore, total complexity of the Karatsuba 2-way formula calculated as follows:

$$\begin{aligned} &= 6CPF(n/2) + 3CM(n/2) + 3R(n/2) \\ &= 2CPF(n) + CM(n) + R(n) \\ &= 2(n^{\log_2 3} - n) + n^{\log_2 3} + \frac{7}{2}n^{\log_2 3} - 7n + \frac{3}{2} \\ &= 6,5n^{\log_2 3} - 7n + 1,5 \end{aligned} \tag{2.6}$$

Table 2.2: Block complexities of Karatsuba 2-way formula

Blocks	Complexity
$CPF(n)$	$n^{\log_2 3} - n$
$CM(n)$	$n^{\log_2 3}$
$R(n)$	$\frac{7}{2}n^{\log_2 3} - 7n + \frac{3}{2}$

2.4.3 Karatsuba 3-way Algorithm

Let A and B are polynomials with degree of n-1. A and B polynomials are divided into 3 such as $A = A_0 + A_1x^{n/3} + A_2x^{2n/3}$ and $B = B_0 + B_1x^{n/3} + B_2x^{2n/3}$. Multiplication of A(x)B(x) require 6 recursive products of size $n/3$ and then result polynomial is found as follows:

$$\begin{aligned} &P_0 = A_0B_0, P_1 = A_1B_1, \\ &P_2 = (A_1 + A_2)(B_1 + B_2), P_3 = A_2B_2, \\ &P_4 = (A_0 + A_1)(B_0 + B_1), P_5 = (A_0 + A_2)(B_0 + B_2) \end{aligned} \tag{2.7}$$

Result of polynomial multiplication $C(x)$ can be found by 6 $(n/3)$ degree polynomial multiplication and some additions. If $M(n)$ is the complexity of $A(x)B(x)$ multiplication then $M(n) = 6M(n/3) + c.n$. By master theorem [11], complexity of Karatsuba 3-way method can be found $O(n^{\log_3 6}) = O(n^{1.63})$. Total complexity of Karatsuba 3-way algorithm can be calculated from block complexities as follows:

Component Polynomial Formation (CPF):

$$\begin{aligned} CPF(A) &= [(A_0), (A_1), (A_1 + A_2), (A_2), (A_0 + A_1), (A_0 + A_2)] \\ CPF(B) &= [(B_0), (B_1), (B_1 + B_2), (B_2), (B_0 + B_1), (B_0 + B_2)] \end{aligned} \tag{2.8}$$

Totally $12CPF(n/4)$ block required where $CPF(n) = 6CPF(n/4) + n/4$

Component multiplication (CM):

$$
\begin{aligned}
CM(A,B) &= CPF(A) \times CPF(B) \\
CPF(A) * CPF(B) &= \underbrace{(A_0B_0)}_{P_0}, \underbrace{(A_1B_1)}_{P_1}, \underbrace{(A_1+A_2)(B_1+B_2)}_{P_2}, \underbrace{(A_2B_2)}_{P_3}, \\
&\quad \underbrace{(A_0+A_1)(B_0+B_1)}_{P_4}, \underbrace{(A_0+A_2)(B_0+B_2)}_{P_5}
\end{aligned} \tag{2.9}
$$

After CM part, we have six recursive products of polynomials with size $n/3$, i.e $6CM(n/3)$ required.

Then Karatsuba 3-way reconstruction algorithm applied to get result C in the following expressions in terms of $P_i, i = 0, 1, 2, ..., 5$

$C = P_0 + x^{n/3}(P_4 - P_0 - P_1) + x^{2n/3}(P_5 - P_0 - P_1 - P_3) + x^{3n/3}(P_2 - P_1 - P_3) + x^{4n/3}(P_3)$

To reconstruct resulting polynomial, we require $6R(n/3)$ blocks.

Finally, complexity of the Karatsuba 3-way formula can be evaluated as follows:

= $12CPF(n/3)+6CM(n/3)+6R(n/3)$ where complexity of CPF, CM and R blocks are converted in Equation 2.3.

$$
\begin{aligned}
CPF(n/3) &= \frac{1}{6}CPF(n) \\
CM(n/3) &= \frac{1}{6}CM(n) \\
R(n/3) &= \frac{1}{6}R(n)
\end{aligned} \tag{2.10}
$$

Complexity of CPF, CM, R blocks are shown in Table 2.3 and complexity of optimized reconstruction part ($R(n) = \frac{14}{5}n^{\log_3 6} - 4n + \frac{6}{5}$) as given in [7]. Therefore,

total complexity of the Karatsuba 3-way formula calculated as follows:

$$\begin{aligned} &= 12CPF(n/3) + 6CM(n/3) + 6R(n/3) \\ &= 2CPF(n) + CM(n) + R(n) \\ &= 2(n^{\log_3 6} - n) + n^{\log_3 6} + \frac{14}{5}n^{\log_3 6} - 4n + \frac{6}{5} \\ &= 5,8n^{\log_3 6} - 6n + 1,2 \end{aligned} \tag{2.11}$$

Table 2.3: Block complexities of Karatsuba 3-way formula

Blocks	Complexity
$CPF(n)$	$n^{\log_3 6} - n$
$CM(n)$	$n^{\log_3 6}$
$R(n)$	$\frac{14}{5}n^{\log_3 6} - 4n + \frac{6}{5}$

2.4.4 Bernstein 4-way Split Formula

The strategy of the Bernstein four-way split formula is two recursive applications of the Karatsuba algorithm. Bernstein [6] optimize the reconstruction part of the four-way split formula. Let A and B be two polynomials with the degree of $n-1$, where $n = 2^k$ for $k \geq 2$. A and B polynomials are divided into four $A = A_0 + A_1x^{n/4} + A_2x^{2n/4} + A_3x^{3n/4}$ and $B = B_0 + B_1x^{n/4} + B_2x^{2n/4} + B_3x^{3n/4}$. To multiply polynomials $A(x)$ and $B(x)$ with three separate computations in terms of the CPF, CM, and R as given below:

Component Polynomial Formation (CPF): CPF function is used twice recursively. Let $A = A_v + A_wx^{n/2}$, $B = B_v + B_wx^{n/2}$ then $A_v = A_0 + A_1x^{n/4}$, $A_w = A_2 + A_3x^{n/4}$

$$CPF(A) = (A_v, A_v + A_w, A_w)$$

$$CPF(B) = (B_v, B_v + B_w, B_w)$$

$$CPF(CPF(A)) = CPF(A_v), CPF(A_v + A_w), CPF(A_w)$$

$$CPF(CPF(B)) = CPF(B_v), CPF(B_v + B_w), CPF(B_w)$$

$$CPF(A_v) = (A_0, A_0 + A_1, A_1)$$

$$CPF(B_v) = (B_0, B_0 + B_1, B_1)$$

$CPF(A_v + A_w), CPF(A_w), CPF(B_v + B_w), CPF(B_w)$ calculated in a similar way

(2.12)

Totally $18CPF(n/4)$ blocks required where $CPF(n) = 3CPF(n/2) + n/2$

Component multiplication (CM):

$$\begin{aligned}
CM(A, B) =& CPF(A_v) \times CPF(B_v), \\
& CPF(A_v + A_w) \times CPF(B_v + B_w), \\
& CPF(A_w) \times CPF(B_w)
\end{aligned}$$

$$CPF(A_v) \times CPF(B_v) = \underbrace{(A_0 B_0)}_{P_0}, \underbrace{(A_0 + A_1)(B_0 + B_1)}_{P_2}, \underbrace{(A_1 B_1)}_{P_1}$$

$$CPF(A_w) \times CPF(B_w) = \underbrace{(A_2 B_2)}_{P_3}, \underbrace{(A_2 + A_3)(B_2 + B_3)}_{P_5}, \underbrace{(A_3 B_3)}_{P_4}$$

$$\begin{aligned}
CPF(A_v + A_w) \times CPF(B_v + B_w) =& \underbrace{(A_0 + A_2)(B_0 + B_2)}_{P_6}, \\
& \underbrace{(A_0 + A_1 + A_2 + A_3)(B_0 + B_1 + B_2 + B_3)}_{P_8}, \\
& \underbrace{(A_1 + A_3)(B_1 + B_3)}_{P_7}
\end{aligned}$$

(2.13)

Reconstruction (R): After CM part, we have nine polynomial products with degree of $n/4$, i.e, $9CM(n/4)$ blocks are required. Then two layer of the Karatsuba reconstruction algorithm applied recursively to get result C in the following expressions in terms of P_i, where $i = 0, 1, 2, ..., 8$

$$\begin{aligned} C =& P_0 + x^{n/4}(P_2 - P_0 - P_1) + x^{2n/4}(P_6 + P_1 - P_0 - P_3) \\ &+ x^{3n/4}(P_8 + P_0 + P_3 + P_1 + P_4 - P_2 - P_5 - P_6 - P_7) \\ &+ x^{4n/4}(P_7 - P_1 - P_4 + P_3) + x^{5n/4}(P_5 - P_3 - P_4) + x^{6n/4}P_4 \end{aligned} \tag{2.14}$$

To reconstruct polynomial result, we need $9R(n/4)$ blocks. Finally, Bernstein four-way split formula's complexity can be evaluated as $18CPF(n/4) + 9CM(n/4) + 9R(n/4)$ in Equation 2.15 where complexity of CPF, CM and R blocks are presented in Table 2.4. Optimized block complexity is derived in [6].

$$\begin{aligned} &= 18CPF(n/4) + 9CM(n/4) + 9R(n/4) \\ &= 2CPF(n) + CM(n) + R(n) \\ &= 2(n^{\log_2 3} - n) + n^{\log_2 3} + \frac{137}{40}n^{\log_2 3} - \frac{24n}{5} + \frac{11}{8} \\ &= 6,43n^{\log_2 3} - 6,8n + 1,38 \end{aligned} \tag{2.15}$$

where $CPF(n/4) = \frac{1}{9}CPF(n), CM(n/4) = \frac{1}{9}CM(n), R(n/4) = \frac{1}{9}R(n)$

Table 2.4: Block complexities of Bernstein four-way formula

Blocks	Complexity
$CPF(n)$	$n^{\log_2 3} - n$
$CM(n)$	$n^{\log_2 3}$
$R(n)$	$\frac{137}{40}n^{\log_2 3} - \frac{24n}{5} + \frac{11}{8}$

2.4.5 Toom-Cook Algorithm

Toom-Cook multiplication [27] and [9] is a generalization technique of Karatsuba method, i.e, Toom-Cook k-way method which divide polynomial into k equal parts instead of 2 in Karatsuba. Then splitted polynomials are multiplied and result is constructed from this polynomial multiplications. If $M(n)$ is the complexity of $A(x)B(x)$ multiplication then $M(n) = (2k-1)M(n/k) + c.n$. For example, Toom-Cook 4-way ($k = 4$) complexity, $M(n) = 7M(n/4) + c.n$, can be can be calculated by Master Theorem as $O(n^{1.40})$.

2.4.6 Toom-Cook 4-way Method

Let A_v and B_v are two polynomials degree with $n-1$, where $n = 2^k$ for $k \geq 2$. A_v and B_v polynomials are divided into four $A_v = A_{v0} + A_{v1}x^{n/4} + A_{v2}x^{2n/4} + A_{v3}x^{3n/4}$ and $B_v = B_{v0} + B_{v1}x^{n/4} + B_{v2}x^{2n/4} + B_{v3}x^{3n/4}$. Product of A_vB_v is calculated as given steps below:

Evaluation Part: A_v polynomial is evaluated in 7 different points $(0, 1, -1, 2, -2, -3, \infty)$, i.e, thanks to Toom-Cook 4-way method, A_v polynomial can be splitted to $2k-1$ polynomials have degree $n/4 - 1$ where $k = 4$.

General matrix definition is as follows:

$$\begin{bmatrix} A_v(0) \\ A_v(1) \\ A_v(-1) \\ A_v(2) \\ A_v(-2) \\ A_v(-3) \\ A_v(\infty) \end{bmatrix} = \begin{bmatrix} 0^0 & 0^1 & 0^2 & 0^3 \\ 1^0 & 1^1 & 1^2 & 1^3 \\ -1^0 & -1^1 & -1^2 & -1^3 \\ 2^0 & 2^1 & 2^2 & 2^3 \\ -2^0 & -2^1 & -2^2 & -2^3 \\ -3^0 & -3^1 & -3^2 & -3^3 \\ \infty^0 & \infty^1 & \infty^2 & \infty^3 \end{bmatrix} . \begin{bmatrix} A_{v0} \\ A_{v1} \\ A_{v2} \\ A_{v3} \end{bmatrix} \tag{2.16}$$

Similarly, B_v polynomial is evaluated in 7 points with the same way and evaluated polynomials in Equation 2.16 transposed as follows:

$$Let[A_v(0), A_v(1), ..., A_v(\infty)]^T = X^T, [B_v(0), B_v(1), ..., B_v(\infty)]^T = Y^T \tag{2.17}$$

$$\begin{aligned} X^T =& ((A_{v0}), (A_{v0} + A_{v1} + A_{v2} + A_{v3}), (A_{v0} - A_{v1} + A_{v2} - A_{v3}), \\ & (A_{v0} + 2A_{v1} + 4A_{v2} + 8A_{v3}), (A_{v0} - 2A_{v1} + 4A_{v2} - 8A_{v3}), \\ & (A_{v0} - 3A_{v1} + 9A_{v2} - 27A_{v3}), (A_{v3})) \\ Y^T =& ((B_{v0}), (B_{v0} + B_{v1} + B_{v2} + B_{v3}), (B_{v0} - B_{v1} + B_{v2} - B_{v3}), \\ & (B_{v0} + 2B_{v1} + 4B_{v2} + 8B_{v3}), (B_{v0} - 2B_{v1} + 4B_{v2} - 8B_{v3}), \\ & (B_{v0} - 3B_{v1} + 9B_{v2} - 27B_{v3}), (B_{v3})) \end{aligned} \tag{2.18}$$

Multiplication Part:

$$X^T Y^T = C_v^T = [C_v(0), C_v(1), C_v(-1), C_v(2), C_v(-2), C_v(-3), C_v(\infty)] \tag{2.19}$$

Result of component multiplication ($X^T Y^T$) can be calculated by the combination of Equation 2.18 and 2.19 as follows:

$$
\begin{aligned}
C_v(0) &= A_{v0}B_{v0} \\
C_v(1) &= (A_{v0} + A_{v1} + A_{v2} + A_{v3})(B_{v0} + B_{v1} + B_{v2} + B_{v3}) \\
C_v(-1) &= (A_{v0} - A_{v1} + A_{v2} - A_{v3})(B_{v0} - B_{v1} + B_{v2} - B_{v3}) \\
C_v(2) &= (A_{v0} + 2A_{v1} + 4A_{v2} + 8A_{v3})(B_{v0} + 2B_{v1} + 4B_{v2} + 8B_{v3}) \\
C_v(-2) &= (A_{v0} - 2A_{v1} + 4A_{v2} - 8A_{v3})(B_{v0} - 2B_{v1} + 4B_{v2} - 8B_{v3}) \\
C_v(3) &= (A_{v0} - 3A_{v1} + 9A_{v2} - 27A_{v3})(B_{v0} - 3B_{v1} + 9B_{v2} - 27B_{v3}) \\
C_v(\infty) &= A_{v3}B_{v3}
\end{aligned}
\tag{2.20}
$$

Interpolation Part:

General matrix definition is given below:

$$
\begin{bmatrix} C_v(0) \\ C_v(1) \\ C_v(-1) \\ C_v(2) \\ C_v(-2) \\ C_v(-3) \\ C_v(\infty) \end{bmatrix}
=
\begin{bmatrix}
0^0 & 0^1 & 0^2 & 0^3 & 0^4 & 0^5 & 0^6 \\
1^0 & 1^1 & 1^2 & 1^3 & 1^4 & 1^5 & 1^6 \\
-1^0 & -1^1 & -1^2 & -1^3 & -1^4 & -1^5 & -1^6 \\
2^0 & 2^1 & 2^2 & 2^3 & 2^4 & 2^5 & 2^6 \\
-2^0 & -2^1 & -2^2 & -2^3 & -2^4 & -2^5 & -2^6 \\
-3^0 & -3^1 & -3^2 & -3^3 & -3^4 & -3^5 & -3^6 \\
\infty^0 & \infty^1 & \infty^2 & \infty^3 & \infty^4 & \infty^5 & \infty^6
\end{bmatrix}
\cdot
\begin{bmatrix} C_{v0} \\ C_{v1} \\ C_{v2} \\ C_{v3} \\ C_{v4} \\ C_{v5} \\ C_{v6} \end{bmatrix}
\tag{2.21}
$$

The equation 2.21 is converted to 2.22 by inverting the matrix as follow:

$$\begin{bmatrix} C_{v0} \\ C_{v1} \\ C_{v2} \\ C_{v3} \\ C_{v4} \\ C_{v5} \\ C_{v6} \end{bmatrix} = \begin{bmatrix} 0^0 & 0^1 & 0^2 & 0^3 & 0^4 & 0^5 & 0^6 \\ 1^0 & 1^1 & 1^2 & 1^3 & 1^4 & 1^5 & 1^6 \\ -1^0 & -1^1 & -1^2 & -1^3 & -1^4 & -1^5 & -1^6 \\ 2^0 & 2^1 & 2^2 & 2^3 & 2^4 & 2^5 & 2^6 \\ -2^0 & -2^1 & -2^2 & -2^3 & -2^4 & -2^5 & -2^6 \\ -3^0 & -3^1 & -3^2 & -3^3 & -3^4 & -3^5 & -3^6 \\ \infty^0 & \infty^1 & \infty^2 & \infty^3 & \infty^4 & \infty^5 & \infty^6 \end{bmatrix}^{-1} \cdot \begin{bmatrix} C_v(0) \\ C_v(1) \\ C_v(-1) \\ C_v(2) \\ C_v(-2) \\ C_v(-3) \\ C_v(\infty) \end{bmatrix} \tag{2.22}$$

Next, $[C_{v0}, C_{v1}, C_{v2}, C_{v3}, C_{v4}, C_{v5}, C_{v6}]$ is found from above general expression of interpolation part and result polynomial reconstructed as follows:

Reconstruction Part:

$$\begin{aligned} C_v &= A_v B_v \\ &= C_{v0} + C_{v1}x^{n/4} + C_{v2}x^{2n/4} + C_{v3}x^{3n/4} + C_{v4}x^{4n/4} + C_{v5}x^{5n/4} + C_{v6}x^{6n/4} \end{aligned} \tag{2.23}$$

To calculate the complexity of Toom Cook 4-way method, the analysis from the Equation 2.22 continues below:

$$\begin{bmatrix} C_{v0} \\ C_{v1} \\ C_{v2} \\ C_{v3} \\ C_{v4} \\ C_{v5} \\ C_{v6} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & -1 & -\frac{1}{20} & \frac{1}{4} & -\frac{1}{30} & 12 \\ -\frac{5}{4} & \frac{2}{3} & \frac{2}{3} & -\frac{1}{24} & -\frac{1}{24} & 0 & 4 \\ -\frac{5}{12} & \frac{1}{24} & \frac{7}{12} & \frac{1}{24} & -\frac{7}{24} & \frac{1}{24} & -15 \\ \frac{1}{4} & -\frac{1}{6} & -\frac{1}{6} & \frac{1}{24} & -\frac{1}{24} & 0 & -5 \\ \frac{1}{12} & -\frac{1}{24} & -\frac{1}{12} & \frac{1}{120} & \frac{1}{24} & -\frac{1}{120} & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} C_v(0) \\ C_v(1) \\ C_v(-1) \\ C_v(2) \\ C_v(-2) \\ C_v(-3) \\ C_v(\infty) \end{bmatrix} \quad (2.24)$$

Interpolation part is resumed in Equation 2.25 and the complexity of the method is calculated in terms of blocks' complexity as given in Equation 2.26.

$$
\begin{aligned}
C_{v0} &= C_v(0) = A_{v0}B_{v0} \\
C_{v1} &= \frac{1}{3}C_v(0) + \frac{1}{2}C_v(1) - C_v(-1) - \frac{1}{20}C_v(2) + \frac{1}{4}C_v(-2) - \frac{1}{30}C_V(-3) + 12C_v(\infty) \\
C_{v2} &= -\frac{5}{4}C_v(0) + \frac{2}{3}C_v(1) - \frac{2}{3}C_v(-1) - \frac{1}{24}C_v(2) - \frac{1}{24}C_v(-2) + 4C_v(\infty) \\
C_{v3} &= -\frac{5}{12}C_v(0) + \frac{1}{24}C_v(1) - \frac{7}{12}C_v(-1)) - \frac{1}{24}C_v(2) + \frac{-7}{24}C_v(-2) + \frac{1}{24}C_V(-3) \\
&\quad - 15C_v(\infty) \\
C_{v4} &= \frac{1}{4}C_v(0) - \frac{1}{6}C_v(1) - \frac{1}{6}C_v(-1) + \frac{1}{24}C_v(2) + \frac{1}{24}C_v(-2) - 5C_v(\infty) \\
C_{v5} &= \frac{1}{12}C_v(0) - \frac{1}{24}C_v(1) - \frac{1}{12}C_v(-1)) + \frac{1}{120}C_v(2) + \frac{1}{24}C_v(-2) - \frac{1}{120}C_V(-3) \\
&\quad + 3C_v(\infty) \\
C_{v6} &= C_v(\infty) = A_{v3}B_{v3}
\end{aligned}
\tag{2.25}
$$

$$
M(n) = \underbrace{7M(n/4)}_{CM} + \underbrace{30(\frac{n}{2} - 1) + 6(\frac{n}{4} - 1)}_{R} + \underbrace{30(\frac{n}{4})}_{CPF} \tag{2.26}
$$

From Equation 2.25 and 2.26, reconstruction cost comes from the interpolation part required 30 polynomial additions between polynomials degree with $(\frac{n}{2}) - 2$ and that causes computational cost as $30(\frac{n}{2}-1)$. Cost of overlap, $6(\frac{n}{4}-1)$ also comes from the addition of coefficients of the result polynomial ($C_0 + C_1x^{n/4} + C_2x^{2n/4} + C_3x^{3n/4} + C_4x^{4n/4} + C_5x^{35n/4} + C_6x^{6n/4}$). Cost of polynomial evaluation at 7 different points, i.e., CPF part have 30 polynomial additions between polynomials degree with $(\frac{n}{4}-1)$ whose computational cost equal to $30(\frac{n}{4})$. Cost of multiplication of evaluated polynomials, i.e., CM part have 7 polynomial multiplications between polynomials degree with $(\frac{n}{4} - 1)$ that causes $7M(n/4)$ computational cost. At this point, complexity is

derived as follows: (Note that: $n = 4^l$ and $7^l = n^{1.40}$):

$$\begin{aligned}
M(n) &= 7M(n/4) + 30(\frac{n}{2} - 1) + 6(\frac{n}{4} - 1) + 30(\frac{n}{4}) \\
&= 7M(\frac{n}{4}) + \frac{48n}{2} - 36 \\
&= 7(7M(\frac{n}{16}) + \frac{48n}{8} - 33) + \frac{48n}{2} - 36) \\
&= 7^2M(\frac{n}{16}) + 7(\frac{48n}{8}) + \frac{48n}{2} - 8(36) \\
&= 7^2((7M(\frac{n}{64}) + \frac{48n}{32} - 36) + 11(\frac{48n}{8} - 8(36)) \\
&= 7^3M(\frac{n}{4^3}) + \frac{48n}{2}(\frac{7^2}{4^2} + \frac{7}{4} + 1) - 36(7^2 + 7 + 1) \\
&= 7^l + 24n(\frac{\frac{7}{4}^l - 1}{\frac{7}{4} - 1}) - 36(\frac{7^l - 1}{7 - 1}) \\
&= 7^l + 24n\frac{4}{3}(\frac{7}{4}^l - 1) - 6(7^l - 1) \\
&= 7^l + 32(7^l) - 32n - 6(7^l) + 6 \\
&= 27n^{1.40} - 32n + 6
\end{aligned} \tag{2.27}$$

CPF and CM blocks' complexity is given $CPF(n) = (n^{\log_4 7} - n)$ and $CM(n) = n^{\log_4 7}$ in [7]. Complexity of reconstruction block is found in Equation 2.28 and hence all complexity of blocks CPF, CM, R shown in Table 2.5.

$$\begin{aligned}
R(n) &= M(n) - 2CPF(n) - CM(n) \\
&= 24n^{1.40} - 30n + 6
\end{aligned} \tag{2.28}$$

Additionally, reconstruction part of the algorithm is optimized in [28] and that explained as follows:

Table 2.5: Block complexities of Toom Cook-4 way method

Blocks	Complexity
$CPF(n)$	$n^{\log_4 7} - n$
$CM(n)$	$n^{\log_4 7}$
$R(n)$	$24n^{\log_4 7} - 30n + 6$

Consider $C(x, y) = A(x, y)B(x, y)$. First of all, $C(x, y)$ are calculated at 7 points.

$$
\begin{aligned}
C(0,1) &= c_0 = a_0 b_0 \\
C(1,0) &= c_6 = a_3 b_3 \\
C(1,1) &= c_6 + c_5 + c_4 + c_3 + c_2 + c_1 + c_0 \\
&= (a_3 + ... + a_0)(b_3 + ... + b_0) \\
C(1,-1) &= c_6 - c_5 + c_4 - c_3 + c_2 - c_1 + c_0 \\
&= (-a_3 + ... + a_0)(-b_3 + ... + b_0) \\
C(2,1) &= 64c_6 + 32c_5 + 16c_4 + 8c_3 + 4c_2 + 2c_1 + c_0 \\
&= (8a_3 + 4a_2 + 2a_1 + a_0)(8b_3 + 4b_2 + 2b_1 + b_0) \\
C(2,-1) &= 64c_6 - 32c_5 + 16c_4 - 8c_3 + 4c_2 - 2c_1 + c_0 \\
&= (-8a_3 + 4a_2 - 2a_1 + a_0)(-8b_3 + 4b_2 - 2b_1 + b_0) \\
C(1,2) &= c_6 + 2c_5 + 4c_4 + 8c_3 + 16c_2 + 32c_1 + 64c_0 \\
&= (a_3 + 2a_2 + 4a_1 + 8a_0)(b_3 + 2b_2 + 4b_1 + 8b_0)
\end{aligned}
\tag{2.29}
$$

Secondly, coefficients of the result polynomial $(C_0, C_1, ..., C_6)$ are found below:

$$
\begin{aligned}
V_1 &= C(1,1) - c_6 - c_0 \\
V_2 &= c_6 + c_0 - C(1,-1) \\
V_3 &= (C(2,1) - c_0)/2 - 32c_6 \\
V_4 &= 32c_6 + (c_0 - C(2,-1))/2 \\
V_5 &= (C(1,2) - c_6)/2 - 32c_0 \\
V_6 &= (V_1 - V_2)/2 \\
V_7 &= (V_3 - V_4)/2 \\
B_1 &= V_1 - c_2 - c_4 \\
B_2 &= V_3 - 8c_4 - 2c_2 \\
B_3 &= V_5 - 2c_4 - 8c_2 \\
B_4 &= (B_2 - 4B_1)/3 \\
B_5 &= (B_3 - 4B_1)/3 \\
B_6 &= (B_4 + B_5)/3 \\
c_1 &= (B_5 + B_6)/5 \\
c_2 &= V_6 - c_4 \\
c_3 &= B_1 - c_5 - c_1 \\
c_4 &= (V_7 - V_6)/3 \\
c_5 &= B_6 - c_1
\end{aligned}
\tag{2.30}
$$

When Equations 2.29 and 2.30 are analyzed, thanks to this method, 30 algebraic additions decrease to 27 in reconstruction part. Therefore total complexity reduce to $25n^{\log_4 7} - 32n + 6$ and reconstruction complexity given in Table 2.5 is updated in Table 2.6.

Table 2.6: Block complexities of Toom Cook-4 way method with optimized reconstruction part

Blocks	Complexity
$CPF(n)$	$n^{\log_4 7} - n$
$CM(n)$	$n^{\log_4 7}$
$R(n)$	$22n^{\log_4 7} - 30n + 6$

CHAPTER 3

ANALYZES OF BLOCK RECOMBINATION AND LAZY INTERPOLATION METHODS

In Toom-Cook and Karatsuba based multiplication algorithms, polynomials are multiplied with smaller degrees than the original polynomials. Before and after the polynomial multiplications, pre-and post-processing operations are required as we told in Section 2.4. In this section, we explain how the efficiency of post-processing operations increased using the two novel methods, block recombination and lazy interpolation. They are analyzed to show the equivalence by considering the advantages they provide and the techniques they use. Block recombination method introduced and applied to Bernstein 4-way formula in [7] while lazy interpolation introduced and applied to Toom-Cook 4-way method in [23]. In the last part of this section, their comparisons with classical approaches demonstrate the advantages of the two methods in the scenario with the sum of two / three polynomial multiplications.

3.1 Lazy Interpolation Method

The lazy interpolation method [23] reduces the complexity of the Toom-Cook and Karatsuba algorithms by decreasing interpolation stages of the algorithms. Toom-Cook algorithm consists of TC (Toom-Cook transformation) and TC^{-1} (Toom-Cook reverse transformation), i.e., TC means the evaluation stage of algorithm and TC^{-1} refers to interpolation stage of the algorithm to reconstruct the resulting polynomial. In the TC step, polynomials are divided into polynomials with small degrees in which polynomials are multiplied in CM (component multiplication) stage. CA (component addition) contains arithmetic additions to accumulate polynomial pairs before interpolation stages. The final part is TC^{-1} consisting of interpolation stages to reconstruct polynomials. In equation 3.1, where A_i, B_i polynomials are degree with $n-1$ and $A_0B_0 + A_1B_1$ is computed with classical approach. $\hat{A}_0$ and $\hat{B}_0$ are the result of TC transformations of the A_0, B_0.

$$\begin{aligned} S &= (A_0B_0) + (A_1B_1) \\ &= TC^{-1}[TC(A_0) \times TC(B_0)] + TC^{-1}[TC(A_1) \times TC(B_1)] \\ &= TC^{-1}[CM(\hat{A}_0, \hat{B}_0)] + TC^{-1}[CM(\hat{A}_1, \hat{B}_1)] \\ &= TC^{-1}[\hat{C}_0] + TC^{-1}[\hat{C}_1] \end{aligned} \tag{3.1}$$

In Equation 3.1, the strategy of operation sequence: "two Toom-Cook reverse transformations (TC^{-1}) and then add" can be reversed to reduce the required number of Toom-Cook reverse transformation TC^{-1} by using the lazy interpolation method in Equation 3.2

$$
\begin{aligned}
S &= (A_0 B_0) + (A_1 B_1) \\
&= TC^{-1}[(TC(A_0) \times TC(B_0) + TC(A_1) \times TC(B_1)] \\
&= TC^{-1}[CM(\hat{A}_0, \hat{B}_0) + CM(\hat{A}_1, \hat{B}_1)] \\
&= TC^{-1}[\hat{C}_0 + \hat{C}_1] = TC^{-1}[CA(\hat{C}_0, \hat{C}_1)]
\end{aligned}
\tag{3.2}
$$

When Equation 3.1 and 3.2 are compared, main advantage of the lazy interpolation method is using less reverse Toom-Cook transformation (TC^{-1}), i.e. interpolation stage whose complexity is given in Table 2.5. Since most of the computational cost comes from the interpolation stage compared to the other stages, applying lazy interpolation to the Toom-Cook algorithm improves performance, as explained in the following section.

3.1.1 Toom-Cook 4-way with Lazy Interpolation Method

After the evaluation and multiplication step as described in Section 2.4.5, the result is not sent through the interpolation part this time. First, polynomials are calculated in evaluation points. Second, the multiplication results of these polynomials are accumulated then the interpolation part is used to reconstruct polynomials. Therefore, this method is named lazy interpolation in [23] to reduce the number of interpolation parts.

To calculate two polynomial multiplications ($A_v B_v$+$A_w B_w$) using Toom-Cook multiplication and add them together in the classical approach. Thanks to the lazy interpolation method, multiplied polynomials are accumulated before the interpolation stage. Therefore, it is enough to use the interpolation stage of the Toom-Cook algo-

rithm once instead of using two. In details, back to the evaluation part of the section 2.4.5, this time let us examine how to get the result of A_vB_v+A_wB_w.

$$Let[A_w(0), A_w(1), ..., A_w(\infty)]^T = Z^T, [B_w(0), B_w(1), ..., B_w(\infty)]^T = W^T \tag{3.3}$$

where A_w and B_w are evaluated in 7 points in same way with Equation 2.16

$$\begin{aligned}
Z^T =&((A_{w0}), (A_{w0} + A_{w1} + A_{w2} + A_{w3}), (A_{w0} - A_{w1} + A_{w2} - A_{w3}),\\
&(A_{w0} + 2A_{w1} + 4A_{w2} + 8A_{w3}), (A_{w0} - 2A_{w1} + 4A_{w2} - 8A_{w3}),\\
&(A_{w0} - 3A_{w1} + 9A_{w2} - 27A_{w3}), (A_{w3}))\\
W^T =&((B_{w0}), (B_{w0} + B_{w1} + B_{w2} + B_{w3}), (B_{w0} - B_{w1} + B_{w2} - B_{w3}),\\
&(B_{w0} + 2B_{w1} + 4B_{w2} + 8B_{w3}), (B_{w0} - 2B_{w1} + 4B_{w2} - 8B_{w3}),\\
&(B_{w0} - 3B_{w1} + 9B_{w2} - 27B_{w3}), (B_{w3}))
\end{aligned} \tag{3.4}$$

From the Equation 2.19, component multiplication of A_vB_v+A_wB_w given below:

$$\begin{aligned}
X^TY^T = C_v^T = [C_v(0), C_v(1), ..., C_v(\infty)]\\
Z^TW^T = C_w^T = [C_w(0), C_w(1), ..., C_w(\infty)]
\end{aligned} \tag{3.5}$$

In this step, instead of calculating C_v and C_w separately and then adding them together as $C^T = C_v^T + C_w^T$ in the classical approach. The process continues with lazy interpolation method as follows:

$$
\begin{aligned}
C^T &= [\underbrace{(C_v(0)+C_w(0))}_{C(0)}, \underbrace{(C_v(1)+C_w(1))}_{C(1)}, ..., \underbrace{(C_v(\infty)+C_w(\infty))}_{C(\infty)}] \\
C(0) &= (A_{v0}B_{v0} + A_{w0}B_{w0}) \\
C(1) &= (A_{v0}+A_{v1}+A_{v2}+A_{v3})(B_{v0}+B_{v1}+B_{v2}+B_{v3}) \\
&\quad + (A_{w0}+A_{w1}+A_{w2}+A_{w3})(B_{w0}+B_{w1}+B_{w2}+B_{w3}) \\
C(-1) &= (A_{v0}-A_{v1}+A_{v2}-A_{v3})(B_{v0}-B_{v1}+B_{v2}-B_{v3}) \\
&\quad + (A_{w0}-A_{w1}+A_{w2}-A_{w3})(B_{w0}-B_{w1}+B_{w2}-B_{w3}) \\
C(2) &= (A_{v0}+2A_{v1}+4A_{v2}+8A_{v3})(B_{v0}+2B_{v1}+4B_{v2}+8B_{v3}) \\
&\quad + (A_{w0}+2A_{w1}+4A_{w2}+8A_{w3})(B_{w0}+2B_{w1}+4B_{w2}+8B_{w3}) \\
C(-2) &= (A_{v0}-2A_{v1}+4A_{v2}-8A_{v3})(B_{v0}-2B_{v1}+4B_{v2}-8B_{v3}) \\
&\quad + (A_{w0}-2A_{w1}+4A_{w2}-8A_{w3})(B_{w0}-2B_{w1}+4B_{w2}-8B_{w3}) \\
C(3) &= (A_{v0}-3A_{v1}+9A_{v2}-27A_{v3})(B_{v0}-3B_{v1}+9B_{v2}-27B_{v3}) \\
&\quad + (A_{w0}-3A_{w1}+9A_{w2}-27A_{w3})(B_{w0}-3B_{w1}+9B_{w2}-27B_{w3}) \\
C(\infty) &= (A_{v3}B_{v3} + A_{w3}B_{w3})
\end{aligned}
\tag{3.6}
$$

Now, result is goes to interpolation part to calculate $C_0, C_1, C_2, C_3, C_4, C_5, C_6$ from $C(0), C(1), C(-1), C(2), C(-2), C(3), C(\infty)$ as described in Equation 2.22. Next, output of interpolation part is reconstructed as like in Equation 2.23 to get result polynomial $C = C_0+C_1x^{n/4}+C_2x^{2n/4}+C_3x^{3n/4}+C_4x^{4n/4}+C_5x^{5n/4}+C_6x^{6n/4}$ Thanks to the lazy interpolation method, the interpolation part is used once no matter how many polynomial products are summed. Instead of using two interpolation stages, once is enough to get a result with little overhead cost causing the extra accumulation, which is explained in the following implementation section.

3.2 Block Recombination Method

Block recombination method [7, 16] reduces the complexity of the Toom-Cook and Karatsuba algorithms. Polynomial multiplication algorithms are divided into four blocks CPF, CM, R, CA. First, CPF represents the evaluation stages of multiplication algorithms, where polynomials are split into polynomials with a small degree. Second, CM is the multiplication stage, where the multiplication method is determined by polynomial degree. For example, if the split polynomials have degree 16, the schoolbook method is the most efficient way to multiply polynomials [18]. CA (component addition)contains arithmetic additions to accumulate polynomial pairs before interpolation stages. Finally, the reconstruction part consists of the interpolation stages is used to reconstruct polynomials.

To understand the advantage of the block recombination method, first, multiply two pairs of polynomials and then add them together with the classical approach as shown in Equation 3.7 where A_i, B_i polynomials are degrees with $n-1$.

$$\begin{aligned} S &= (A_0 B_0) + (A_1 B_1) \\ &= R[CPF(A_0) \times CPF(B_0)] + R[CPF(A_1) \times CPF(B_1)] \\ &= R[CM(\hat{A}_0, \hat{B}_0)] + R[CM(\hat{A}_1, \hat{B}_1)] \\ &= R[\hat{C}_0] + R[\hat{C}_1] \end{aligned} \tag{3.7}$$

In Equation 3.7, the strategy of operation sequence: "two reconstructions then add" can be reversed to recombine the blocks with block recombination method in Equation 3.8.

$$
\begin{aligned}
S &= (A_0 B_0) + (A_1 B_1) \\
&= R[(CPF(A_0) \times CPF(B_0) + CPF(A_1) \times CPF(B_1)] \\
&= R[CM(\hat{A}_0, \hat{B}_0) + CM(\hat{A}_1, \hat{B}_1)] \\
&= R[\hat{C}_0 + \hat{C}_1] = R[CA(\hat{C}_0, \hat{C}_1)]
\end{aligned}
\tag{3.8}
$$

When the required blocks of two strategy in Equation 3.7 and 3.8 compared, main advantage of the block recombination method is using less reconstruction blocks whose space complexity in the interval $[3n^{\rho}, 4n^{\rho}]$ where $\rho \in (\log_2 3, \log_3 6)$. On the other hand, component addition block comes out as an extra computational cost, it has total space complexity with n^{ρ} where $\rho \in (\log_2 3, \log_3 6)$. Therefore, at least $2n^{\rho}$ computational cost are saved thanks to this method.

3.2.1 Bernstein Four-way Split Formula with Block Recombination Method

Let A and B be two polynomials with the degree of $n-1$, where $n = 2^k$ for $k \geq 2$. A and B polynomials are divided into four $A = A_0 + A_1 x^{n/4} + A_2 x^{2n/4} + A_3 x^{3n/4}$ and $B = B_0 + B_1 x^{n/4} + B_2 x^{2n/4} + B_3 x^{3n/4}$. Product of A and B can be found as follows:

$$
\begin{aligned}
AB = & \underbrace{A_0B_0}_{C_0} + \underbrace{(A_0B_1 + A_1B_0)}_{C_1} x^{n/4} \\
& + \underbrace{(A_0B_2 + A_1B_1 + A_2B_0)}_{C_2} x^{2n/4} \\
& + \underbrace{(A_0B_3 + A_1B_2 + A_2B_1 + A_3B_0)}_{C_3} x^{3n/4} + \underbrace{(A_1B_3 + A_2B_2 + A_3B_1)}_{C_4} x^{4n/4} \\
& + \underbrace{(A_2B_3 + A_3B_2)}_{C_5} x^{5n/4} \\
& + \underbrace{A_3B_3}_{C_6} x^{6n/4}
\end{aligned}
\tag{3.9}
$$

According to the above expression, 9 component addition $CA(n/4)$, 7 reconstruction $R(n/4)$, 8 component polynomial transformation $CPF(n/4)$ and 16 component multiplication $CM(n/4)$ blocks are needed to evaluate the complexity of the method. Complexity of CPF, CM, R blocks are given in Table 2.4 and hence the total complexity of recombined Bernstein four way formula calculated [7] as follows:

$$
\begin{aligned}
& = 8CPF(n/4) + 16CM(n/4) + 7R(n/4) + 9CA(n/4) + 3n/2 - 6 \\
& = \frac{8}{9}CPF(n) + \frac{16}{9}CM(n) + \frac{7}{9}R(n) + 9CA(n/4) + 3n/2 - 6 \\
& = 6,34n^{\log_2 3} - 8,9n - 3,63
\end{aligned}
$$

where $CA(n) = (n^{\log_2 3})$

(3.10)

3.3 Comparisons of Block Recombination and Lazy Interpolation

In this section, computational costs of $\sum_{i=1}^{\tau} A_iB_i$, where $\tau \in (2, 3)$, are compared between Classical Toom-Cook 4-way method, Toom-Cook 4-way method with lazy

interpolation, Bernstein 4-way split formula and Bernstein 4-way split formula with block recombination method as described in sections 2.4.5, 3.1.1, 2.4.4 and 3.2.1 respectively.

Let A_v, A_w, A_z and B_v, B_w, B_z be 6 polynomials with the degree of $n-1$, where $n = 2^k$ for $k \geq 2$.

Toom-Cook 4-way Method: To calculate $(A_vB_v) + (A_wB_w)$, complexity is given in Equation 3.11 by using block complexity values in Table 2.5.

$$\begin{aligned} M(n) &= 2CM(n) + 2R(n) + 4CPF(n) + 2n - 1 \\ &= 2n^{1.4} + 2(24n^{1.4} - 30n + 6) + 4(n^{1.4} - n) + 2n - 1 \\ &= 54n^{1.4} - 62n + 7 \end{aligned} \tag{3.11}$$

Where $2n-1$ cost comes from the addition of polynomials A_vB_v and A_wB_w with degree of $2n-2$. Complexity of $(A_vB_v)+(A_wB_w)+(A_zB_z)$ is also found in Equation 3.12.

$$\begin{aligned} M(n) &= 3CM(n) + 3R(n) + 6CPF(n) + 4n - 2 \\ &= 3n^{1.4} + 3(24n^{1.4} - 30n + 6) + 6(n^{1.4} - n) + 2n - 1 \\ &= 81n^{1.4} - 92n + 16 \end{aligned} \tag{3.12}$$

Toom-Cook 4-way method with lazy interpolation: To calculate $(A_vB_v) + (A_wB_w)$, complexity is given in equation 3.13 by using block complexity values in Table 2.5.

$$\begin{aligned} M(n) &= 2CM(n) + R(n) + 4CPF(n) + CA(n) \\ &= 2n^{1.4} + (24n^{1.4} - 30n + 6) + 4(n^{1.4} - n) + n^{1.4} \\ &= 31n^{1.4} - 34n + 6 \end{aligned} \tag{3.13}$$

Where $CA(n) = n^{1.4}$. Complexity of calculating $(A_vB_v) + (A_wB_w) + (A_zB_z)$ is

given in Equation 3.14.

$$\begin{aligned} M(n) &= 3CM(n) + R(n) + 6CPF(n) + 2CA(n) \\ &= 3n^{1.4} + (24n^{1.4} - 30n + 6) + 6(n^{1.4} - n) + 2n^{1.4} \\ &= 35n^{1.4} - 36n + 6 \end{aligned} \tag{3.14}$$

Bernstein 4-way method: To calculate $(A_vB_v) + (A_wB_w)$, complexity is given in Equation 3.15 by using block complexity values in Table 2.4.

$$\begin{aligned} M(n) &= 2CM(n) + 2R(n) + 4CPF(n) + 2n - 1 \\ &= 4(n^{\log_2 3} - n) + 2n^{\log_2 3} + 2(\frac{137}{40}n^{\log_2 3} - \frac{24n}{5} + \frac{11}{8}) + 2n - 1 \\ &= 12,85n^{1.58} - 11.6n + 1.75 \end{aligned} \tag{3.15}$$

Complexity of calculating $(A_vB_v) + (A_wB_w) + (A_zB_z)$ is also found in Equation 3.16.

$$\begin{aligned} M(n) &= 3CM(n) + 3R(n) + 6CPF(n) + 4n - 2 \\ &= 6(n^{\log_2 3} - n) + 3n^{\log_2 3} + 3(\frac{137}{40}n^{\log_2 3} - \frac{24n}{5} + \frac{11}{8}) + 4n - 2 \\ &= 19.28n^{1.58} - 16.4n + 2.13 \end{aligned} \tag{3.16}$$

Bernstein 4-way method with block recombination: To calculate $(A_vB_v) + (A_wB_w)$, complexity is given in Equation 3.17 by using block complexity values in Table 2.4.

$$\begin{aligned} M(n) &= 16CPF(n/4) + 32CM(n/4) + 7R(n/4) + 18CA(n/4) + 7CA(n/4) \\ &\quad + 3n - 12 \\ &= \frac{16}{9}CPF(n) + \frac{32}{9}CM(n) + \frac{7}{9}R(n) + 25CA(n/4) + 3n - 12 \\ &= \frac{16}{9}(n^{\log_2 3} - n) + \frac{32}{9}n^{\log_2 3} + \frac{7}{9}(\frac{137}{40}n^{\log_2 3} - \frac{24n}{5} + \frac{11}{8}) \\ &\quad + \frac{25}{9}n^{log_2 3} + 3n - 12 \\ &= 10,78n^{1.58} - 2.50n - 10.94 \end{aligned} \tag{3.17}$$

Complexity of calculating $(A_vB_v) + (A_wB_w) + (A_zB_z)$ is also found in Equation 3.18.

$$\begin{aligned} M(n) &= 24CPF(n/4) + 48CM(n/4) + 7R(n/4) + 27CA(n/4) + 14CA(n/4) \\ &\quad + 4.5n - 18 \\ &= \frac{24}{9}CPF(n) + \frac{48}{9}CM(n) + \frac{7}{9}R(n) + 41CA(n/4) + 4.5n - 18 \\ &= \frac{24}{9}(n^{\log_2 3} - n) + \frac{48}{9}n^{\log_2 3} + \frac{7}{9}(\frac{137}{40}n^{\log_2 3} - \frac{24n}{5} + \frac{11}{8}) \\ &\quad + \frac{41}{9}n^{log_2 3} + 4.5n - 18 \\ &= 15.21n^{1.58} - 6.11n - 16.9 \end{aligned} \tag{3.18}$$

As a result, required minimum number of operations to calculate $A_vB_v + A_wB_w$ ($M(n)$ for $\tau = 2$) and $A_vB_v + A_wB_w + A_zB_z$ ($M(n)$ for $\tau = 3$) are given in the Table 3.1 and 3.2 for polynomial degree with $n = 256$ and $n = 64$ respectively. Results are calculated for given methods from calculated complexities in Equations 3.11, 3.12, 3.13, 3.14, 3.15, 3.16, 3.17, 3.18.

Table 3.1: Minimum number of operations with given methods for $n = 256$

Method	M(n) for $\tau = 2$	M(n) for $\tau = 3$
Classical Toom-Cook 4-way	111175	167019
Toom-Cook 4-way with lazy interpolation	64230	73128
Bernstein 4-way formula	79052	118866
Bernstein 4-way with block recombination	68156	95503

The equivalence of block recombination and lazy interpolation methods is explained as follows:

Table 3.2: Minimum number of operations with given methods for $n = 64$

Method	M(n) for $\tau = 2$	M(n) for $\tau = 3$
Classical Toom-Cook 4-way	14279	21489
Toom-Cook 4-way with lazy interpolation	8301	9524
Bernstein 4-way formula	8435	12720
Bernstein 4-way with block recombination	7527	10453

It is clear from Table 3.1 and 3.2, the required minimum number of operations decreases in case of using lazy interpolation and block recombination methods. When the cost of all blocks analyzed for Toom-Cook and Bernstein 4-way formula in Table 2.5 and 2.4, most of the computational cost belongs to reconstruction blocks that consist of interpolation stages of the multiplication algorithm. Therefore, reducing interpolation stages by lazy interpolation or block recombination method with the same techniques leads to decrease computational cost of polynomial multiplications. Especially for matrix-vector products, increased performance impact of lazy interpolation and block recombination method while the number of multiplied polynomial pairs increases (A_iB_i) also observed in Table 3.1 and 3.2. The number of polynomial pairs indicates the dimension of the matrix that depends on the security level of the algorithm.

The minimum number of operations required by block recombination and lazy interpolation methods can slightly differ from each other depending degree of polynomial to be multiplied ($n = 64, 256$) and how many pairs of polynomials ($\tau = 2, 3$) need to be added. Therefore, analyzes are given for Saber implementation considering recursive applications in the next section.

CHAPTER 4

APPLICATIONS TO SABER

In this section, first, we give brief information about the application of Saber with the classical approach and lazy interpolation approach. Second, we apply block recombination to Saber and compare it with the application given in [23]. The minimum number of operations required by our work using block recombination and application in [23] using lazy interpolation are calculated to show the equivalence of two methods in Saber implementation.

In Saber, two matrix-vector multiplications in Algorithms 4, 5 and two vector-dot products in Algorithms 5, 6 need to be performed. For Saber with level-3 security in Equation 4.1, $3x3$ matrix and $1x3$ vector multiplications performed as $A.S = B$, where each elements of matrix A and S (such as a_{00} and s_0) are polynomials have degree 256.

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} s_0 \\ s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} a_{00}.s_0 + a_{01}.s_1 + a_{02}.s_2 \\ a_{10}.s_0 + a_{11}.s_1 + a_{12}.s_2 \\ a_{20}.s_0 + a_{21}.s_1 + a_{22}.s_2 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} = B \quad (4.1)$$

For $256x256$ polynomial multiplication with classical approach, first, each poly-

nomial is split by Toom-Cook 4-way method, and this reduces from two polynomial multiplication have degree 256 to 7 polynomial multiplications have degree 64. Next, these polynomials keep on splitting using the iterative two-level Karatsuba 2-way method that produces nine polynomial multiplications with degree 16. Totally 63 polynomial multiplications, each of them having 16 degrees, are done by the schoolbook method. Then schoolbook multiplications are reconstructed by using iterative two-level Karatsuba 2-way interpolation stage following up Toom-Cook 4-way interpolation stage. Finally, after finding the result of each polynomial pair $(a_{00}.s_0), (a_{01}.s_1), (a_{01}.s_1)$ which are accumulated like that $a_{00}.s_0 + a_{01}.s_1 + a_{02}.s_2$ to get the result of first row (b_0) at matrix B in Equation 4.1. Polynomial multiplication steps of classical approach using iterative two-level Karatsuba and Toom-Cook 4-way formula (TC4+2KA) is given in Algorithm 7 at pseudocode level.

4.1 Lazy Interpolation Application to Saber

Iterative two-level application of Toom-Cook 4-way method with lazy interpolation is introduced as a combination of multiplication algorithm with the best performance results on general-purpose processors in [23]. Each element of matrix B in Equation 4.1 is calculated by the accumulation of polynomial multiplications. The number of interpolation stages of the Toom-Cook algorithm is reduced using the lazy interpolation method. The lazy interpolation method differs from the classical method with that component addition stage applied before sending the component multiplication result to Toom-Cook interpolation stages. In other words, instead of reconstructing to component multiplication result of each polynomial pair $(a_{00}s_0), (a_{01}s_1), (a_{02}s_2)$ by Toom-Cook interpolation algorithm, results of multiplications are accumulated. Then

Algorithm 7 Pseudocode of hybrid multiplication algorithm (TC4+2KA) with classical approach

Let $a_{00} = A_0$, $a_{01} = A_1$, $a_{02} = A_2$ and $b_0 = B_0$, $b_1 = B_1$, $b_2 = B_2$ in Equation 4.1

▷ Evaluation Stage

for i=0 to 2 **do**

$$CPF(A_i) = 2KA_{ev}(TC4_{ev}(A_i)) = A_i^j$$

$$CPF(B_i) = 2KA_{ev}(TC4_{ev}(B_i)) = B_i^j$$

where $j = 0$ to 62

▷ Multiplication Stage

for j=0 to 62 **do**

$$C_0^j = A_0^j * B_0^j = CM(A_0^j, B_0^j)$$

$$C_1^j = A_1^j * B_1^j = CM(A_1^j, B_1^j)$$

$$C_2^j = A_2^j * B_2^j = CM(A_2^j, B_2^j)$$

where each C_i polynomial have degree 31

▷ Karatsuba Interpolation Stage

for i=0 to 6 **do**

$$C_0^i = 2KA_{intp}(C_0^{i*9}, C_0^{(i*9)+1}, C_0^{(i*9)+2}, C_0^{(i*9)+3}, C_0^{(i*9)+4}, C_0^{(i*9)+5}, C_0^{(i*9)+6}, C_0^{(i*9)+7}, C_0^{(i*9)+8})$$

$$C_1^i = 2KA_{intp}(C_1^{i*9}, C_1^{(i*9)+1}, C_1^{(i*9)+2}, C_1^{(i*9)+3}, C_1^{(i*9)+4}, C_1^{(i*9)+5}, C_1^{(i*9)+6}, C_1^{(i*9)+7}, C_1^{(i*9)+8})$$

$$C_2^i = 2KA_{intp}(C_2^{i*9}, C_2^{(i*9)+1}, C_2^{(i*9)+2}, C_2^{(i*9)+3}, C_2^{(i*9)+4}, C_2^{(i*9)+5}, C_2^{(i*9)+6}, C_2^{(i*9)+7}, C_2^{(i*9)+8})$$

where each C_i polynomial have degree 127

▷ Toom-Cook Interpolation Stage

$$C_0 = TC4_{intp}(C_0^0, C_0^1, C_0^2, C_0^3, C_0^4, C_0^5, C_0^6)$$

$$C_1 = TC4_{intp}(C_1^0, C_1^1, C_1^2, C_1^3, C_1^4, C_1^5, C_1^6)$$

$$C_2 = TC4_{intp}(C_2^0, C_2^1, C_2^2, C_2^3, C_2^4, C_2^5, C_2^6)$$

▷ Addition Stage

$$C = C_0 + C_1 + C_2$$

$b_0 = C$ in Equation 4.1

Toom-Cook interpolation stage used recursively to find the b_0 result in 4.1.Polynomial multiplication steps of iterative two-level Toom-Cook 4-way formula (TC4+TC4) with lazy interpolation method is given in Algorithm 8 at pseudocode level.

Algorithm 8 Pseudocode of hybrid multiplication algorithm (TC4+TC4) in [23] using lazy interpolation

Let $a_{00} = A_0$, $a_{01} = A_1$, $a_{02} = A_2$ and $b_0 = B_0$, $b_1 = B_1$, $b_2 = B_2$ in Equation 4.1

▷ Evaluation Stage

for i=0 to 2 **do**

$$CPF(A_i) = TC4_{ev}(TC4_{ev}(A_i)) = A_i^j$$

$$CPF(B_i) = TC4_{ev}(TC4_{ev}(B_i)) = B_i^j$$

where $j = 0$ to 48

▷ Multiplication Stage

for j=0 to 48 **do**

$$C_0^j = A_0^j * B_0^j = CM(A_0^j, B_0^j)$$

$$C_1^j = A_1^j * B_1^j = CM(A_1^j, B_1^j)$$

$$C_2^j = A_2^j * B_2^j = CM(A_2^j, B_2^j)$$

▷ Addition Stage

for j=0 to 48 **do**

$$C^j = C_0^j + C_1^j + C_2^j = CA(C_0^j, C_1^j, C_2^j)$$

▷ Toom-Cook Interpolation Stage

for i=0 to 6 **do**

$$C_i = TC4_{intp}(C^{i*7}, C^{(i*7)+1}, C^{(i*7)+2}, C^{(i*7)+3}, C^{(i*7)+4}, C^{(i*7)+5}, C^{(i*7)+6})$$

where each C_i polynomial have degree 127

▷ Toom-Cook Interpolation Stage

$C = TC4_{intp}(C_0, C_1, C_2, C_3, C_4, C_5, C_6)$

$b_0 = C$ in Equation 4.1

4.2 Our Work

In Equation 4.1 to calculate b_0, first of all, evaluation algorithms of Toom-Cook 4-way and iterative two-level Karatsuba 2-way are used to evaluate each polynomials (a_{00}, a_{01}, a_{02}) at 63 point, i.e, splitting each polynomials have degree 256 to 63 polynomials have degree 16. Iterative two-level Karatsuba 2-way algorithm is implemented by using recombined Bernstein 4-way split formula as described in section 3.2.1. Second, the schoolbook method is applied to perform the component multiplication stage. After the component multiplication stage, results are accumulated instead of sending results to the Karatsuba interpolation stage. Then accumulated these results are reconstructed by iterative two-level Karatsuba 2-way and Toom-Cook 4-way interpolation methods, respectively. After the interpolation stages, result polynomial is reconstructed (i.e first row of result matrix B in Equation 4.1 is calculated). To find matrix-vector multiplication results for Saber with level-3 security, another second (b_1) and third (b_2) coefficients of a matrix B are calculated in the same way. Pseudocode of polynomial multiplication steps of Toom-Cook 4-way and iterative two-level Karatsuba 2-way with block recombination method is given Algorithm 9.

In Saber's implementation, whether using block recombination or lazy interpolation methods, hybrid multiplication algorithms can be implemented by Toom-Cook k-way and Karatsuba k-way functions. Both Toom-Cook and Karatsuba algorithms are divided into evaluation, multiplication, addition, and interpolation stages. The cost of the Toom-Cook and Karatsuba algorithms' evaluation and interpolation stages are calculated for $n = 64, 256$ in Table 4.1. The minimum number of operations required to obtain the matrix-vector product in Equation 4.1, are shown in Table 4.2, 4.3 and 4.4, for classical approach, method in [23], our work respectively. Block

Algorithm 9 Pseudocode of hybrid multiplication algorithm (TC4+2KA) in our work using block recombination

Let $a_{00} = A_0$, $a_{01} = A_1$, $a_{02} = A_2$ and $b_0 = B_0$, $b_1 = B_1$, $b_2 = B_2$ in Equation 4.1

▷ Evaluation Stage

for i=0 to 2 **do**

$$CPF(A_i) = 2KA_{ev}(TC4_{ev}(A_i)) = A_i^j$$

$$CPF(B_i) = 2KA_{ev}(TC4_{ev}(B_i)) = B_i^j$$

where $j = 0$ to 62

▷ Multiplication Stage

for j=0 to 62 **do**

$$C_0^j = A_0^j * B_0^j = CM(A_0^j, B_0^j)$$

$$C_1^j = A_1^j * B_1^j = CM(A_1^j, B_1^j)$$

$$C_2^j = A_2^j * B_2^j = CM(A_2^j, B_2^j)$$

▷ Addition Stage

for j=0 to 62 **do**

$$C^j = C_0^j + C_1^j + C_2^j = CA(C_0^j, C_1^j, C_2^j)$$

where each C_i^j polynomial have degree 31

▷ Karatsuba Interpolation Stage

for i=0 to 6 **do**

$$C_i = 2KA_{intp}(C^{i*9}, C^{(i*9)+1}, C^{(i*9)+2}, C^{(i*9)+3}, C^{(i*9)+4},$$
$$C^{(i*9)+5}, C^{(i*9)+6}, C^{(i*9)+7}, C^{(i*9)+8})$$

where each C_i polynomial have degree 127

▷ Toom-Cook Interpolation Stage

$$C = TC4_{intp}(C_0, C_1, C_2, C_3, C_4, C_5, C_6)$$

$b_0 = C$ in Equation 4.1

recombination and lazy interpolation methods provide an 80% percent improvement in the minimum number of operations required for the matrix-vector product. This improvement is provided by the development made in the interpolation stage, as we explained above sections. As can be seen from Table 4.3 and 4.4, block recombination application 4.2 needs less operations in interpolation stages but more operations in other stages than the lazy interpolation application in [23]. To sum up, both methods given in Algorithm 8, 9 almost require the same number of arithmetic operations in total. Although lazy interpolation application to iterative two levels of Toom-Cook 4-way multiplication has low complexity, 32-bit processing requirement cause disadvantage on performance which is given and explained in next implementation section.

Table 4.1: Blocks' complexity of evaluation and interpolation stages of Toom-Cook 4-way and Bernstein 4-way formulas

Blocks	Complexity M(n)	n=64	n = 256
$TC4_{intp}(n)$	$24n^{\log_4 7} - 30n + 6$	6193	48786
$TC4_{ev}(n)$	$2n^{\log_4 7} - 2n$	548	4193
$2KA_{intp}(n)$	$\frac{137}{40}n^{\log_2 3} - \frac{24n}{5} + \frac{11}{8}$	2140	20634
$2KA_{ev}(n)$	$2n^{\log_4 7} - 2n$	1300	4193

Table 4.2: Minimum number of operations for Algorithm 7

Stages	Minimum Number of Operations
Evaluation	$9 * (7 * 2KA_{ev}(64) + TC4_{ev}(256)) = 119637$
Multiplication	$9 * 63 * (16^2) = 145152$
Interpolation	$9 * (7 * 2KA_{intp}(64) + TC4_{intp}(256)) = 573894$
Total: 838683	

Table 4.3: Minimum number of operations for Algorithm 8

Stages	Minimum Number of Operations
Evaluation	$9 * (7 * TC4_{ev}(64) + TC4_{ev}(256)) = 72261$
Multiplication	$9 * 49 * (16^2) = 112896$
Addition	$6 * 49 * 31 = 9114$
Interpolation	$3 * (7 * TC4_{intp}(64) + TC4_{intp}(256) = 276411$
Total: 470682	

Table 4.4: Minimum number of operations for Algorithm 9 with our work

Stages	Minimum Number of Operations
Evaluation	$9 * (7 * 2KA_{ev}(64) + TC4_{ev}(256)) = 119637$
Multiplication	$9 * 63 * (16^2) = 145152$
Addition	$6 * 63 * 31 = 11718$
Interpolation	$3 * (7 * 2KA_{intp}(64) + TC4_{intp}(256) = 191298$
Total: 467805	

CHAPTER 5

IMPLEMENTATION RESULTS FOR SABER

For Saber, we firstly compare hybrid polynomial multiplications combining Toom-Cook and Karatsuba algorithms in terms of performance results and show the best. Second, we implement our work using the block recombination method to Saber and compare with [23] using the lazy interpolation method. Benchmark tests for Saber crypto-scheme are performed on an Intel Core i7-7700HQ Kaby Lake processor running at 2800 MHz with Turbo Boost and Hyper-threading disabled. The operating system is Ubuntu 20.04.2 LTS with Linux Kernel 4.15.0, and all software is compiled with GCC-7.4.0. The benchmark results of C implementations for Saber with level-3 security are given and compared in terms of clock cycles of key generation, encapsulation, and decapsulation parts in Table 5.1 and 5.2. In Table 10, clock cycles are obtained for iterative two-level Toom-Cook 4-way method from source: `https://github.com/KULeuven-COSIC/TCHES2020_SABER` and reference code from source: `https://github.com/KULeuven-COSIC/SABER`.

Saber needs matrix-vector and vector-dot products, consisting of multiplications of polynomials with degree 256. In Table 5.1, different hybrid multiplications are compared, and iterative two-level Karatsuba 2-way following up Toom-Cook 4-way

(TC4+2KA) is the best selection for Saber. (TC4+2KA) have up to 7.5% better performance against its closest competitor (TC4+KA). For other combinations of multiplication algorithms (except (TC4+TC4)), school-book multiplications are implemented in the polynomial degree is not equal to 16. Performing school-book multiplications is more suitable to multiply polynomials that have degrees less than 16 [18]. Therefore, splitting polynomials up to a degree of 16 is enough and more efficient to implement school-book multiplication.

In Table 5.2, our proposed implementation, iterative two-level Karatsuba 2-way following up Toom-Cook 4-way using block recombination method is the best result and up to 13% faster than the nearest competitor. Iterative two-level Toom-Cook 4-way (TC4+TC4) using lazy interpolation method in [23], which has higher cycle values, even though it has almost the same minimum number of arithmetic operations with our work, because of the requirement of 32 bits data type instead of 16. This requirement comes from using iterative two-level interpolation of the Toom-Cook 4-way method during polynomial multiplication steps. In this case, performance results are affected dramatically; therefore, the cycle values of Saber almost double compared to the best result. It should also be taken into account that any arithmetic operation larger than 16-bits create an overflow on 16-bit processors like Cortex-M4 microcontrollers.

As can be concluded from Table 5.1 and 5.2, performance results of Saber implementation on general-purpose Intel processors show that the best result is achieved with iterative two-level Karatsuba 2-way following up Toom-Cook 4-way by our work using block recombination method. The other multiplication strategies in Table 5.1 have disadvantages because they exceed 16-bit operations or perform their school-book multiplications in the polynomial degree, which be greater and less than 16.

Therefore, the (TC4+2KA) multiplication strategy is more suitable for performing polynomial multiplications and further improved by using the block recombination method proposed in Saber.

Table 5.1: Compared clock cycles of hybrid multiplication algorithms

C implementations of Saber with level-3 security

Methods	Cycles for Keygen	Cycles for Enc.	Cycles for Dec.
TC4 + 2KA	150343	188656	208099
TC4 + KA3	208266	265395	304167
TC4 + KA	162019	195339	217831
TC3 + 2KA	229655	285431	334767
TC3 + KA3	289575	373371	440275
TC3 + TC3	376722	490040	586010
TC3 + TC3 + KA	223383	285665	329449
TC4 + TC4	333117	431219	511693
TC4 + TC3 + KA	334603	432201	513088
TC4 + TC4 + KA	282461	363278	427059
TC4 + TC4 + 2KA	294137	379469	446971

Note: KA3 refers to Karatsuba 3-way, 2KA means that iterative two-level Karatsuba 2-way method used. TC3 and TC4 represented as Toom-Cook 3-way and Toom-Cook 4-way method, respectively.

Table 5.2: Compared clock cycles of our work and other implementations

C implementations of Saber with level-3 security

Methods	Cycles for Keygen	Cycles for Enc.	Cycles for Dec.
TC4 + 2KA (4.2)	133985	168720	181905
TC4 + 2KA (reference)	150343	188656	208099
TC4 + TC4 [23]	277005	347040	410742

CHAPTER 6

CONCLUSION AND DISCUSSION

To summarize our work, we focus on Toom-Cook and Karatsuba algorithms in case of using block recombination and lazy interpolation methods whose analyzes are given for them theoretically and experimentally. We show the equivalences of block recombination and lazy interpolation methods. On the practical side, we compare the performance results for Saber to select an optimized hybrid multiplication algorithm. After that best result is improved further with our implementation using the block recombination method.

Toom-Cook and Karatsuba based polynomial multiplications are gaining importance day by day after NIST's PQC standardization procedure. Our ideas are considered in the implementation stages of multiplication algorithms to improve the performance results of all crypto-schemes, which are independent of Saber. We give the results of the C implementation of Saber on general-purpose Intel processors. Other platform-based applications can be optimized and might be good alternatives to Toom-Cook and Karatsuba algorithms. The outcomes of our study will give the direction on the efficient hybrid application usage of Toom-Cook and Karatsuba algorithms.

REFERENCES

[1] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, et al., Status report on the second round of the nist post-quantum cryptography standardization process, US Department of Commerce, NIST, 2020.

[2] M. R. Albrecht and A. Deo, Large modulus ring-lwe $\geq$ module-lwe, Cryptology ePrint Archive, Report 2017/612, 2017, `https://eprint.iacr.org/2017/612`.

[3] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, Cyrystals-kyber - algorithm specifications and supporting documentation (version 2.0), NIST Post-Quantum Cryptography Standardization Process, 2019, `https://pq-crystals.org/kyber/`.

[4] A. Banerjee, C. Peikert, and A. Rosen, Pseudorandom functions and lattices, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 719–737, Springer, 2012.

[5] A. Basso, J. M. B. Mera, J.-P. D'Anvers, A. Karmakar, S. S. Roy, M. Van Beirendonck, and F. Vercauteren, Saber: Mod-lwr based kem (round 3 submission), 2020.

[6] D. J. Bernstein, Batch binary edwards, in *Annual International Cryptology Conference*, pp. 317–336, Springer, 2009.

[7] M. Cenk, M. A. Hasan, and C. Negre, Efficient subquadratic space complexity binary polynomial multipliers based on block recombination, IEEE Transactions on Computers, 63(9), pp. 2273–2287, 2014.

[8] C. Chen, Q. Danba, J. Hoffstein, A. Hülsing, J. Rijneveld, J. M.Schanck, P. Schwabe, W. Whyte, and Z. Zhang, Algorithm specifications and supporting documentatiton (version 2), NIST Post-Quantum Cryptography Standardization Process, 2019.

[9] S. A. Cook and S. O. Aanderaa, On the minimum computation time of functions, Transactions of the American Mathematical Society, 142, pp. 291–314, 1969.

[10] J. W. Cooley and J. W. Tukey, An algorithm for the machine calculation of complex fourier series, Mathematics of computation, 19(90), pp. 297–301, 1965.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, MIT press, 2009.

[12] V. B. Dang, F. Farahmand, M. Andrzejczak, K. Mohajerani, D. T. Nguyen, and K. Gaj, Implementation and benchmarking of round 2 candidates in the nist post-quantum cryptography standardization process using hardware and software/hardware co-design approaches, Cryptology ePrint Archive: Report 2020/795, 2020.

[13] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem, in *International Conference on Cryptology in Africa*, pp. 282–305, Springer, 2018.

[14] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, Saber: Mod-lwr based kem (round 2 submission), URL: https://csrc. nist. gov/projects/post-quantum-cryptography/round-2-submissions. Citations in this document, 1(4.1), 2019.

[15] E. Fujisaki and T. Okamoto, Secure integration of asymmetric and symmetric encryption schemes, in *Annual international cryptology conference*, pp. 537–554, Springer, 1999.

[16] M. A. Hasan, N. Meloni, A. H. Namin, and C. Negre, Block recombination approach for subquadratic space complexity binary field multiplication based on toeplitz matrix-vector product, IEEE Transactions on Computers, 61(2), pp. 151–163, 2010.

[17] H. Jiang, Z. Zhang, L. Chen, H. Wang, and Z. Ma, Ind-cca-secure key encapsulation mechanism in the quantum random oracle model, revisited, in *Annual International Cryptology Conference*, pp. 96–125, Springer, 2018.

[18] F. Johansson, Arb: efficient arbitrary-precision midpoint-radius interval arithmetic, IEEE Transactions on Computers, 66(8), pp. 1281–1292, 2017.

[19] A. A. Karatsuba and Y. P. Ofman, Multiplication of many-digital numbers by automatic computers, in *Doklady Akademii Nauk*, volume 145, pp. 293–294, Russian Academy of Sciences, 1962.

[20] A. Langlois and D. Stehlé, Worst-case to average-case reductions for module lattices, Designs, Codes and Cryptography, 75(3), pp. 565–599, 2015.

[21] A. Langlois and D. Stehlé, Worst-case to average-case reductions for module lattices, Designs, Codes and Cryptography, 75(3), pp. 565–599, 2015.

[22] V. Lyubashevsky, C. Peikert, and O. Regev, On ideal lattices and learning with errors over rings, in *Annual international conference on the theory and applications of cryptographic techniques*, pp. 1–23, Springer, 2010.

[23] J. M. B. Mera, A. Karmakar, and I. Verbauwhede, Time-memory trade-off in toom-cook multiplication: an application to module-lattice based cryptography, IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 222–244, 2020.

[24] M. Mosca, Cybersecurity in an era with quantum computers: Will we be ready?, IEEE Security & Privacy, 16(5), pp. 38–41, 2018.

[25] P. Q. Nguyen and B. Vallée, *The LLL algorithm*, Springer, 2010.

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

[27] A. L. Toom, The complexity of a scheme of functional elements realizing the multiplication of integers, in *Soviet Mathematics Doklady*, volume 3, pp. 714–716, 1963.

[28] B. S. Verkhovsky, *Integer Algorithms in Cryptology and Information Assurance*, World Scientific, 2014.

[29] M. Yagisawa, Fully homomorphic encryption without bootstrapping., IACR Cryptol. EPrint Arch., 2015, p. 474, 2015.