NEW TMVP-BASED MULTIPLICATION ALGORITHMS FOR POLYNOMIAL QUOTIENT RINGS AND APPLICATION TO POST-QUANTUM CRYPTOGRAPHY

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

BY

İREM KESKİNKURT PAKSOY

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

JULY 2022

Approval of the thesis:

NEW TMVP-BASED MULTIPLICATION ALGORITHMS FOR POLYNOMIAL QUOTIENT RINGS AND APPLICATION TO POST-QUANTUM CRYPTOGRAPHY

submitted by **İREM KESKİNKURT PAKSOY** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Cryptography Department, Middle East Technical University** by,

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

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

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

Examining Committee Members:

Assoc. Prof. Dr. Ali Doğanaksoy
Mathematics, METU

Prof. Dr. Murat Cenk
Cryptography, IAM, METU

Assoc. Prof. Dr. Oğuz Yayla
Cryptography, IAM, METU

Assoc. Prof. Dr. Fatih Sulak
Mathematics, Atılım University

Assist. Prof. Dr. Erdem Alkım
Computer Science, Dokuz Eylül University

Date:

ABSTRACT

NEW TMVP-BASED MULTIPLICATION ALGORITHMS FOR POLYNOMIAL QUOTIENT RINGS AND APPLICATION TO POST-QUANTUM CRYPTOGRAPHY

Keskinkurt Paksoy, İrem
Ph.D., Department of Cryptography
Supervisor : Prof. Dr. Murat Cenk

July 2022, 90 pages

One of the quantum-safe cryptography research areas is lattice-based cryptography. Most lattice-based schemes need efficient algorithms for multiplication in polynomial quotient rings. The fastest algorithm known for multiplication is the Number Theoretic Transform (NTT), which requires certain restrictions on the parameters of the ring, such as prime modulus. Direct NTT application is not an option for some schemes that do not comply with these restrictions, e.g., the two finalists of the PQC standardization competition, Saber and NTRU, which use a power-of-two modulus. Toom-Cook and Karatsuba are the most commonly used non-NTT multiplication algorithms. Even though a method of using NTT in NTT-unfriendly rings with larger parameters that prevent any modular reduction on the original result is proposed, developing non-NTT multiplication algorithms can also improve the efficiency of multiplication in such rings.

In this thesis, we focused on developing Toeplitz Matrix-Vector Product (TMVP) based multiplication algorithms for PQC schemes. First, we propose new three- and four-way TMVP split formulas with five and seven multiplications. We choose Saber and NTRU schemes for our case study. We develop TMVP-based multiplication algorithms using the new four-way formula for the rings on which Saber and NTRU are defined. We also propose an improved version of the algorithm for Saber and present a padding method for NTRU to utilize TMVP split formulas.

Moreover, we implement the proposed algorithms on ARM Cortex-M4, which NIST recommends as an evaluation platform for PQC candidates on microprocessors. We improve performance and stack memory consumption compared to all Toom implementations. We also observe that our TMVP-based algorithms are faster than NTT for three of the parameter sets of NTRU, and they reduce the stack usage for all. We integrate our codes into state-of-the-art implementations of Saber and NTRU in the literature to see the effect of our algorithm on the total performance of the schemes. For Saber, our algorithm achieves improvements up to 18.6% in performance and up to 44.2% in memory consumption compared to the Toom method. For all parameter sets of NTRU, we reduce stack usage between $5.9\% - 20.9\%$ compared to Toom and $5.1\% - 19.3\%$ compared to NTT. Moreover, we observe performance improvements between $4.4\% - 17.5\%$ compared to Toom for all parameter sets. Except for one of the parameter sets of NTRU, our algorithms outperform the NTT method. Furthermore, we propose new formulas for non-square TMVP calculations and a new approach for deriving new TMVP split formulas using the non-square formulas. The arithmetic complexity calculations and theoretical efficiency comparisons are also presented in this thesis.

Keywords: Toeplitz matrix-vector product, lattice-based cryptography, post-quantum cryptography, Saber, NTRU, key encapsulation mechanism, polynomial multiplication, ARM Cortex-M4

ÖZ

POLİNOM HALKALARI İÇİN YENİ TMVP-TABANLI ÇARPIM ALGORİTMALARI VE QUANTUM-SONRASI KRİPTOGRAFİYE UYGULAMALARI

Keskinkurt Paksoy, İrem
Doktora, Kriptografi Bölümü
Tez Yöneticisi : Prof. Dr. Murat Cenk

Temmuz 2022, 90 sayfa

Kuantum bilgisayarlara karşı güvenli kriptografi araştırma alanlarından biri kafes tabanlı kriptografidir. Kafes tabanlı sistemlerin birçoğu, polinom bölüm halkalarında çarpma için verimli algoritmalara ihtiyaç duyar. Çarpma için bilinen en hızlı algoritma, halkanın parametrelerinde modülün asal olması gibi belirli kısıtlamalar gerektiren Sayı Kuramsal Dönüşümdür (NTT). Bu kısıtlamalara uymayan bazı şemalar için doğrudan NTT uygulaması bir seçenek değildir; örneğin, ikinin kuvveti bir modül kullanan PQC standardizasyon yarışmasının iki finalisti Saber ve NTRU gibi. Toom-Cook ve Karatsuba, NTT direkt uygulanamadığında en yaygın kullanılan çarpma algoritmalarıdır. NTT'ye uygun olmayan halkalarda, modüler indirgeme gerektirmeyen daha büyük parametreler ile NTT kullanımına olanak veren bir yöntem önerilmiş olsa da, NTT olmayan verimli çarpma algoritmaları geliştirmek de bu halkalardaki çarpma işlemini iyileştirebilir.

Bu tezde, kuantum-sonrası kriptografik (PQC) sistemler için Toeplitz Matris-Vektör Çarpımı (TMVP) tabanlı çarpma algoritmaları geliştirmeye odaklandık. İlk olarak, beş ve yedi çarpma gerektiren yeni üçlü ve dörtlü TMVP formüllerini önerdik. Uygulama için Saber ve NTRU şemalarını seçtik. Saber ve NTRU'nun tanımlandığı halkalar için yeni dörtlü formülü kullanarak TMVP tabanlı çarpma algoritmaları geliştirdik. Ayrıca, Saber için geliştirilen algoritmanın iyileştirilmiş bir versiyonunu sunduk ve NTRU şemasındaki çarpma işlemlerinde TMVP formüllerini kullanabilmek için bir doldurma yöntemi önerdik.

Ayrıca, önerilen algoritmaları, PQC adaylarının mikroişlemciler üzerinde değerlendirilmesi için NIST tarafından önerilen platform olan ARM Cortex-M4 üzerinde gerçekledik. Performans ve hafıza kullanımını, literatürdeki tüm Toom gerçeklemelerine kıyasla iyileştirdik. Ayrıca, TMVP tabanlı algoritmaların, NTRU'nun üç parametre seti için NTT'den daha hızlı olduğunu ve tümü için hafıza kullanımını azalttığını gözlemledik. Algoritmalarımızın şemalar üzerindeki etkisini görmek için, kodlarımızı literatürdeki en gelişmiş Saber ve NTRU gerçeklemelerine entegre ettik. Saber için önerilen algoritmamız, Toom yöntemine kıyasla performansta %18,6'ya ve bellek tüketiminde %44.2'ye varan iyileştirmeler sağlamıştır. NTRU'nun tüm parametreleri için, hafıza kullanımını Toom'a kıyasla %5,9-%20,9 ve NTT'ye kıyasla %5,1-%19,3 arasında azalttık. Ayrıca, tüm parametreler için Toom metoduna kıyasla %4.4-%17.5 arasında performans artışı gözlemledik. NTRU'nun bir tanesi dışındaki parametreleri için, algoritmalarımız NTT yönteminden daha iyi performans göstermiştir. Ayrıca, kare olmayan TMVP hesaplamaları için yeni formüller sunduk ve bunları kullanarak yeni TMVP formülleri türetmek için bir yaklaşım önerdik. Önerilen formüllerin aritmetik karmaşıklık hesaplamaları ve teorik verimlilik karşılaştırmaları da bu tezde sunulmaktadır.

Anahtar Kelimeler: Toeplitz matris-vektör çarpımı, kafes tabanlı kriptografi, kuantum sonrası kriptografi, Saber, NTRU, anahtar kapsülleme mekanizması, polinom çarpımı, ARM Cortex-M4

I dedicate this thesis to my beloved mother, Sabiş.
I wish you were still here.
I love you.
I miss you.
IXMCMLIV-XIIXMMXX

ACKNOWLEDGMENTS

I would like to express my very great appreciation to my thesis supervisor Prof. Dr. Murat Cenk, for his patient guidance, enthusiastic encouragement, and valuable advice during the development and preparation of this thesis. His willingness to give his time and share his experiences has brightened my path.

I also would like to thank Assist. Prof. Dr. Erdem Alkım, Dr. Y. Alper Bilgin, and H. Bartu Yünüak, for sparing their time and enlightening me with their valuable comments whenever I ask for help.

I thank The Council of Higher Education (YÖK) for supporting a part of this work under the 100/2000 Ph.D. scholarship program.

I thank The Scientific and Technological Research Council of Turkey (TÜBİTAK) for partially supporting this work under the 2211/C National Ph.D. Scholarship Program in the Priority Fields in Science and Technology.

I am grateful to my family for their unwavering support during this long journey. I thank my mother, Sabiş, for teaching me to overcome obstacles and not give up. Thanks to my father, İlker, for always supporting and encouraging me. I would not be who I am if it were not for my parents. Thanks to my aunts Nagiş and Rabiş, my cousin Yağmur, and my niece Güneş for being there for me. I also would like to express my appreciation to my in-laws, Paksoys and Duffys, for their support.

Big thanks to my BFF, Eser Güven, for his unconditional friendship. I appreciate him for so many things he has done for me.

Last but not least, thanks to my husband, Sinan Paksoy, for his support, love, and patience. His unfailing faith in me gives me strength and confidence when I need it most.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

TMVP	Toeplitz Matrix-Vector Product
SB	Schoolbook method
NIST	National Institute of Standards and Technology
PKE	Public-Key Encryption
DPKE	Deterministic Public-Key Encryption
KEM	Key Encapsulation Mechanism
DS	Digital Signature
IND-CPA	Indistinguishable under chosen plaintext attack
IND-CCA	Indistinguishable under chosen ciphertext attack
NTT	Number Theoretic Transform
NTRU	Number Theory Research Unit or Number Theorists 'R' Us
TMVP-k	k-split TMVP formula
TMVP-k_s	k-split TMVP formula that requires s multiplications
PQC	Post-Quantum Cryptography
SIMD	Single Instruction Multiple Data
FO	Fujisaki-Okamoto
DSP	Digital Signal Processing

CHAPTER 1

INTRODUCTION

Modern cryptography relies on hard-to-solve mathematical problems and deals with designing/developing algorithms based on these problems to provide security of private data. The main idea is to conceal data using an encryption key and ensure that only an authorized entity can reveal it using a decryption key. If encryption and decryption keys are generated from the same key material, then it is called symmetric (secret-key) cryptography; otherwise, it is called asymmetric (public-key) cryptography. Today, widely used public-key cryptosystems are based on two hard problems: the Integer Factorization Problem and the Discrete Logarithm Problem. It is infeasible to solve these problems in a reasonable amount of time using classical computers. In 1994, Peter Shor proposed a polynomial-time algorithm that can solve these problems on a quantum computer [40]. Quantum computers are not yet a part of daily life, but it is predicted that they will be in ten to fifteen years. Before that day comes, designing cryptographic algorithms that are secure against both classical and quantum computers is essential. Research on this subject has been pursued for years in five main classes: lattice-based cryptography, code-based cryptography, multivariate polynomial-based cryptography, hash-based cryptography, and isogeny-based cryptography. It is believed that the cryptographic systems belonging to these classes are resistant to both classical and quantum computers.

In 2017 National Institute of Standards and Technology (NIST) started the 'Post-Quantum Cryptography (PQC) Standardization' competition. Among the 69 submitted proposals, 5 of them had withdrawn within a year. Twenty-seven of the remaining 64 were lattice-based constructions (public-key encryptions (PKE), key encapsulation mechanisms (KEM), digital signatures (DS)). For the second round of the standardization process, 26 candidates were announced [2] in January 2019, of which 12 are lattice-based constructions. In July 2020, the third round officially began with 4 PKE/KEM (Classic McEliece [10], Crystals-Kyber [7], NTRU [18], Saber [24]) and 3 DS (Crystals-Dilithium [21], Falcon [26], Rainbow [20]) finalists together with 5 PKE/KEM (BIKE [6], FrodoKEM [14], HQC [37], NTRU Prime [11], SIKE [8]), and 3 DS (GeMMS [16], Picnic [17], SPHINCS$^+$ [12]) alternate candidates. 7 of these 15 schemes in Round 3 were lattice-based. As these numbers indicate, lattice-based cryptography has been one of the most powerful candidates for the post-quantum cryptography standardization process. In the second round status report [1], NIST stated that they would consider at most one of the lattice-based PKE/KEM finalists for standardization since they are all using structured lattices. The third round was expected to be concluded by the end of 2021. NIST updated the date to March 2022. In April 2022, NIST still was not ready to announce the decision. It was reported that the reason for the delay was legal and procedural details and not related to the technical evaluation. Resolving the issues caused by these legal and procedural details took longer than they expected. Finally, in July 2022, the third round is closed with the announcement [3] of one PKE/KEM (Crystals-KYBER) and three DS (Crystals-Dilithium, FALCON, SPHINCS$^+$) schemes being selected for standardization. Among the remaining algorithms, four PKE/KEMs (BIKE, Classic McEliece, HQC, SIKE) advanced to the fourth round for further consideration. For the rest of the algorithms, NIST stated that they will not be subjected to further evaluation and will not be considered for standardization. Nevertheless, some changes may occur later due to the intellectual property issues that Kyber and Saber have been facing, which seem to be the main reason for the delay. Although NIST eliminated the NTRU PKE/KEM, they also stated that they could consider standardizing it if an agreement could not be reached on the intellectual property by the end of 2022 (footnote on page 18 of [3]). Regardless of the outcome of the competition, it seems that further research on lattice-based cryptosystems will continue, at least for a while.

In this thesis, we mainly focus on TMVP-based multiplication algorithms for PQC and the application of some lattice-based cryptosystems. In Chapter 2, notations used in this thesis and some background information on the subject are given. The new three- and four-way formulas for TMVPs with a square Toeplitz matrix with five and seven multiplications are presented in Chapter 3. A detailed explanation of the derivation technique used for the formulas can also be found in Chapter 3. Moreover, this chapter includes the simplified versions of the formulas for TMVPs representing multiplication modulo $x^n + 1$. Chapter 4 contains a new, non-square approach for TMVP calculations. The derivation of unbalanced TMVP formulas (2×3, 3×2, 2×4, 4×2, 3×4, and 4×3) are presented in this chapter. The new six-, eight-, and twelve-way formulas derived via composing the unbalanced formulas are also explained in this chapter.

A TMVP-based multiplication algorithm using the four-way formula and an improved version of it for Saber can be found in Chapter 5. Similarly, in Chapter 6, TMVP-based algorithms for all parameter sets of NTRU are presented with a padding technique. Chapter 7 contains the results of the implementations of the proposed algorithms on ARM Cortex-M4. The benchmark results of the applications of the proposed algorithms to Saber and NTRU are also placed in Chapter 7. Finally, Chapter 8 concludes the thesis.

CHAPTER 2

PRELIMINARY TO THE SUBJECT

This chapter introduces some definitions and properties to build a background. First, we describe the Toeplitz matrix and explain how to express a multiplication in the ring $\mathbb{Z}_q[x]/\langle x^n \pm 1\rangle$ as a Toeplitz matrix-vector product (TMVP). Then we introduce the two- and three-way formulas for TMVP calculations in the literature. Secondly, we explain two of the finalists of the NIST PQC competition, namely Saber and NTRU. Finally, we briefly introduce the ARM Cortex-M4 microprocessor, which we use as the implementation platform in this thesis.

2.1 Notation

Throughout the thesis, we use the notation $M(n)$ to state the arithmetic complexity of an algorithm for dimension n. The ring of polynomials modulo $x^n + 1$ with integer coefficients in $[0, q)$ is denoted by $\mathcal{R}_{q^+} = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$, and the ring of polynomials modulo $x^n - 1$ with integer coefficients in $[0, q)$ is denoted by $\mathcal{R}_{q^-} = \mathbb{Z}_q[x]/\langle x^n - 1\rangle$. The multiplication of an $m \times n$ matrix by a vector of length n is referred to as an $m \times n$-dimensional matrix-vector multiplication if $m \neq n$; otherwise, we refer it to as an n-dimensional matrix-vector multiplication. Single-word and double-word additions are denoted by A_S and A_D, respectively. When $A_D = A_S$, we omit both from arithmetic complexity calculations and use $M'(n)$ instead of $M(n)$. The symbol "$\circ$" is used to denote the component-wise multiplication of two vectors of the same length.

2.2 Toeplitz Matrix Vector Product

There are many cryptographic applications that utilize TMVP in the literature. The use of TMVP in cryptographic computations first appeared in [25] for multiplying elements of binary extension fields. Many proposals were then suggested [4,25,27,28, 39,41]. Recently, in [4] and [41], the use of TMVP for integer modular multiplication is proposed to speed up the residue multiplication modulo the Mersenne prime $2^{521}-1$ and the prime $2^{255}-19$, respectively. TMVP can also be used to calculate the product of two polynomials modulo a polynomial as explained in [43].

Definition 2.1. *Let m and n be two positive integers. A Toeplitz matrix T is an $m \times n$ matrix whose entry in i-th row and j-th column is defined as $T_{i,j} = T_{i-1,j-1}$ for $i = 2, \dots, m$ and $j = 2, \dots n$.*

$$T = \begin{pmatrix} a_0 & a'_1 & a'_2 & \cdots & \cdots & \cdots & a'_{m\text{-}1} & \cdots & \cdots & a'_{n\text{-}1} \\ a_1 & a_0 & a'_1 & a'_2 & & & \vdots & \ddots & \ddots & a'_{n\text{-}2} \\ a_2 & a_1 & a_0 & a'_1 & \ddots & & \vdots & & \ddots & \vdots \\ \vdots & a_2 & a_1 & \ddots & \ddots & \ddots & \vdots & & & a'_{m\text{-}1} \\ \vdots & \ddots & \ddots & \ddots & \ddots & a'_1 & a'_2 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & a_1 & a_0 & a'_1 & \ddots & \ddots & a'_{n\text{-}m\text{-}1} \\ a_{m\text{-}1} & \cdots & \cdots & \cdots & a_2 & a_1 & a_0 & \cdots & \cdots & a'_{n\text{-}m} \end{pmatrix}. \tag{2.1}$$

The matrix T in (2.1) shows the special form of an $m \times n$ Toeplitz matrix. From time to time, for convenience, we may represent a Toeplitz matrix as an array of its elements in the first row and the first column. For example, the matrix in (2.1) might be represented as the array $T = (a'_{n\text{-}1}, \dots, a'_1, a_0, a_1, \dots, a_{m\text{-}1})$. Clearly, specifying only $m+n-1$ of its elements would suffice to identify T. Therefore, adding two Toeplitz matrices requires only $m+n-1$ additions, while adding regular matrices requires $m.n$. Moreover, every submatrix of a Toeplitz matrix is also a Toeplitz matrix. These properties become very useful when it comes to calculating a TMVP efficiently. Instead of using the naive matrix-vector multiplication, the divide and conquer method works very well for TMVP for large dimensions. Suppose we want to compute the product of the Toeplitz matrix T in (2.1) for $m = n$ by a vector B where the transpose of B is $B^T = (b_0, b_1, \dots, b_n)$. We may apply different splitting methods [28] to compute the following TMVP:

$$T \cdot B = \begin{pmatrix} a_0 & a'_1 & \dots & a'_{n\text{-}2} & a'_{n\text{-}1} \\ a_1 & a_0 & \dots & a'_{n\text{-}3} & a'_{n\text{-}2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n\text{-}2} & a_{n\text{-}3} & \dots & a_0 & a'_1 \\ a_{n\text{-}1} & a_{n\text{-}2} & \dots & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n\text{-}2} \\ b_{n\text{-}1} \end{pmatrix}. \tag{2.2}$$

For example, a two-way TMVP formula allows us to compute an n dimensional TMVP via three $n/2$ dimensional TMVPs. For this, we denote the TMVP in (2.2) by

$$T \cdot B = \begin{pmatrix} T_1 & T_0 \\ T_2 & T_1 \end{pmatrix} \begin{pmatrix} B_0 \\ B_1 \end{pmatrix}$$

where the partitions T_0, T_1, T_2 are $\frac{n}{2} \times \frac{n}{2}$ Toeplitz matrices and B_0, B_1 are vectors of length $\frac{n}{2}$. The n-dimensional Toeplitz matrix-vector product $T.B$ can be calculated as follows:

$$\begin{pmatrix} T_1 & T_0 \\ T_2 & T_1 \end{pmatrix} \begin{pmatrix} B_0 \\ B_1 \end{pmatrix} = \begin{pmatrix} P_0 + P_1 \\ P_0 - P_2 \end{pmatrix}, \tag{2.3}$$

where

$$\begin{aligned} P_0 &= T_1(B_0 + B_1), \\ P_1 &= (T_0 - T_1)B_1, \\ P_2 &= (T_1 - T_2)B_0. \end{aligned}$$

We refer to the formula in (2.3) as TMVP-2. Every TMVP split formula comprises three consecutive steps, which are given in Algorithm 1.

Algorithm 1 Steps of using a TMVP split formula

```
1: Evaluation ($\mathbf{E}$)
        1.a: Matrix Evaluation ($\mathbf{E}_{\text{mtrx}}$)
        1.b: Vector Evaluation ($\mathbf{E}_{\text{vctr}}$)
2: Multiplication ($\mathbf{M}$)
3: Recombination ($\mathbf{R}$)
```

The diagram of the steps of Algorithm 1 for a Toeplitz matrix **A** and a vector **B** is given in Figure 2.1.

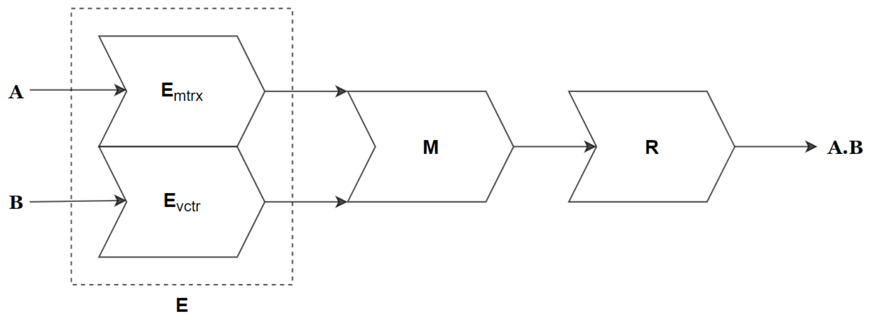


Figure 2.1: Steps of TMVP formula

We explain the operations that corresponds to each step of Algorithm 1 for TMVP-2 formula given in (2.3) in details.

1. ***Evaluation*** (**E**)

   This step evaluates the half-size matrices and vectors for $n/2$-dimensional (n/k-dimensional for TMVP-k formula) TMVPs.

   (a) Matrix evaluation of TMVP-2 formula: $\mathbf{E}_{\text{mtrx}}(T) = (T_1, T_0 - T_1, T_1 - T_2)$,

   (b) Vector evaluation of TMVP-2 formula: $\mathbf{E}_{\text{vctr}}(B) = (B_0 + B_1, B_1, B_0)$.

2. ***Multiplication*** (**M**)

   This step multiplies componentwise the vectors $\mathbf{E}_{\text{mtrx}}(T)$ (a vector of matrices) and $\mathbf{E}_{\text{vctr}}(B)$ (a vector of vectors) from *Evaluation* step, and obtain a vector $P = (P_0, P_1, P_2)$ of vectors. Here each P_i is of length $n/2$ (n/k for TMVP-k formula).

$$\begin{aligned} P =& (P_0, P_1, P_2) \\ =& (T_1, T_0 - T_1, T_1 - T_2) \circ (B_0 + B_1, B_1, B_0) \\ =& (T_1(B_0 + B_1), (T_0 - T_1)B_1, (T_1 - T_2)B_0) \end{aligned}$$

3. ***Recombination*** (**R**)

   This step recombines the output vectors P_0, P_1, and P_2 of *Multiplication* step to obtain the result $R = (R_0, R_1)$ of (2.3) as follows:

$$\begin{aligned} R =& (R_0, R_1) \\ =& (P_0 + P_1, P_0 - P_2). \end{aligned}$$

The number of additions to compute $T_0 - T_1$ is $n-1$, which are all single-word additions. It is sufficient to perform only $n/2$ single-word additions to evaluate $T_1 - T_2$ because it shares $n/2-1$ elements with $T_0 - T_1$ which we do not need to recompute. So, $\mathbf{E}_{\text{mtrx}}$ step requires $3n/2-1$ single-word additions. On the other hand, we need $n/2$ single-word additions to calculate $B_0 + B_1$ for $\mathbf{E}_{\text{vctr}}$ step. Therefore, the evaluation step **E** is completed with $2n-1$ single-word additions. Since the output of $\mathbf{E}_{\text{mtrx}}$ and $\mathbf{E}_{\text{vctr}}$ are vectors of length three, the multiplication step **M** requires three half-size TMVPs. Finally, $n/2$ double-word additions for each $P_0 + P_1$, and $P_0 - P_2$ computations in recombination step **R**. Therefore, the evaluation, multiplication, and recombination steps of TMVP-2 require $2n-1$ single-word additions, 3 half-size TMVPs, and n double-word additions, respectively. Hence, the arithmetic complexity of the TMVP-2 formula is

$$M(n) = 3M(n/2) + (2n-1)A_S + (n)A_D.$$

Since we perform arithmetic modulo 2^{16} for the implementations, there is no need to take care of carry propagation for double-word additions. Therefore, we assume that $A_S = A_D$ for our applications we present in this thesis. So, the arithmetic complexity of TMVP-2 can also be expressed as follows:

$$M'(n) = 3M(n/2) + 3n - 1$$

.

Similarly, a three-way TMVP formula [41] allows us to compute an n-dimensional TMVP via six $n/3$-dimensional TMVPs. For this, the n-dimensional Toeplitz matrix-vector multiplication in (2.2) can be expressed as follows:

$$T \cdot B = \begin{pmatrix} T_2 & T_1 & T_0 \\ T_3 & T_2 & T_1 \\ T_4 & T_3 & T_2 \end{pmatrix} \begin{pmatrix} B_0 \\ B_1 \\ B_2 \end{pmatrix}$$

where the partitions T_0, T_1, T_2, T_3, T_4 are $\frac{n}{3} \times \frac{n}{3}$ Toeplitz matrices and B_0, B_1, B_2 are vectors of length $\frac{n}{3}$. This TMVP can be calculated via the following formula:

$$\begin{pmatrix} T_2 & T_1 & T_0 \\ T_3 & T_2 & T_1 \\ T_4 & T_3 & T_2 \end{pmatrix} \begin{pmatrix} B_0 \\ B_1 \\ B_2 \end{pmatrix} = \begin{pmatrix} P_0 + P_3 + P_4 \\ P_1 - P_3 + P_5 \\ P_2 - P_4 - P_5 \end{pmatrix}, \tag{2.4}$$

where

$$\begin{aligned}
P_0 &= (T_0 + T_1 + T_2)B_2,\\
P_1 &= (T_1 + T_2 + T_3)B_1,\\
P_2 &= (T_2 + T_3 + T_4)B_0,\\
P_3 &= T_1(B_1 - B_2),\\
P_4 &= T_2(B_0 - B_2),\\
P_5 &= T_3(B_0 - B_1).
\end{aligned}$$

In the following sections, we present another three-way formula requiring five multiplications. To avoid confusion, we refer to the formula in (2.4) as the TMVP-3_6, where we use the index 6 to represent the number of multiplications required by the formula. For arithmetic complexity calculation of this formula, we express the $n/3 \times n/3$ Toeplitz matrices T_0, T_1, T_2, T_3, T_4 as the following arrays of length $2n/3 - 1$ as mentioned in Section 2.2.

$$\begin{aligned}
T_0 &= X_0 || a_{n/3-1} || X_1,\\
T_1 &= X_1 || a_{2n/3-1} || X_2,\\
T_2 &= X_2 || a_{3n/3-1} || X_3,\\
T_3 &= X_3 || a_{4n/3-1} || X_4,\\
T_4 &= X_4 || a_{5n/3-1} || X_5
\end{aligned}$$

where $||$ denotes concatenation and

$$\begin{aligned}
X_0 &= (a_0, \ldots, a_{n/3-2}),\\
X_1 &= (a_{n/3}, \ldots, a_{2n/3-2}),\\
X_2 &= (a_{2n/3}, \ldots, a_{3n/3-2}),\\
X_3 &= (a_{3n/3}, \ldots, a_{4n/3-2}),\\
X_4 &= (a_{4n/3}, \ldots, a_{5n/3-2}),\\
X_5 &= (a_{5n/3}, \ldots, a_{6n/3-2}).
\end{aligned}$$

are arrays of length $n/3 - 1$. Therefore, we have

$$\begin{aligned}
T_0 + T_1 + T_2 &= X_0 + X_1 + X_2 || a_{n/3-1} + a_{2n/3-1} + a_{3n/3-1} || X_1 + X_2 + X_3\\
T_1 + T_2 + T_3 &= X_1 + X_2 + X_3 || a_{2n/3-1} + a_{3n/3-1} + a_{4n/3-1} || X_2 + X_3 + X_4\\
T_2 + T_3 + T_4 &= X_2 + X_3 + X_4 || a_{3n/3-1} + a_{4n/3-1} + a_{5n/3-1} || X_3 + X_4 + X_5.
\end{aligned}$$

For these evaluations we use Algorithm 2. Each line of Algorithm 2 represents an operation and contains the number of additions that are required by this operation.

Algorithm 2 Matrix Evaluations

```
1: $S_1 = X_1 + X_2$                          ▷ n/3-1
2: $S_2 = X_0 + S_1$                          ▷ n/3-1
3: $S_3 = S_1 + X_3$                          ▷ n/3-1
4: $S_4 = X_3 + X_4$                          ▷ n/3-1
5: $S_5 = X_2 + S_4$                          ▷ n/3-1
6: $S_6 = S_4 + X_5$                          ▷ n/3-1
7: $S_7 = a_{2n/3-1} + a_{3n/3-1}$            ▷ 1
8: $S_8 = a_{n/3-1} + S_7$                    ▷ 1
9: $S_9 = S_7 + a_{4n/3-1}$                   ▷ 1
10: $S_{10} = a_{3n/3-1} + a_{4n/3-1}$        ▷ 1
11: $S_{11} = S_{10} + a_{5n/3-1}$            ▷ 1
```

We conclude the matrix evaluation step by concatenating the length $n/3 - 1$ vectors S_i as follows:

$$T_0 + T_1 + T_2 = S_2||S_8||S_3$$
$$T_1 + T_2 + T_3 = S_3||S_9||S_5$$
$$T_2 + T_3 + T_4 = S_5||S_{11}||S_6.$$

The number of single-word additions needed to compute S_i ($i = 1, \dots, 11$) via Algorithm 2 is $2n-1$. Since concatenation has no operational cost, $\mathbf{E}_{\mathsf{mtrx}}$ step of TMVP-3_6 completed with $2n-1$ single word additions. Evaluating the vectors B_1-B_2, B_0-B_2, and $B_0 - B_1$ in $\mathbf{E}_{\mathsf{vctr}}$ step each require $n/3$ single-word additions. Therefore, the evaluation step $\mathbf{E}$ of TMVP-3_6 formula requires $3n - 1$ single-word additions. For recombination step, each $P_1 + P_4 + P_5$, $P_2 - P_4 + P_6$, and $P_3 - P_5 - P_6$ vector computations need $2n/3$ double-word additions. Hence, the arithmetic complexity of the TMVP-3_6 formula is

$$M(n) = 6M(n/3) + (3n - 1)A_S + (2n)A_D.$$

When $A_S = A_D$ the arithmetic complexity is represented as follows:

$$M'(n) = 6M(n/3) + 5n - 1.$$

The choice of the formula for efficiently calculating a TMVP differs depending on various details such as the size of the Toeplitz matrix and the implementation platform. In this thesis, we use Winograd's technique in [43] to derive a three-way TMVP formula from the Toom-3 algorithm and a four-way TMVP formula from the Toom-4 algorithm. The three- and four-way formulas we propose require five and seven multiplications, respectively. We explain the derivation technique in detail and present the formulas in Section 3.

2.2.1 Polynomial Multiplication Modulo $x^n \pm 1$ via TMVP

Let $c'(x) = \sum_{i=0}^{2n-2} c'_i x^i \in \mathbb{Z}[x]$ denote the product of the polynomials $a(x) = \sum_{i=0}^{n-1} a_i x^i$ and $b(x) = \sum_{i=0}^{n-1} b_i x^i$ in $\mathbb{Z}[x]$ where $c'_i = \sum_{j+k=i} a_j b_k$. The product $c(x) = \sum_{i=0}^{n-1} c_i x^i$ of the polynomials $a(x)$ and $b(x)$ in $\mathbb{Z}[x]/\langle x^n \pm 1\rangle$ can be calculated by reducing $c'(x)$ modulo $x^n \pm 1$. Clearly, $c_i = c'_i \mp c'_{i+n}$ for $i = 0, \ldots, n-2$ and $c_{n-1} = c'_{n-1}$. The coefficients of the polynomial $c(x)$ can be calculated via the following TMVP:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n\text{-}2} \\ c_{n\text{-}1} \end{pmatrix} = \begin{pmatrix} a_0 & \mp a_{n\text{-}1} & \ldots & \mp a_2 & \mp a_1 \\ a_1 & a_0 & \ldots & \mp a_3 & \mp a_2 \\ a_2 & a_1 & \ldots & \mp a_4 & \mp a_3 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n\text{-}2} & a_{n\text{-}3} & \ldots & a_0 & \mp a_{n\text{-}1} \\ a_{n\text{-}1} & a_{n\text{-}2} & \ldots & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n\text{-}2} \\ b_{n\text{-}1} \end{pmatrix}. \qquad (2.5)$$

The Toeplitz matrix in (2.5) has a more special form than (2.1). It contains only n distinct entries, which are the coefficients of one of the multiplicand polynomials. Having n components instead of $2n-1$ means we can simplify the formulas we use. Therefore, we can calculate these types of TMVPs even more efficiently. More specifically, if we use a k-split method (TMVP-k formula), we would have k different components instead of $2k-1$. Thus, it allows us to reduce the number of additions in the matrix evaluation step.

2.3 Saber

Saber [22, 24] is a lattice-based KEM and one of the finalists of the NIST PQC standardization competition. Its security relies on the conjectural hardness of the Module Learning with Rounding (MLWR) problem [9, 35]. Saber defines a public-key encryption scheme (Saber.PKE) which is indistinguishable under chosen plaintext attack (IND-CPA). Saber.PKE consists key generation (Saber.PKE.Keygen), encryption (Saber.PKE.Enc), decryption (Saber.PKE.Dec) algorithms as described in Algorithm 3, Algorithm 4, and Algorithm 5, respectively. It uses a version of the Fujisaki-Okamoto (FO) transformation to have a key encapsulation mechanism (Saber.KEM) which is indistinguishable under chosen ciphertext attack (IND-CCA). Saber.KEM consists of key generation (Saber.KEM.KeyGen), encapsulation (Saber.KEM.Encaps), decapsulation (Saber.KEM.Decaps) algorithms. The details of all algorithms can be found in [24].

Algorithm 3 Saber.PKE.KeyGen()

```
1: $seed_{\boldsymbol{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$
2: $\boldsymbol{A} = \mathsf{Gen}(seed_{\boldsymbol{A}}) \in \mathcal{R}_{q^+}^{\ell\times\ell}$
3: $\boldsymbol{s} = \beta(\mathcal{R}_{q^+}^{\ell\times 1})$
4: $\boldsymbol{b} = ((\boldsymbol{A}^T\boldsymbol{s} + \boldsymbol{h}) \mod q) \gg (\epsilon_q - \epsilon_p) \in \mathcal{R}_{p^+}^{\ell\times 1}$
5: return $(pk = (seed_{\boldsymbol{A}}, \boldsymbol{b}), sk = (\boldsymbol{s}))$
```

Algorithm 4 Saber.PKE.Enc($pk = (seed_{\boldsymbol{A}}, \boldsymbol{b}), m \in \mathcal{R}_2$)

```
1: $seed_{\boldsymbol{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$
2: $\boldsymbol{s'} = \beta(\mathcal{R}_{q^+}^{\ell\times 1})$
3: $\boldsymbol{b'} = ((\boldsymbol{A}\boldsymbol{s'} + \boldsymbol{h}) \mod q) \gg (\epsilon_q - \epsilon_p) \in \mathcal{R}_{p^+}^{\ell\times 1}$
4: $v' = \boldsymbol{b}^T(\boldsymbol{s'} \mod p) \in \mathcal{R}_{p^+}$
5: $c_m = (v' + h_1 - 2^{\epsilon_p - 1}m \mod p) \gg (\epsilon_p - \epsilon_T) \in \mathcal{R}_{T^+}$
6: return $c = (c_m, \boldsymbol{b'})$
```

Algorithm 5 Saber.PKE.Dec($sk = \boldsymbol{s}, c = (c_m, \boldsymbol{b'})$)

```
1: $v = \boldsymbol{b'}^T(\boldsymbol{s} \mod p) \in \mathcal{R}_{p^+}$
2: $m' = ((v - 2^{\epsilon_p - \epsilon_T} c_m + h_2) \mod p) \gg (\epsilon_p - 1) \in \mathcal{R}_{2^+}$
3: return $m'$
```

In the algorithms above, bold uppercase letters (e.g., $\boldsymbol{A}$) are used to denote matrices, and bold lowercase letters (e.g. $\boldsymbol{b}$) are used to denote vectors, all of which have entries from $\mathcal{R}_{q^+}$ and $\mathcal{R}_{p^+}$. Sampling from the uniform and binomial distributions on a set S is denoted by $\mathcal{U}(S)$, and $\beta(S)$, respectively. Shifting each coefficient of a polynomial $poly$ right by t bits is denoted by $poly \gg t$.

The scheme specifies three different values for the dimension ℓ of the module that determines the security level of the scheme. The values $\ell = 2$ (LightSaber), $\ell = 3$ (Saber), $\ell = 4$ (FireSaber) provide level 1, level 3, level 5 security, respectively. Saber operates on the finite polynomial rings $\mathcal{R}_{q^+} = \mathcal{R}_{2^{13}+} = \mathbb{Z}_{2^{13}+}[x]/\langle x^{256} + 1\rangle$ and $\mathcal{R}_{p^+} = \mathcal{R}_{2^{10}+} = \mathbb{Z}_{2^{10}+}[x]/\langle x^{256} + 1\rangle$. Like most of the lattice-based cryptosystems defined on polynomial rings, multiplication in these rings directly affects the efficiency of the scheme.

2.4 NTRU

NTRU is one of the lattice-based finalists of the NIST PQC competition. NTRU KEM is a merger of NTRUEncrypt and NTRU-HRSS-KEM submissions of the first round, and it is based on the classical NTRU system proposed by Hoffstein, Pipher, and Silverman [29,30]. Unlike the original NTRU, this system utilizes a perfectly correct deterministic public key encryption (DPKE) instead of a partially correct probabilistic one. The KEM is obtained by applying a variant of FO transformation to this DPKE. The key generation, encryption, and decryption algorithms of NTRU CCA-DPKE are given in Algorithm 6, Algorithm 7, and Algorithm 8.

Algorithm 6 NTRU DPKE KeyGen($seed$)

```
1: (f,g) ← Sample($seed$)
2: $\mathbf{f}_q \leftarrow (1/\mathbf{f}) \mod (q, \mathbf{\Phi}_n)$
3: $\mathbf{h} \leftarrow (3 \cdot \mathbf{g} \cdot \mathbf{f}_q) \mod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
4: $\mathbf{h}_q \leftarrow (1/\mathbf{h}) \mod (q, \mathbf{\Phi}_n)$
5: $\mathbf{f}_p \leftarrow (1/\mathbf{f}) \mod (3, \mathbf{\Phi}_n)$
6: return $((\mathbf{f}, \mathbf{f}_p, \mathbf{f}_q)$, h)
```

Algorithm 7 NTRU DPKE Encrypt(**h,(r,m)**)

```
1: $\mathbf{m}' \leftarrow \mathsf{Lift}(\mathbf{m})$
2: $\mathbf{c} \leftarrow (\mathbf{r}\cdot\mathbf{h}+\mathbf{m}') \mod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
3: return c
```

Algorithm 8 NTRU DPKE Decrypt(**(f, f$_p$,h$_q$),c)**

```
1: if $\mathbf{c} \not\equiv 0( \mod (q, \mathbf{\Phi}_1))$ return $(0, 0, 1)$
2: $\mathbf{a} \leftarrow (\mathbf{c}\cdot\mathbf{f}) \mod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
3: $\mathbf{m} \leftarrow (\mathbf{a}\cdot\mathbf{f}_p) \mod (3, \mathbf{\Phi}_n)$
4: $\mathbf{m}' \leftarrow \mathsf{Lift}(\mathbf{m})$
5: $\mathbf{r} \leftarrow ((\mathbf{c}\text{-}\mathbf{m})'\cdot\mathbf{h}_q) \mod (q, \mathbf{\Phi}_n)$
6: if r.m $\in \mathcal{L}_r \times \mathcal{L}_m$ return (r, m, $0$)
7: else return $(0, 0, 1)$
```

Here n is an odd prime number and $\mathbf{\Phi}_i$ is the the i-th cyclotomic polynomial. Therefore, we have $\mathbf{\Phi}_1 = x - 1$, $\mathbf{\Phi}_n = x^{n-1} + x^{n-2} + \cdots + x + 1$, and $\mathbf{\Phi}_1\mathbf{\Phi}_n = x^n - 1$. By $\mod (q, \mathbf{\Phi})$ we refer to the reduction both modulo q and the polynomial Φ. All bold lowercase letters in the algorithms above denote polynomials. In Table 2.1, the recommended values of n and q parameters, and the security category of the schemes that corresponds to these parameters are given.

Table 2.1: Recommended Parameters and Security Levels for NTRU

	ntruhps2048509	ntruhps2048677	ntruhps4096821	ntruhrss701
n	509	677	821	701
q	2048	2048	4096	8192
Sec. Levels	1	3	3	5

All ring multiplications in the scheme are done using $\mathcal{R}_{q^-} = \mathbb{Z}_q[x]/\langle \mathbf{\Phi}_1 \mathbf{\Phi}_n \rangle = \mathbb{Z}_q[x]/\langle x^n - 1 \rangle$ multiplications via performing a relevant reduction to the result. So, multiplication in $\mathcal{R}_{q^-}$ is the main operation that affects the efficiency of all multiplication operations in the scheme. Improvements in ring multiplication had very little impact on the key generation until the inversion using Bernstein-Yang's constant time greatest common divisor algorithm [13] was proposed in [36]. With this work, the dominance of inversion on the performance of key generation decreased dramatically, and the key generation algorithm of NTRU KEM is improved 97%.

The rings $\mathcal{R}_{q^+}$ and $\mathcal{R}_{q^-}$ that Saber and NTRU are defined on are not suitable for directly using the Number Theoretic Transform (NTT), which is the most efficient polynomial multiplication algorithm known. NTT requires some restrictions on parameters. For example, for a Radix-2 NTT we need n to be a power of two and q to be a prime that satisfies $q \equiv 1 \mod 2n$. Since neither complies with these conditions, regardless of the platform, in most of the implementations of Saber and NTRU, a combination of Toom-Cook, Karatsuba, and schoolbook methods were used for efficient polynomial multiplication together with a polynomial reduction. Details of existing implementations that use Toom-Cook and Karatsuba algorithms on different platforms can be found in [31, 34, 38]. The work proposed in [19] shows that NTT can be used for the schemes defined on 'NTT-unfriendly' rings. The method in this work uses a big enough 'NTT-friendly' polynomial ring and performs operations on this ring to avoid any modular reductions which may cause errors in the final result. Once it computed the result in the 'NTT-friendly' ring, it performs modular reduction to this result to obtain the actual result in the target 'NTT-unfriendly' ring. In this thesis, we present the first-time use of TMVP-based algorithms for multiplication in polynomial quotient rings for post-quantum cryptographic schemes.

2.5 Implementation Platform: ARM Cortex-M4

We chose the ARM Cortex-M4 microcontroller as the implementation platform. The Cortex-M4 implements the ARMv7E-M instruction set, and NIST recommends it as a reference implementation platform for the evaluation of PQC candidates on microcontrollers. It has sixteen 32-bit general-purpose registers, and aside from Program

Counter (PC) and Stack Pointer (SP) registers, they are all available for development. It also has thirty-two 32-bit floating point registers, which can be used to keep frequently used values to decrease the use of load and store instructions. We use the STM32F4DISCOVERY development board, which is used in many implementations of PQC candidates [5, 15, 31, 32, 34]. The ARM Cortex-M4 is designed especially for digital signal processing (DSP), and it supports many useful single instruction multiple data (SIMD) instructions that can perform parallel arithmetic operations on two 16-bit halfword parts of two 32-bit registers in one cycle. These instructions allow us to efficiently implement the schoolbook matrix-vector multiplication for small dimensions. In Table 2.2, descriptions of some instructions we use in our implementation are given.

Table 2.2: Example instructions

General data processing		
ADD Rd, Rn, Rm	Rd $=$ Rn $+$ Rm	
USUB16 Rd, Rn, Rm	Rd$_b$ $=$ (Rn$_b$ $-$ Rm$_b$)$\bmod 2^{16}$	Rd$_t$ $=$ (Rn$_t$ $-$ Rm$_t$)$\bmod 2^{16}$
Multiply-Accumulate		
SMUADX Rd, Rn, Rm	Rd $=$ Rn$_b$Rm$_t$ $+$ Rn$_t$Rm$_b$	
SMLADX Rd, Rn, Rm, Rt	Rd $=$ Rn$_b$Rm$_t$ $+$ Rn$_t$Rm$_b$ $+$ Rt	
Packing-Unpacking		
PKHBT Rd, Rn, Rm LSL # k	Rd$_b$ $=$ Rn$_b$	Rd$_t$ $=$ (Rm $\ll k)_t$
PKHTB Rd, Rn, Rm ASR # k	Rd$_b$ $=$ (Rm $\gg k)_b$	Rd$_t$ $=$ Rn$_t$

The indices b and t denote the bottom (bits $0-15$) and top (bits $16-31$) halfwords of the relevant register, respectively. The symbols $\ll$ and $\gg$ denote the left and right shifts, respectively.

2.6 The Block Recombination Method

The block recombination method was proposed in [27] for parallel multiplication in binary finite fields that are expressed as a TMVP. The two-way split formula for an n dimensional TMVP defined on a binary field is as follows:

$$U.V = \begin{bmatrix} U_1 & U_0 \\ U_2 & U_1 \end{bmatrix} \begin{bmatrix} V_0 \\ V_1 \end{bmatrix} = \begin{bmatrix} W_0 + W_1 \\ W_1 + W_2 \end{bmatrix} = \begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix}, \tag{2.6}$$

where

$$W_0 = U_1(V_0 + V_1),$$
$$W_1 = (U_0 + U_1)V_1,$$
$$W_2 = (U_1 + U_2)V_0.$$

Here U_i are $n/2 \times n/2$ dimensional Toeplitz matrices, while V_j and W_i are vectors of length $n/2$ with binary components. In [27], they decompose the multiplier proposed by Fan-Hasan [25] into four steps defined as follows:

1. **Component Matrix Formation (*CMF*):** Corresponds to the recursive matrix evaluation of U in (2.6). The first step of recursion is equivalent to $\mathbf{E}_{\mathsf{mtrx}}$ from Section 2.2, and outputs the vector $(U_1, U_0 + U_1, U_1 + U_2)$ of binary matrices. So, we say $\textit{CMF}(U) = (\textit{CMF}(U_1), \textit{CMF}(U_0 + U_1), \textit{CMF}(U_1 + U_2))$ is the component matrix formation of U.
2. **Component Vector Formation (*CVF*):** Corresponds to the recursive vector evaluation of V in (2.6). The first step of recursion is equivalent to $\mathbf{E}_{\mathsf{vctr}}$ from Section 2.2, and outputs the $(V_0 + V_1, V_1, V_0)$ of binary vectors. So, we say $\textit{CVF}(V) = (\textit{CVF}(V_0 + V_1), \textit{CVF}(V_1), \textit{CVF}(V_0))$ is the component vector formation of V.
3. **Component Multiplication (*CM*):** This step is similar to the step **M** from Section 2.2. *CM* multiplies the binary vectors of $\textit{CMF}(U)$ and $\textit{CVF}(V)$ component-wise, and outputs a binary vector of same length as $\textit{CMF}(U)$ and $\textit{CVF}(V)$.
4. **Reconstruction (*R*):** This step is similar to **R** from Section 2.2. Corresponds to the recombination of the product $Y = U.V$ from the output of the previous step. Let $\hat{W} = \textit{CMF}(U) \circ \textit{CVF}(V) = [\hat{W}_0, \hat{W}_1, \hat{W}_2]$. Therefore, we have $W = R(\hat{W}) = (R(\hat{W}_0) + R(\hat{W}_1), R(\hat{W}_0) + R(\hat{W}_2))$.

In [27], the sum of two or more TMVPs can be calculated more efficiently using this decomposition. Applying a layer of schoolbook method to (2.6) before decomposing the blocks as shown in (2.7) reduces the complexity.

$$U.V = \begin{bmatrix} U_1 & U_0 \\ U_2 & U_1 \end{bmatrix} \begin{bmatrix} V_0 \\ V_1 \end{bmatrix} = \begin{bmatrix} U_1V_0 + U_0V_1 \\ U_2V_0 + U_1V_1 \end{bmatrix} \tag{2.7}$$

To compute U_1V_0 and U_0V_1 using block decomposition, we perform Steps 1-4 above for both. Then we calculate $U_1V_0 + U_0V_1$ to obtain the first half of the result of (2.7). Therefore, 2 *CMF*s, 2 *CVF*s, 2 *CM*s, 2 *R*s, and $n/2$ additions (double-word) are required for this computation. The block recombination method proposes to perform additions before the recombination step. More precisely, after calculating *CMF*, *CVF*, and *CM* steps of U_1V_0 and U_0V_1, the component addition step (*CA*) is performed before recombination. As the name indicates, *CA* is adding the resulting vectors component by component. Therefore, $U_1V_0 + U_0V_1$ is calculated with 2 *CMF*s, 2 *CVF*s, 2 *CM*s, 1 *CA*, 1 R. A similar calculation is valid for $U_2V_0 + U_1V_1$, except for one *CMF* and 2 *CVF*s. Since we already have *CMF*(U_1), *CVF*(V_0) and *CVF*(V_1), there is no need to repeat these calculations. So, the result of (2.7) is calculated with 3 *CMF*s, 2 *CVF*s, 4 *CM*s, 2 *CA*s, and 2 *R*s via the block recombination method, instead of 3 *CMF*s, 2 *CVF*s, 4 *CM*s, 4 *R*s, and 2 *CA*s. Hence, the block recombination method improves the arithmetic complexity by reducing the number of reconstruction steps from four to two.

We use a modified version of the block recombination method to develop a TMVP-based multiplication algorithm that is unique to Saber. In our version, we work on $\mathbb{Z}_q$ instead of binary fields, and we use various TMVP formulas consecutively until reaching a precomputed dimension, while the block recombination method uses the same TMVP formula recursively until the dimension is one.

CHAPTER 3

NEW THREE- AND FOUR-WAY TMVP FORMULAS

TMVP split formulas can be derived from any given polynomial multiplication algorithm using Winograd's technique given in [43]. We derive the new three- and four-way TMVP formulas from Toom-3 and Toom-4 algorithms [42] using a similar technique in [43]. The new algorithms are denoted by TMVP-3 and TMVP-4, and they require five and seven smaller TMVPs, respectively. As we mentioned previously, we may use the notation TMVP-3_5 instead of TMVP-3 to emphasize the number of multiplications required by the formula, if needed. This section explains the derivation of our new three- and four-way TMVP formulas elaborately. Moreover, the simplified versions of the formulas for TMVPs that represent multiplication modulo $x^n + 1$ are given. The arithmetic complexity calculations of the algorithms and their comparisons are also presented in this section.

3.1 Three-way TMVP formula with five multiplications

Let $c(x) = c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4$ be the product of two three-term polynomials $a(x) = a_0 + a_1x + a_2x^2$ and $b(x) = b_0 + b_1x + b_2x^2$. The coefficients c_i of the product polynomial $c(x)$ can be calculated using different methods including the schoolbook and three-way Toom-Cook (Toom-3) algorithms.

- **Schoolbook:** Using the schoolbook polynomial multiplication, the coefficients of the polynomial $c(x)$ are computed as follows:

$$\begin{aligned}
c_0 &= a_0b_0,\\
c_1 &= a_0b_1 + a_1b_0,\\
c_2 &= a_0b_2 + a_1b_1 + a_2b_0,\\
c_3 &= a_1b_2 + a_2b_1,\\
c_4 &= a_2b_2.
\end{aligned} \tag{3.1}$$

- **Toom-3**: The Toom-3 algorithm evaluates 3-term multiplicand polynomials at 5 chosen points. Then, it obtains the evaluation of the product polynomial at these same five points by multiplying the results point-wise. Finally, the product polynomial is interpolated from these evaluations using an interpolation algorithm.

We choose $S = \{0, 1, -1, -2, \infty\}$ as the set of evaluation points and evaluate the polynomials $a(x)$ and $b(x)$ at these points, where evaluation at ∞ equals the leading coefficient of the corresponding polynomial. For each $s \in S$ the equality $c(s) = a(s)b(s)$ holds. Then we have the following:

$$\begin{aligned}
c(0) &= a_0b_0,\\
c(1) &= (a_0 + a_1 + a_2)(b_0 + b_1 + b_2),\\
c(-1) &= (a_0 - a_1 + a_2)(b_0 - b_1 + b_2),\\
c(-2) &= (a_0 - 2a_1 + 4a_2)(b_0 - 2b_1 + 4b_2),\\
c(\infty) &= a_2b_2.
\end{aligned} \tag{3.2}$$

We interpolate the coefficients c_i of $c(x)$ from $c(s)$ values ($s \in S$) using Algorithm 9.

Algorithm 9 Interpolation of Toom3

```
1: $S_1 = c(0)$                          ▷ $c_0 = S_1$
2: $S_2 = c(\infty)$                     ▷ $c_4 = S_2$
3: $S_3 = \frac{c(-2)-c(1)}{3}$
4: $S_4 = \frac{c(1)-c(-1)}{2}$
5: $S_5 = c(-1) - c(0)$
6: $S_6 = \frac{S_5-S_3}{2} + 2S_2$      ▷ $c_3 = S_6$
7: $S_7 = S_5 + S_4 - S_2$               ▷ $c_2 = S_7$
8: $S_8 = S_4 - S_6$                     ▷ $c_1 = S_8$
```

Therefore, the coefficient c_i of $c(x)$ can be written in terms of $c(s)$ for $s \in S$ as follows:

$$\begin{aligned} c_0 &= c(0), \\ c_1 &= \frac{1}{6}\left(3c(0) + 2c(1) - 6c(-1) + c(-2) - 12c(\infty)\right), \\ c_2 &= \frac{1}{2}\left(-2c(0) + c(1) + c(-1) - 2c(\infty)\right), \\ c_3 &= \frac{1}{6}\left(-3c(0) + c(1) + 3c(-1) - c(-2) + 12c(\infty),\right) \\ c_4 &= c(\infty). \end{aligned} \tag{3.3}$$

Now we use the two different calculations (3.1) and (3.3) of coefficients, to derive the TMVP-3 formula. For this, first, we multiply each equation in (3.1) corresponds to c_i by a symbolic variable z_{4-i} for $i = 0, \ldots, 4$ and then we take the sum of all equations as seen in (3.4).

$$\begin{aligned} z_4c_0 + z_3c_1 + z_2c_2 + z_1c_3 + z_0c_4 = z_4a_0b_0& \\ + z_3(a_0b_1 + a_1b_0)& \\ + z_2(a_0b_2 + a_1b_1 + a_2b_0)& \\ + z_1(a_1b_2 + a_2b_1)& \\ + z_0(a_2b_2).& \end{aligned} \tag{3.4}$$

Then, we rearrange the right hand side of (3.4) in the form $k_2b_2 + k_1b_1 + k_0b_0$, where the coefficients k_i of b_i are as follows:

$$\begin{pmatrix} k_2 \\ k_1 \\ k_0 \end{pmatrix} = \begin{pmatrix} z_2 & z_1 & z_0 \\ z_3 & z_2 & z_1 \\ z_4 & z_3 & z_2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}. \tag{3.5}$$

Similarly, we also multiply each equation in (3.3) corresponds to c_i by a symbolic variable z_{4-i} for $i = 0, \ldots, 4$ and we take the sum of all equations as given in (3.6)

$$\begin{aligned} & z_4c_0 + z_3c_1 + z_2c_2 + z_1c_3 + z_0c_4 \\ =& \frac{1}{6}\big[z_4 6c(0) \\ &+ z_3\left(3c(0) + 2c(1) - 6c(-1) + c(-2) - 12c(\infty)\right) \\ &+ z_2\left(-6c(0) + 3c(1) + 3c(-1) - 6c(\infty)\right) \\ &+ z_1\left(-3c(0) + c(1) + 3c(-1) - c(-2) + 12c(\infty)\right) \\ &+ z_0\left(6c(\infty)\right)\big]. \end{aligned} \tag{3.6}$$

Rearranging the terms in (3.6) in the form $k_2b_2 + k_1b_1 + k_0b_0$ according to (3.2), gives the coefficients k_i of b_i as seen in (3.7).

$$\begin{pmatrix} k_2 \\ k_1 \\ k_0 \end{pmatrix} = \begin{pmatrix} P_1 + P_2 + 4P_3 + P_4 \\ P_1 - P_2 - 2P_3 \\ P_0 + P_1 + P_2 + P_3 \end{pmatrix}, \tag{3.7}$$

where

$$\begin{aligned} P_0 &= \frac{1}{2}\left(2z_4 + z_3 - 2z_2 - z_1\right)a_0, \\ P_1 &= \frac{1}{6}\left(2z_3 + 3z_2 + z_1\right)\left(a_0 + a_1 + a_2\right), \\ P_2 &= \frac{1}{2}\left(-2z_3 + z_2 + z_1\right)\left(a_0 - a_1 + a_2\right), \\ P_3 &= \frac{1}{6}\left(z_3 - z_1\right)\left(a_0 - 2a_1 + 4a_2\right), \\ P_4 &= \left(-2z_3 - z_2 + 2z_1 + z_0\right)a_2. \end{aligned} \tag{3.8}$$

Thus, using the equality of the left-hand sides of (3.5) and (3.7), we have the following formula, which we refer to as the TMVP-3 formula with five multiplications:

$$\begin{pmatrix} z_2 & z_1 & z_0 \\ z_3 & z_2 & z_1 \\ z_4 & z_3 & z_2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} P_1 + P_2 + 4P_3 + P_4 \\ P_1 - P_2 - 2P_3 \\ P_0 + P_1 + P_2 + P_3 \end{pmatrix}, \tag{3.9}$$

where P_i are as given in (3.8). Therefore, with this formulation, we define a TMVP calculation method with 5 smaller TMVPs whose sizes are $1/3$ of the original.

- **TMVP-3 formula for multiplication modulo $x^n + 1$:**

Now, we simplify the TMVP-3 formula for TMVPs representing polynomial multiplication modulo $x^n + 1$. Here, we assume that n is a multiple of three. As mentioned in Section 2.2.1, the coefficients of the product $c(x) = c_0 + c_1x + c_2x^2$ of the polynomials $a(x) = a_0 + a_1x + a_2x^2$ and $b(x) = b_0 + b_1x + b_2x^2$ modulo $x^n + 1$ can be calculated via the following TMVP:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} a_0 & -a_2 & -a_1 \\ a_1 & a_0 & -a_2 \\ a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}. \tag{3.10}$$

The Toeplitz matrices in (3.9) and (3.10) are identified by five and three distinct elements, respectively. So, the matrix evaluations of TMVP-3 formula in (3.9) become simpler for (3.10). Updating the P_i computations for $i = 0, \ldots, 4$ in TMVP-3 formula accordingly, we have the following formula, which we refer to as TMVP-3 formula for $x^n + 1$ and denote as TMVP*-3:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} P_1 + P_2 + 4P_3 + P_4 \\ P_1 - P_2 - 2P_3 \\ P_0 + P_1 + P_2 + P_3 \end{pmatrix} = \begin{pmatrix} a_0 & -a_2 & -a_1 \\ a_1 & a_0 & -a_2 \\ a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix},$$

where

$$\begin{aligned}
P_0 &= \frac{1}{2}\left(a_2 + a_1 - 2a_0\right) b_0, \\
P_1 &= \frac{1}{6}\left(-a_2 + 2a_1 + 3a_0\right)\left(b_0 + b_1 + b_2\right), \\
P_2 &= \frac{1}{2}\left(-a_2 - 2a_1 + a_0\right)\left(b_0 - b_1 + b_2\right), \\
P_3 &= \frac{1}{6}\left(a_2 + a_1\right)\left(b_0 - 2b_1 + 4b_2\right), \\
P_4 &= \left(-2a_2 - 3a_1 - a_0\right) b_2.
\end{aligned}$$

3.1.1 Arithmetic Complexity Calculations

In this section, we calculate the arithmetic complexity of the Toom-3, TMVP-3, and TMVP*-3 algorithms step by step. We should note that in the following arithmetic complexity calculations, we omit the multiplication/division-by-scalar operations.

- **Arithmetic Complexity of Toom-3:** We count the additions and multiplications required by each step of the Toom-3 method separately. We sum up the costs from each step to obtain the total arithmetic complexity of the multiplication of two n-term polynomials via Toom-3.

  1. **Evaluation:** The evaluation of the polynomial $a(x)$ at the points in the set $S = \{0, 1, -1, -2, \infty\}$ is given as follows:

  $$\begin{aligned}
  a(0) &= a_0, \\
  a(1) &= a_0 + a_1 + a_2, \\
  a(-1) &= a_0 - a_1 + a_2, \\
  a(-2) &= a_0 - 2a_1 + 4a_2, \\
  a(\infty) &= a_2.
  \end{aligned}$$

  So, it requires 5 additions of $n/3$-term polynomials with single word coefficients. The same is valid for the evaluation of the polynomial $b(x)$. Hence, this step requires $10n/3$ single-word additions (A_S).

2. **Pointwise multiplication:** In this step we multiply $n/3$-term polynomials $a(s)$ and $b(s)$, and obtain the evaluation of the $(2n/3-1)$-term polynomial $c(s)$ with double-word coefficients, for each $s \in S$. So, this step requires five multiplications of $n/3$-term polynomials which we denote as $5M(n/3)$.

3. **Interpolation:** We use Algorithm 9 to calculate the coefficients of $c(x)$. So, interpolation requires 8 additions of $(2n/3-1)$-term polynomials with double-word coefficients (A_D).

4. **Recomposition:** Since the coefficients c_i of $c(x)$ are $(2n/3-1)$-term polynomials for $i = 0, \dots, 4$, there are four overlaps of length $n/3-1$ which occur in this step. So, we require $4(n/3-1)$ double-word additions (A_D) for this step.

Hence, the total arithmetic complexity of n-term polynomial multiplication via Toom-3 can be stated as

$$M(n) = 5M(n/3) + (20n/3 - 12)A_D + (10n/3)A_S$$

or

$$M'(n) = 5M(n/3) + 10n - 12.$$

- **Arithmetic complexity of TMVP-3:** First, we compute the number of additions and multiplications needed to perform each step of TMVP-3 formula. Then, the sum of them is calculated as the total arithmetic complexity.

1. **Evaluation:** The complexity of the evaluation step is computed in two parts using Algorithm 10 and Algorithm 11.

Algorithm 10 Matrix Evaluations of TMVP-3

```
1: $S_1 = z_3 - z_1$
2: $S_2 = z_1 + z_2$
3: $S_3 = z_4 - z_2$
4: $S_4 = S_3 + S_1/2$
5: $S_5 = S_1/3 + S_2/2$
6: $S_6 = S_2/2 - z_3$
7: $S_7 = -2S_1 - z_2$
8: $S_8 = S_7 + z_0$
9: $S_9 = S_1/6$
```

Algorithm 11 Vector Evaluations of TMVP-3

```
1: $S_1 = a_0 + a_2$
2: $S_2 = S_1 + a_1$
3: $S_3 = S_1 - a_1$
4: $S_4 = a_0 - 2a_1$
5: $S_5 = S_4 + 4a_2$
```

- **Matrix Evaluation:** Since z_i are $n/3 \times n/3$-dimensional Toeplitz matrices, steps $1-8$ of Algorithm 10 require $2n/3-1$ single-word additions each. Hence, matrix evaluation is completed with $(16n/3 - 8)A_S$ operations.
- **Vector Evaluation:** The vectors a_i are of length $n/3$. So, $n/3$ single-word additions are performed in every step of Algorithm 11. Therefore, $(5n/3)A_S$ is the total number of operations for vector evaluations.

2. **Multiplication:** In this step, five $n/3$-dimensional TMVPs are required to calculate P_i for $i = 0, \ldots, 4$. So, we have $5M(n/3)$ for the multiplication step.
3. **Recombination:** Since this step requires seven vector additions and P_i are vectors of length $n/3$ with double-word coefficients, the total cost of recombination is $(7n/3)A_D$.

Thus, adding all these costs gives the total arithmetic complexity of TMVP-3 formula as

$$M(n) = 5M(n/3) + (7n-8)A_S + (7n/3)A_D.$$

When $A_S = A_D = A$ we can express the arithmetic complexity as follows:

$$M'(n) = 5M(n/3) + 28n/3 - 8.$$

- **Arithmetic complexity of TMVP*-3:** The matrix evaluation step is the only difference between the computations of TMVP-3 and TMVP*-3 formulas. So, calculating the cost of the matrix evaluation step of TMVP*-3 would suffice to obtain the total arithmetic complexity of TMVP*-3. The matrix evaluations of TMVP*-3 can be done by seven Toeplitz matrix additions of dimension $n/3 \times n/3$ with single-word coefficients, i.e., $(14n/3 - 7)A_S$ operations are needed. Therefore, the arithmetic complexity of TMVP*-3 formula can be expressed as one of the following:

$$M(n) = 5M(n/3) + (19n/3 - 7)A_S + (7n/3)A_D.$$

$$M'(n) = 5M(n/3) + 26n/3 - 7.$$

3.2 Four-way TMVP formula with seven multiplications

Let $c(x) = c_0+c_1x+c_2x^2+c_3x^3+c_4x^4+c_5x^5+c_6x^6$ be the product of the polynomials $a(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ and $b(x) = b_0 + b_1x + b_2x^2 + b_3x^3$. The coefficients c_i of the product $c(x)$ can be calculated using different methods. In this section, we remind the schoolbook and Toom-4 algorithms.

- **Schoolbook:** The coefficients c_i are computed via the schoolbook method as follows:

$$\begin{aligned}
c_0 &= a_0b_0 \\
c_1 &= a_0b_1 + a_1b_0 \\
c_2 &= a_0b_2 + a_1b_1 + a_2b_0 \\
c_3 &= a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0 \\
c_4 &= a_1b_3 + a_2b_2 + a_3b_1 \\
c_5 &= a_2b_3 + a_3b_2 \\
c_6 &= a_3b_3
\end{aligned} \tag{3.11}$$

- **Toom-4:** To compute the coefficients c_i via Toom-4; first, we need to evaluate the multiplicand polynomials at seven different points. We choose the set $S =$

$\{0, 1, -1, 2, -2, 3, \infty\}$ for evaluation, where evaluation at ∞ equals the leading coefficient of the polynomial. Component-wise multiplication of evaluations of multiplicands gives us the evaluation of the product $c(x)$ at the points $s \in S$. Then, we interpolate the polynomial $c(x)$ from these values.

$$\begin{aligned} c(0) &= a_0 b_0 \\ c(1) &= (a_0 + a_1 + a_2 + a_3)(b_0 + b_1 + b_2 + b_3) \\ c(-1) &= (a_0 - a_1 + a_2 - a_3)(b_0 - b_1 + b_2 - b_3) \\ c(2) &= (a_0 + 2a_1 + 4a_2 + 8a_3)(b_0 + 2b_1 + 4b_2 + 8b_3) \\ c(-2) &= (a_0 - 2a_1 + 4a_2 - 8a_3)(b_0 - 2b_1 + 4b_2 - 8b_3) \\ c(3) &= (a_0 + 3a_1 + 9a_2 + 27a_3)(b_0 + 3b_1 + 9b_2 + 27b_3) \\ c(\infty) &= a_3 b_3 \end{aligned} \tag{3.12}$$

In order to compute the coefficients c_i of $c(x)$ using the $c(s)$ values for $s \in S$, we use the interpolation given in Algorithm 12.

Algorithm 12 Interpolation of Toom-4

```
1: $S_1 = c(0)$    $\triangleright\ c_0 = S_1$
2: $S_2 = c(\infty)$    $\triangleright\ c_6 = S_2$
3: $S_3 = \frac{c(1)+c(-1)}{2} - S_1 - S_2$
4: $S_4 = \frac{c(2)+c(-2)-2S_1-128S_2}{8}$
5: $S_5 = \frac{S_4-S_3}{3}$    $\triangleright\ c_4 = S_5$
6: $S_6 = S_3 - S_5$    $\triangleright\ c_2 = S_6$
7: $S_7 = \frac{c(1)-c(-1)}{2}$
8: $S_8 = \frac{\frac{c(2)-c(-2)}{4}-S_7}{3}$
9: $S_9 = \frac{c(3)-S_1-9S_6-81S_5-729S_2}{3}$
10: $S_{10} = \frac{S_9-S_7}{8} - S_8$
11: $S_{11} = S_{10}/5$    $\triangleright\ c_5 = S_{11}$
12: $S_{12} = S_8 - S_9$    $\triangleright\ c_3 = S_{12}$
13: $S_{13} = S_7 - S_{12} - S_{11}$    $\triangleright\ c_1 = S_{13}$
```

So, we can rewrite the coefficients c_i in terms of evaluations $c(s)$ for $s \in S$ as follows:

$$
\begin{aligned}
c_0 &= c(0) \\
c_1 &= \frac{-20c(0) + 60c(1) - 30c(-1) - 15c(2) + 3c(-2) + 2c(3) - 720c(\infty)}{60} \\
c_2 &= \frac{-30c(0) + 16c(1) + 16c(-1) - c(2) - c(-2) + 96c(\infty)}{24} \\
c_3 &= \frac{10c(0) - 14c(1) - c(-1) + 7c(2) - c(-2) - c(3) + 360c(\infty)}{24} \\
c_4 &= \frac{6c(0) - 4c(1) - 4c(-1) + c(2) + c(-2) - 120c(\infty)}{24} \\
c_5 &= \frac{-10c(0) + 10c(1) + 5c(-1) - 5c(2) - c(-2) + c(3) - 360c(\infty)}{120} \\
c_6 &= c(\infty)
\end{aligned}
\tag{3.13}
$$

We multiply each equation in (3.11) and (3.13) to c_i correspondingly by a symbolic variable z_{6-i} for $i = 0, \ldots, 6$ and we take the sum of all equations, separately. From (3.11), we get the following:

$$
\begin{aligned}
z_6c_0 + z_5c_1 + z_4c_2 + z_3c_3 + z_2c_4 + z_1c_5 + z_0c_6 = &z_6a_0b_0 \\
&+z_5(a_0b_1 + a_1b_0) \\
&+z_4(a_0b_2 + a_1b_1 + a_2b_0) \\
&+z_3(a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0) \\
&+z_2(a_1b_3 + a_2b_2 + a_3b_1) \\
&+z_1(a_2b_3 + a_3b_2) \\
&+z_0a_3b_3
\end{aligned}
\tag{3.14}
$$

We rearrange the terms of (3.14) in the form $k_3b_3 + k_2b_2 + k_1b_1 + k_0b_0$ that we can express k_i as follows:

$$
\begin{pmatrix} k_3 \\ k_2 \\ k_1 \\ k_0 \end{pmatrix} = \begin{pmatrix} z_3 & z_2 & z_1 & z_0 \\ z_4 & z_3 & z_2 & z_1 \\ z_5 & z_4 & z_3 & z_2 \\ z_6 & z_5 & z_4 & z_3 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}
\tag{3.15}
$$

Doing the same to (3.13) gives us (3.16).

$$\begin{aligned}
&z_6c_0 + z_5c_1 + z_4c_2 + z_3c_3 + z_2c_4 + z_1c_5 + z_0c_6 \\
=&\frac{1}{120}\big[z_6 120c(0) \\
+&z_5\left(-40c(0) + 120c(1) - 60c(-1) - 30c(2) + 6c(-2) + 4c(3) - 1440c(\infty)\right) \\
+&z_4\left(-150c(0) + 80c(1) + 80c(-1) - 5c(2) - 5c(-2) + 480c(\infty)\right) \\
+&z_3\left(50c(0) - 70c(1) - 5c(-1) + 35c(2) - 5c(-2) - 5c(3) + 1800c(\infty)\right) \\
+&z_2\left(30c(0) - 20c(1) - 20c(-1) + 5c(2) + 5c(-2) - 600c(\infty)\right) \\
+&z_1\left(-10c(0) + 10c(1) + 5c(-1) - 5c(2) - c(-2) + c(3) - 360c(\infty)\right) \\
+&z_0\left(120c(\infty)\right).\big]
\end{aligned} \tag{3.16}$$

Using the equations in (3.12) we can write (3.16) in the form $k_3b_3+k_2b_2+k_1b_1+k_0b_0$ where

$$\begin{pmatrix} k_3 \\ k_2 \\ k_1 \\ k_0 \end{pmatrix} = \begin{pmatrix} P_1 - P_2 + 8P_3 - 8P_4 + 27P_5 + P_6 \\ P_1 + P_2 + 4P_3 + 4P_4 + 9P_5 \\ P_1 - P_2 + 2P_3 - 2P_4 + 3P_5 \\ P_0 + P_1 + P_2 + P_3 + P_4 + P_5 \end{pmatrix}, \tag{3.17}$$

where

$$\begin{aligned}
P_0 &= \frac{1}{12}\left(12z_6 - 4z_5 - 15z_4 + 5z_3 + 3z_2 - z_1\right)a_0, \\
P_1 &= \frac{1}{12}\left(12z_5 + 8z_4 - 7z_3 - 2z_2 + z_1\right)(a_0 + a_1 + a_2 + a_3), \\
P_2 &= \frac{1}{24}\left(-12z_5 + 16z_4 - z_3 - 4z_2 + z_1\right)(a_0 - a_1 + a_2 - a_3), \\
P_3 &= \frac{1}{24}\left(-6z_5 - z_4 + 7z_3 + z_2 - z_1\right)(a_0 + 2a_1 + 4a_2 + 8a_3), \\
P_4 &= \frac{1}{120}\left(6z_5 - 5z_4 - 5z_3 + 5z_2 - z_1\right)(a_0 - 2a_1 + 4a_2 - 8a_3), \\
P_5 &= \frac{1}{120}\left(4z_5 - 5z_3 + z_1\right)(a_0 + 3a_1 + 9a_2 + 27a_3), \\
P_6 &= \left(-12z_5 + 4z_4 + 15z_3 - 5z_2 - 3z_1 + z_0\right)a_3.
\end{aligned} \tag{3.18}$$

Since the left-hand sides of (3.15) and (3.17) are equal, so the right-hand sides must also be. Therefore, we obtain the TMVP-4 formula as follows:

$$\begin{pmatrix} z_3 & z_2 & z_1 & z_0 \\ z_4 & z_3 & z_2 & z_1 \\ z_5 & z_4 & z_3 & z_2 \\ z_6 & z_5 & z_4 & z_3 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} P_1 - P_2 + 8P_3 - 8P_4 + 27P_5 + P_6 \\ P_1 + P_2 + 4P_3 + 4P_4 + 9P_5 \\ P_1 - P_2 + 2P_3 - 2P_4 + 3P_5 \\ P_0 + P_1 + P_2 + P_3 + P_4 + P_5 \end{pmatrix}, \tag{3.19}$$

where P_i are as given in (3.18). Hence, with this formulation we define TMVP with 7 smaller TMVPs whose size are $1/4$ of the original.

- **TMVP-4 formula for multiplication modulo $x^n + 1$:**

Now, we apply the TMVP-4 formula to a TMVP representing polynomial multiplication modulo x^n+1. Let $a(x) = a_0+a_1x+a_2x^2+a_3x^3$ and $b(x) = b_0+b_1x+b_2x^2+b_3x^3$ be two n-term polynomials, which are divided into $n/4$-term polynomials a_i and b_i. Here, we assume that n is a divisible by four. Suppose $c(x)$ is the product of $a(x)$ and $b(x)$ modulo $x^n + 1$, i.e. $a(x)b(x) \mod x^n + 1 = c(x) = c_0 + c_1x + c_2x^2 + c_3x^3$. As we know from Section 2.2.1, the coefficients of $c(x)$ can be calculated via the following TMVP:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} a_0 & -a_3 & -a_2 & -a_1 \\ a_1 & a_0 & -a_3 & -a_2 \\ a_2 & a_1 & a_0 & -a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} \tag{3.20}$$

At least seven and four components are needed to identify the Toeplitz matrices in (3.19) and (3.20), respectively. So, the matrix evaluations of TMVP-4 formula in (3.9) become simpler for (3.20). Updating the P_i computations for $i = 0, \dots, 6$ in TMVP-4 formula accordingly, we have the following formula, which we refer to as TMVP-4 formula for $x^n + 1$ and denote as TMVP*-4:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} P_1 - P_2 + 8P_3 - 8P_4 + 27P_5 + P_6 \\ P_1 + P_2 + 4P_3 + 4P_4 + 9P_5 \\ P_1 - P_2 + 2P_3 - 2P_4 + 3P_5 \\ P_0 + P_1 + P_2 + P_3 + P_4 + P_5 \end{pmatrix} = \begin{pmatrix} a_0 & -a_3 & -a_2 & -a_1 \\ a_1 & a_0 & -a_3 & -a_2 \\ a_2 & a_1 & a_0 & -a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix},$$

where

$$
\begin{aligned}
P_0 &= \frac{1}{12}\left(5a_0 - 15a_1 - 3a_2 + 9a_3\right) b_0,\\
P_1 &= \frac{1}{12}\left(-7a_0 + 8a_1 + 11a_2 + 2a_3\right)\left(b_0 + b_1 + b_2 + b_3\right),\\
P_2 &= \frac{1}{24}\left(-a_0 + 16a_1 - 13a_2 + 4a_3\right)\left(b_0 - b_1 + b_2 - b_3\right),\\
P_3 &= \frac{1}{24}\left(7a_0 - a_1 - 5a_2 - a_3\right)\left(b_0 + 2b_1 + 4b_2 + 8b_3\right),\\
P_4 &= \frac{1}{120}\left(-5a_0 - 5a_1 + 7a_2 - 5a_3\right)\left(b_0 - 2b_1 + 4b_2 - 8b_3\right),\\
P_5 &= \frac{1}{120}\left(-5a_0 + 3a_2\right)\left(b_0 + 3b_1 + 9b_2 + 27b_3\right),\\
P_6 &= \left(15a_0 + 3a_1 - 9a_2 + 5a_3\right) b_3.
\end{aligned}
$$

3.2.1 Arithmetic Complexity Calculations

The arithmetic complexity calculations of the Toom-4, TMVP-4, and TMVP*-4 algorithms are explained in this section. Similar to the three-way case, we omit the multiplication/division-by-scalar operations.

- **Arithmetic Complexity of Toom-4:** To calculate the arithmetic complexity of the multiplication of two n-term polynomials via Toom-4, we count the additions and multiplications required by each step of the Toom-4 algorithm.

  1. **Evaluation:** The evaluation of the polynomial $a(x)$ at the points $S = \{0, 1, -1, 2, -2, 3, \infty\}$ is given as follows:

$$
\begin{aligned}
a(0) &= a_0\\
a(1) &= a_0 + a_1 + a_2 + a_3\\
a(-1) &= a_0 - a_1 + a_2 - a_3\\
a(2) &= a_0 + 2a_1 + 4a_2 + 8a_3\\
a(-2) &= a_0 - 2a_1 + 4a_2 - 8a_3\\
a(3) &= a_0 + 3a_1 + 9a_2 + 27a_3\\
a(\infty) &= a_3
\end{aligned}
$$

     11 additions of $n/4$-term polynomials with single-word coefficients are required for the evaluations above. The same is also valid for the evaluations of $b(x)$. Hence, the evaluation step of Toom-4 requires $(11n/2)A_S$ operations.

2. **Pointwise multiplication:** In this step we multiply $n/4$-term polynomials $a(s)$ and $b(s)$, and obtain the evaluations of the $(n/2-1)$-term polynomials $c(s)$ with double-word coefficients, for each $s \in S$. So, this step requires seven multiplications of $n/4$-term polynomials, i.e., $7M(n/4)$.
3. **Interpolation:** We use Algorithm 12 to calculate the coefficients of $c(x)$. So, interpolation requires 20 additions of $n/2-1$-term polynomials with double-word coefficients, i.e., $(10n-20)A_D$.
4. **Recomposition:** Since the coefficients c_i of $c(x)$ are $(n/2-1)$-term polynomials for $i = 0, \ldots, 6$, there are six overlaps of length $n/4-1$ occur in this step. So, this step requires $6(n/4-1)A_D$.

Hence, the total arithmetic complexity of n-term polynomial multiplication via Toom-4 can be expressed as follows :

$$M(n) = 7M(n/4) + (11n/2)A_S + (23n/2 - 26)A_D.$$

If $A_S = A_D$ then we have

$$M'(n) = 7M(n/4) + 17n - 26.$$

- **Arithmetic complexity of TMVP-4:** First, we compute the number of additions and multiplications needed to perform each step of TMVP-4 formula. Then, the sum of them gives the total arithmetic complexity.

  1. **Evaluation:** The complexity of the evaluation step is computed in two parts using Algorithm 13 and Algorithm 14.
     - **Matrix Evaluation:** Since z_i are $n/4 \times n/4$-dimensional Toeplitz matrices and each step of Algorithm 13 require $n/2-1$ single-word additions, matrix evaluation requires $(21n/2-21)A_S$ operations.
     - **Vector Evaluation:** The vectors a_i are of length $n/4$. So, $n/4$ single-word additions are performed in each step of Algorithm 14. Therefore, $(11n/4)A_S$ is the total number of operations for vector evaluations.

Algorithm 13 Matrix Evaluations of TMVP-4

```
1: $S_1 = 4z_5 - 5z_3$
2: $S_2 = S_1 + z_1$
3: $S_3 = z_4 - z_2$
4: $S_4 = S_2 - 5S_3$
5: $S_5 = S_4 + 2z_5$
6: $S_6 = S_3 + 4z_4$
7: $S_7 = 12z_6 - 3S_6$
8: $S_8 = S_7 - S_2$
9: $S_9 = z_1 + 6z_5$
10: $S_{10} = z_2 - z_0$
11: $S_{11} = 4S_3 - S_{10}$
12: $S_{12} = S_{11} - 3S_2$
13: $S_{13} = 4z_5 - z_3$
14: $S_{14} = S_2 + 2S_{13}$
15: $S_{15} = 4z_4 - z_2$
16: $S_{16} = S_{14} + 2S_{15}$
17: $S_{17} = z_5 - z_3$
18: $S_{18} = S_2 + 2S_{17}$
19: $S_{19} = -S_{18} - S_3$
20: $S_{20} = S_2 - 4S_{13}$
21: $S_{21} = S_{20} + 4S_{15}$
```

Algorithm 14 Vector Evaluations of TMVP-4

```
1: $S_1 = a_0 + a_2$
2: $S_2 = a_1 + a_3$
3: $S_3 = S_1 + S_2$
4: $S_4 = S_1 - S_2$
5: $S_5 = S_1 + 3a_2$
6: $S_6 = 2(S_2 + 3a_3)$
7: $S_7 = S_5 + S_6$
8: $S_8 = S_5 - S_6$
9: $S_9 = S_5 + 5a_2$
10: $S_{10} = 3(S_1 + S_6)$
11: $S_{11} = S_9 + S_{10}$
```

2. **Multiplication:** In this step, seven $n/4$-dimensional TMVPs are required to calculate P_i for $i = 0, \ldots, 6$. So, we have $7M(n/4)$ for the multiplication step.

3. **Recombination:** Recombination step requires thirteen vector additions as seen in Algorithm 15. Since P_i are vectors of length $n/4$ with double-word coefficients, the total cost of recombination is $(13n/4)A_D$.

Algorithm 15 Recombination of P_i for TMVP-4

```
1: $S_1 = P_1 + P_2$
2: $S_2 = P_1 - P_2$
3: $S_3 = P_3 + P_4$
4: $S_4 = P_3 - P_4$
5: $S_5 = S_1 + S_3$
6: $S_6 = P_0 + P_5$
7: $S_7 = S_5 + S_6$
8: $S_8 = S_2 - 2S_4$
9: $S_9 = S_8 + 3P_5$
10: $S_{10} = S_1 + 4S_3$
11: $S_{11} = S_{10} + 9P_5$
12: $S_{12} = S_2 + 8S_4$
13: $S_{13} = S_{12} + 27P_5$
```

Thus, adding all these costs gives the total arithmetic complexity of TMVP-4 formula as

$$M(n) = 7M(n/4) + (53n/4)A_S + (13n/4)A_D$$

or

$$M'(n) = 7M(n/4) + 33n/2 - 21.$$

- **Arithmetic complexity of TMVP*-4:** The matrix evaluation step is the only difference between the computations of TMVP-4 and TMVP*-4. So, calculating the cost of the matrix evaluation step of TMVP*-4 would suffice to obtain the total arithmetic complexity of TMVP*-4. The matrix evaluations of TMVP*-4 can be done by nineteen Toeplitz matrix additions of dimension $n/4 \times n/4$ with single-word coefficients. Therefore, the arithmetic complexity of TMVP*-4 formula is

$$M(n) = 7M(n/4) + (49n/4)A_S + (13n/4)A_D.$$

If $A_S = A_D$, then the cost of the TMVP*-4 formula is represented as follows:

$$M'(n) = 7M(n/4) + 31n/2 - 19.$$

CHAPTER 4

NEW APPROACH FOR TMVP: NON-SQUARE FORMULAS

In this section, we introduce some TMVP formulas with a non-square Toeplitz matrix, which we call non-square TMVPs in short. To derive a non-square formula with a Toeplitz matrix of dimension $n \times m$ or $m \times n$, in other words an $(n \times m)$- or $(m \times n)$-way TMVP, we use a multiplication algorithm for n- and m-term polynomials. Then, we derive new TMVPs with square Toeplitz matrices by composing non-square TMVP formulas.

4.1 (2×3)- and (3×2)-way TMVPs

To derive these formulas, we use the schoolbook method and another multiplication algorithm for two polynomials of degrees one and two. Let $c(x) = c_0 + c_1x + c_2x^2 + c_3x^3$ be the product of the polynomials $a(x) = a_0 + a_1x + a_2x^2$ and $b(x) = b_0 + b_1x$. First, we use the schoolbook method, which we know that the coefficients c_i are computed as follows:

$$\begin{aligned} c_0 &= a_0b_0 \\ c_1 &= a_0b_1 + a_1b_0 \\ c_2 &= a_1b_1 + a_2b_0 \\ c_3 &= a_2b_1 \end{aligned} \tag{4.1}$$

Then, we choose $S = \{0, 1, -1, \infty\}$ as the set of evaluation points to calculate the product polynomial. The evaluations are given in (4.2).

$$\begin{aligned} c(0) &= a_0b_0 \\ c(1) &= (a_0 + a_1 + a_2)(b_0 + b_1) \\ c(-1) &= (a_0 - a_1 + a_2)(b_0 - b_1) \\ c(\infty) &= a_2b_1. \end{aligned} \tag{4.2}$$

To interpolate the coefficients c_i, we use Algorithm 16.

Algorithm 16 Interpolation

```
1: $S_1 = c(0)$                              ▷ $c_0 = S_1$
2: $S_2 = c(\infty)$                         ▷ $c_3 = S_2$
3: $S_3 = \frac{c(1)-c(-1)}{2} - S_2$        ▷ $c_1 = S_3$
4: $S_4 = \frac{c(1)+c(-1)}{2} - S_1$        ▷ $c_2 = S_4$
```

Therefore, the coefficients are as follows:

$$\begin{aligned} c_0 &= c(0) \\ c_1 &= \frac{1}{2}(c(1) - c(-1)) - c(\infty) \\ c_2 &= \frac{1}{2}(c(1) + c(-1)) - c(0) \\ c_3 &= c(\infty) \end{aligned} \tag{4.3}$$

We multiply each equation corresponding to c_i by a symbolic variable z_{3-i} for $i = 0, 1, 2, 3$ and we take the sum of all equations. Then, we get (4.4) and (4.5) from (4.1) and (4.3), respectively.

$$\begin{aligned} z_3c_0 + z_2c_1 + z_1c_2 + z_0c_3 = &z_3a_0b_0 \\ &+z_2(a_0b_1 + a_1b_0) \\ &+z_1(a_1b_1 + a_2b_0) \\ &+z_0(a_2b_1), \end{aligned} \tag{4.4}$$

$$\begin{aligned} z_3c_0 + z_2c_1 + z_1c_2 + z_0c_3 =& z_3c(0) \\ &+z_2\left(\frac{c(1)-c(-1)}{2} - c(\infty)\right) \\ &+z_1\left(\frac{c(1)+c(-1)}{2} - c(0)\right) \\ &+z_0c(\infty), \end{aligned} \tag{4.5}$$

Since the left-hand sides of the (4.4) and (4.5) are equal, so the right-hand sides must be. We use this fact to derive two non-square TMVP formulas in the following sections.

4.1.1 TMVP-(2×3) formula with four multiplications

The coefficient k_1 of b_1 equals to $z_2a_0 + z_1a_1 + z_0a_2$ from Equation (4.4) and equals to $1/2(z_1+z_2)(a_0+a_1+a_2) - 1/2(z_1-z_2)(a_0-a_1+a_2) + (z_0-z_2)a_2$ from Equation (4.5). Similarly, the coefficient k_0 of b_0 equals to $z_3a_0 + z_2a_1 + z_1a_2$ from Equation (4.4) and equals to $(z_3-z_1)a_0 + 1/2(z_1+z_2)(a_0+a_1+a_2) + 1/2(z_1-z_2)(a_0-a_1+a_2)$ from Equation (4.5). Therefore, the (2×3)-way TMVP formula which we denote as TMVP-(2×3) is as follows:

$$\begin{pmatrix} k_1 \\ k_0 \end{pmatrix} = \begin{pmatrix} z_2 & z_1 & z_0 \\ z_3 & z_2 & z_1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \frac{P_0-P_1}{2} + P_2 \\ \frac{P_0+P_1}{2} + P_3 \end{pmatrix}, \tag{4.6}$$

where

$$\begin{aligned} P_0 &= (z_1 + z_2)(a_0 + a_1 + a_2), \\ P_1 &= (z_1 - z_2)(a_0 - a_1 + a_2), \\ P_2 &= (z_0 - z_2)a_2, \\ P_3 &= (z_3 - z_1)a_0. \end{aligned}$$

Arithmetic complexity of the TMVP-(2×3) formula is

$$M(n, s) = 4M(n/2, s/3) + (2n + 7s/3 - 4)A_S + 2nA_D$$

or

$$M'(n, s) = 4M(n/2, s/3) + 4n + 7s/3 - 4.$$

4.1.2 TMVP-(3×2) formula with four multiplications

Similar to the derivation of the TMVP-(2×3) formula, we use the coefficients k_2, k_1, k_0 of a_2, a_1, a_0, respectively, to obtain (3×2)-way TMVP formula. We denote this formula by TMVP-(3×2), and the formula is as follows:

$$\begin{pmatrix} k_2 \\ k_1 \\ k_0 \end{pmatrix} = \begin{pmatrix} z_1 & z_0 \\ z_2 & z_1 \\ z_3 & z_2 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = \begin{pmatrix} \frac{P_0+P_1}{2} + P_2 \\ \frac{P_0-P_1}{2} \\ \frac{P_0+P_1}{2} + P_3 \end{pmatrix}, \tag{4.7}$$

where

$$\begin{aligned} P_0 &= (z_1 + z_2)(b_0 + b_1), \\ P_1 &= (z_1 - z_2)(b_0 - b_1), \\ P_2 &= (z_0 - z_2)b_1, \\ P_3 &= (z_3 - z_1)b_0. \end{aligned}$$

Arithmetic complexity of the TMVP-(3×2) formula is

$$M(n, s) = 4M(n/3, s/2) + (4n/3 + 3s - 4)A_S + 4n/3A_D$$

or

$$M'(n, s) = 4M(n/3, s/2) + 8n/3 + 3s - 4.$$

4.2 (2×4)- and (4×2)-way TMVPs

To derive these formulas we utilize the algorithms for multiplying the polynomials $a(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ and $b(x) = b_0 + b_1x$. Let $a(x)b(x) = c(x) = c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4$ be the product polynomial. The schoolbook method gives the following:

$$\begin{aligned} c_0 &= a_0b_0, \\ c_1 &= a_0b_1 + a_1b_0, \\ c_2 &= a_1b_1 + a_2b_0, \\ c_3 &= a_2b_1 + a_3b_0, \\ c_4 &= a_3b_1. \end{aligned} \tag{4.8}$$

For the other algorithm, we choose $S = \{0, 1, -1, 2, \infty\}$ as the set of evaluation points to calculate the coefficients of the product polynomial $c(x)$. The evaluations are given in (4.9).

$$\begin{aligned} c(0) &= a_0 b_0 \\ c(1) &= (a_0 + a_1 + a_2 + a_3)(b_0 + b_1) \\ c(-1) &= (a_0 - a_1 + a_2 - a_3)(b_0 - b_1) \\ c(2) &= (a_0 + 2a_1 + 4a_2 + 8a_3)(b_0 + 2b_1) \\ c(\infty) &= a_3 b_1. \end{aligned} \tag{4.9}$$

To interpolate the coefficients c_i, we use Algorithm 17.

Algorithm 17 Interpolation

```
1: $S_1 = c(0)$                                   ▷ $c_0 = S_1$
2: $S_2 = c(\infty)$                              ▷ $c_4 = S_2$
3: $S_3 = \frac{c(1)+c(-1)}{2} - S_2 - S_1$       ▷ $c_2 = S_3$
4: $S_4 = \frac{c(1)-c(-1)}{2}$
5: $S_5 = \frac{c(2)-c(0)}{2} - 2S_3 - 8S_2$
6: $S_6 = \frac{S_5-S_4}{3}$                      ▷ $c_3 = S_6$
7: $S_7 = S_4 - S_6$                              ▷ $c_1 = S_7$
```

Therefore, the coefficients are as follows:

$$\begin{aligned} c_0 &= c(0) \\ c_1 &= -\frac{1}{6}(3c(0) + 2c(-1) + c(2)) + c(1) + 2c(\infty) \\ c_2 &= \frac{1}{2}(c(1) + c(-1)) - c(0) - c(\infty) \\ c_3 &= \frac{1}{6}(3c(0) - 3c(1) - c(-1) + c(2)) - 2c(\infty) \\ c_4 &= c(\infty) \end{aligned} \tag{4.10}$$

We multiply each equation corresponds to c_i by a symbolic variable z_{4-i} for $i = 0, 1, 2, 3, 4$ and we take the sum of all equations. Then, we get (4.11) and (4.12) from

(4.8) and (4.10), respectively.

$$\begin{aligned} z_4c_0 + z_3c_1 + z_2c_2 + z_1c_3 + z_0c_4 =& z_4a_0b_0 \\ &+z_3(a_0b_1 + a_1b_0) \\ &+z_2(a_1b_1 + a_2b_0) \\ &+z_1(a_2b_1 + a_3b_0) \\ &+z_0a_3b_1. \end{aligned} \tag{4.11}$$

$$\begin{aligned} z_4c_0 + z_3c_1 + z_2c_2 + z_1c_3 + z_0c_4 =& z_4c(0) \\ &+z_3\left(-\frac{1}{6}(3c(0) + 2c(-1) + c(2)) + c(1) + 2c(\infty)\right) \\ &+z_2\left(\frac{1}{2}\left(c(1) + c(-1)\right) - c(0) - c(\infty)\right) \\ &+z_1\left(\frac{1}{6}(3c(0) - 3c(1) - c(-1) + c(2)) - 2c(\infty)\right) \\ &+z_0c(\infty). \end{aligned} \tag{4.12}$$

Since the left-hand sides of the (4.11) and (4.12) are equal, so the right-hand sides must be. We use this fact to derive two non-square TMVP formulas in the following sections.

4.2.1 TMVP-(2×4) formula with five multiplications

Similar to the TMVP-(2×3), we rearrange the terms of both the right-hand sides of (4.11) and (4.12) in the form $k_1b_1 + k_0b_0$. This gives us the (2×4)-way formula, which we denote by TMVP-(2×4), as follows:

$$\begin{pmatrix} k_1 \\ k_0 \end{pmatrix} = \begin{pmatrix} z_3 & z_2 & z_1 & z_0 \\ z_4 & z_3 & z_2 & z_1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \frac{3P_1+P_2-2P_3}{6} + P_4 \\ \frac{3(P_0+P_1)-(P_2+P_3)}{6} \end{pmatrix},$$

where

$$
\begin{aligned}
P_0 &= (2z_4 - z_3 - 2z_2 + z_1)(a_0),\\
P_1 &= (2z_3 + z_2 - z_1)(a_0 + a_1 + a_2 + a_3),\\
P_2 &= (2z_3 - 3z_2 + z_1)(a_0 - a_1 + a_2 - a_3),\\
P_3 &= (z_3 - z_1)(a_0 + 2a_1 + 4a_2 + 8a_3),\\
P_4 &= (2z_3 - z_2 - 2z_1 + z_0)(a_3).
\end{aligned}
$$

4.2.2 TMVP-(4×2) formula with five multiplications

Similarly, we rearrange the terms of both the right-hand sides of (4.11) and (4.12) in the form $k_3a_3 + k_2a_2 + k_1a_1 + k_0a_0$. This gives us the (4×2)-way formula, which we denote by TMVP-(4×2), as follows:

$$
\begin{pmatrix} k_3 \\ k_2 \\ k_1 \\ k_0 \end{pmatrix} = \begin{pmatrix} z_1 & z_0 \\ z_2 & z_1 \\ z_3 & z_2 \\ z_4 & z_3 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = \begin{pmatrix} \frac{3P_1+P_2-8P_3}{6} + P_4 \\ \frac{3P_1-P_2-4P_3}{6} \\ \frac{3P_1+P_2-2P_3}{6} \\ \frac{3(P_0+P_1)-(P_2+P_3)}{6} \end{pmatrix},
$$

where

$$
\begin{aligned}
P_0 &= (2z_4 - z_3 - 2z_2 + z_1)(b_0),\\
P_1 &= (2z_3 + z_2 - z_1)(b_0 + b_1),\\
P_2 &= (2z_3 - 3z_2 + z_1)(b_0 - b_1),\\
P_3 &= (z_3 - z_1)(b_0 + 2b_1),\\
P_4 &= (2z_3 - z_2 - 2z_1 + z_0)(b_1).
\end{aligned}
$$

4.3 (3×4)- and (4×3)-way TMVPs

To derive these formulas we utilize the algorithms for multiplying the polynomials $a(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ and $b(x) = b_0 + b_1x + b_2x^2$. Let $a(x)b(x) = c(x) = c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4 + c_5x^5$ be the product polynomial. The schoolbook method gives the following:

$$\begin{aligned} c_0 &= a_0b_0 \\ c_1 &= a_0b_1 + a_1b_0 \\ c_2 &= a_0b_2 + a_1b_1 + a_2b_0 \\ c_3 &= a_1b_2 + a_2b_1 + a_3b_0 \\ c_4 &= a_2b_2 + a_3b_1 \\ c_5 &= a_3b_2 \end{aligned} \tag{4.13}$$

For the other algorithm we choose $S = \{0, 1, -1, 2, -2, \infty\}$ as the set of evaluation points. The evaluations of $c(x)$ at these points are given in (4.14).

$$\begin{aligned} c(0) &= a_0b_0 \\ c(1) &= (a_0 + a_1 + a_2 + a_3)(b_0 + b_1 + b_2) \\ c(-1) &= (a_0 - a_1 + a_2 - a_3)(b_0 - b_1 + b_2) \\ c(2) &= (a_0 + 2a_1 + 4a_2 + 8a_3)(b_0 + 2b_1 + 4b_2) \\ c(-2) &= (a_0 + 2a_1 + 4a_2 + 8a_3)(b_0 - 2b_1 + 4b_2) \\ c(\infty) &= a_3b_2. \end{aligned} \tag{4.14}$$

So, the coefficients are as follows:

$$\begin{aligned} c_0 =& c(0), \\ c_1 =& \frac{2(c(1) - c(-1))}{3} - \frac{c(2) - c(-2)}{12} + 4c(\infty), \\ c_2 =& \frac{-5c(0)}{4} + \frac{2(c(1) + c(-1))}{3} - \frac{c(2) + c(-2)}{24}, \\ c_3 =& \frac{-c(1) + c(-1)}{6} + \frac{c(2) - c(-2)}{12} - 5c(\infty), \\ c_4 =& \frac{c(0)}{4} - \frac{c(1) + c(-1)}{6} + \frac{c(2) + c(-2)}{24}, \\ c_5 =& c(\infty). \end{aligned} \tag{4.15}$$

We multiply each equation corresponding to c_i by a symbolic variable z_{5-i} for $i = 0, \ldots, 5$ and we take the sum of all equations. Then, we get (4.16) and (4.17) from (4.13) and (4.15), respectively.

$$\begin{aligned}z_5c_0 + z_4c_1 + z_3c_2 + z_2c_3 + z_1c_4 + z_0c_5 =& z_5a_0b_0\\ &+z_4(a_0b_1 + a_1b_0)\\ &+z_3(a_0b_2 + a_1b_1 + a_2b_0)\\ &+z_2(a_1b_2 + a_2b_1 + a_3b_0)\\ &+z_1(a_2b_2 + a_3b_1)\\ &+z_0a_3b_2.\end{aligned} \tag{4.16}$$

$$\begin{aligned}z_5c_0 + z_4c_1 + z_3c_2 + z_2c_3 + z_1c_4 + z_0c_5 =& z_5c(0)\\ &+z_4\left(\frac{2(c(1)-c(-1))}{3} - \frac{c(2)-c(-2)}{12} + 4c(\infty)\right)\\ &+z_3\left(\frac{-5c(0)}{4} + \frac{2(c(1)+c(-1))}{3} - \frac{c(2)+c(-2)}{24}\right)\\ &+z_2\left(\frac{-c(1)+c(-1)}{6} + \frac{c(2)-c(-2)}{12} - 5c(\infty)\right)\\ &+z_1\left(\frac{c(0)}{4} - \frac{c(1)+c(-1)}{6} + \frac{c(2)+c(-2)}{24}\right)\\ &+z_0c(\infty).\end{aligned} \tag{4.17}$$

4.3.1 TMVP(3×4) formula with six multiplications

Similar to the other formulas, we rearrange the terms of both the right-hand sides of (4.16) and (4.17) in the form $k_2b_2 + k_1b_1 + k_0b_0$. This gives us the (3×4)-way formula, which we denote by TMVP-(3×4), as follows:

$$\begin{pmatrix} k_2 \\ k_1 \\ k_0 \end{pmatrix} = \begin{pmatrix} z_3 & z_2 & z_1 & z_0 \\ z_4 & z_3 & z_2 & z_1 \\ z_5 & z_4 & z_3 & z_2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \frac{P_1+P_2+P_3+P_4}{6} + P_5 \\ \frac{P_1-P_2}{6} + \frac{P_3-P_4}{12} \\ \frac{P_1+P_2}{6} + \frac{6P_0+P_3+P_4}{24} \end{pmatrix},$$

where

$$\begin{aligned}P_0 &= (4z_5 - 5z_3 + z_1)(a_0),\\ P_1 &= (4z_4 + 4z_3 - z_2 - z_1)(a_0 + a_1 + a_2 + a_3),\\ P_2 &= (-4z_4 + 4z_3 + z_2 - z_1)(a_0 - a_1 + a_2 - a_3),\end{aligned}$$

$$
\begin{aligned}
P_3 &= (-2z_4 - z_3 + 2z_2 + z_1)(a_0 + 2a_1 + 4a_2 + 8a_3),\\
P_4 &= (2z_4 - z_3 - 2z_2 + z_1)(a_0 - 2a_1 + 4a_2 - 8a_3),\\
P_5 &= (4z_4 - 5z_2 + z_0)(a_3).
\end{aligned}
$$

4.3.2 TMVP-(4×3) formula with six multiplications

We rearrange the terms of both the right-hand sides of (4.16) and (4.17) in the form $k_3a_3 + k_2a_2 + k_1a_1 + k_0a_0$. This gives us the (4×3)-way formula, which we denote by TMVP-(4×3), as follows:

$$
\begin{pmatrix} k_3 \\ k_2 \\ k_1 \\ k_0 \end{pmatrix} = \begin{pmatrix} z_2 & z_1 & z_0 \\ z_3 & z_2 & z_1 \\ z_4 & z_3 & z_2 \\ z_5 & z_4 & z_3 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} \frac{P_1-P_2}{6} + \frac{P_3-P_4}{3} + P_5 \\ \frac{P_1+P_2+P_3+P_4}{6} \\ \frac{P_1-P_2}{6} + \frac{P_3-P_4}{12} \\ \frac{P_1+P_2}{6} + \frac{6P_0+P_3+P_4}{24} \end{pmatrix},
$$

where

$$
\begin{aligned}
P_0 &= (4z_5 - 5z_3 + z_1)(b_0),\\
P_1 &= (4z_4 + 4z_3 - z_2 - z_1)(b_0 + b_1 + b_2),\\
P_2 &= (-4z_4 + 4z_3 + z_2 - z_1)(b_0 - b_1 + b_2),\\
P_3 &= (-2z_4 - z_3 + 2z_2 + z_1)(b_0 + 2b_1 + 4b_2),\\
P_4 &= (2z_4 - z_3 - 2z_2 + z_1)(b_0 - 2b_1 + 4b_2),\\
P_5 &= (4z_4 - 5z_2 + z_0)(b_2).
\end{aligned}
$$

4.4 Composing non-square formulas

By composing non-square formulas, we can derive other square or non-square formulas. The different TMVP split values are given in Table 4.1 by composing the non-square TMVP formulas proposed in this section. The third column of the sixth row of Table 4.1 shows that composing the TMVP-(3×2) and TMVP-(3×4) formulas would result in a TMVP-(8×9) formula. Similarly, as we can see in the last column of the sixth row of the table, composing TMVP-(3×4) and TMVP-(4×3) gives a twelve-way TMVP formula (TMVP-12).

Table 4.1: Dimension of the Composed Split Formulas

TMVP	$(\mathbf{2\times 3})$	$(\mathbf{3\times 2})$	$(\mathbf{2\times 4})$	$(\mathbf{4\times 2})$	$(\mathbf{3\times 4})$	$(\mathbf{4\times 3})$
$(\mathbf{2\times 3})$	(4×9)	6	(4×12)	(8×6)	(6×12)	(8×9)
$(\mathbf{3\times 2})$	6	(9×4)	(6×8)	(12×4)	(9×8)	(12×6)
$(\mathbf{2\times 4})$	(4×12)	(6×8)	(4×16)	8	(6×16)	(8×12)
$(\mathbf{4\times 2})$	(8×6)	(12×4)	8	(16×4)	(12×8)	(16×6)
$(\mathbf{3\times 4})$	(6×12)	(9×8)	(6×16)	(12×8)	(9×16)	12
$(\mathbf{4\times 3})$	(8×9)	(12×6)	(8×12)	(16×6)	12	(16×9)

In this section, we use the TMVP in (4.18) as an example to explain how we compose the TMVP-(2×3) and TMVP-(3×2) formulas to obtain six-way TMVP formulas.

$$A.B = \begin{pmatrix} a_5 & a_4 & a_3 & a_2 & a_1 & a_0 \\ a_6 & a_5 & a_4 & a_3 & a_2 & a_1 \\ a_7 & a_6 & a_5 & a_4 & a_3 & a_2 \\ a_8 & a_7 & a_6 & a_5 & a_4 & a_3 \\ a_9 & a_8 & a_7 & a_6 & a_5 & a_4 \\ a_{10} & a_9 & a_8 & a_7 & a_6 & a_5 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}. \tag{4.18}$$

4.4.1 Six-way TMVP formula with sixteen multiplications

We can derive two six-way TMVP formulas with sixteen multiplications by composing the TMVP-(2×3) and TMVP-(3×2) formulas in different order.

1. Applying TMVP-(3×2) and TMVP-(2×3) formulas consecutively gives the following six-way TMVP formula which we denote by TMVP-6-1:

$$A.B = \begin{pmatrix} \frac{P_0-P_1+P_4-P_5}{4} + \frac{P_2+P_6+P_8-P_9}{2} + P_{10} \\ \frac{P_0+P_1+P_4+P_5}{4} + \frac{-P_3-P_7+P_8+P_9}{2} - P_{11} \\ \frac{P_0-P_1-P_4+P_5}{4} + \frac{P_2-P_6}{2} \\ \frac{P_0+P_1-P_4-P_5}{4} + \frac{-P_3+P_7}{2} \\ \frac{P_0-P_1+P_4-P_5}{4} + \frac{P_2+P_6-P_{12}+P_{13}}{2} - P_{14} \\ \frac{P_0+P_1+P_4+P_5}{4} + \frac{-P_3-P_7-P_{12}-P_{13}}{2} + P_{15} \end{pmatrix}, \tag{4.19}$$

where

$$
\begin{aligned}
P_0 &= (a_4 + a_5 + a_6 + a_7)(b_0 + b_1 + b_2 + b_3 + b_4 + b_5),\\
P_1 &= (a_4 - a_5 + a_6 - a_7)(b_0 - b_1 + b_2 + b_3 - b_4 + b_5),\\
P_2 &= (a_3 - a_7)(b_2 + b_5),\\
P_3 &= (a_4 - a_8)(b_0 + b_3),\\
P_4 &= (a_4 + a_5 - a_6 - a_7)(b_0 + b_1 + b_2 - b_3 - b_4 - b_5),\\
P_5 &= (a_4 - a_5 - a_6 + a_7)(b_0 - b_1 + b_2 - b_3 + b_4 - b_5),\\
P_6 &= (a_3 - 2a_5 + a_7)(b_2 - b_5),\\
P_7 &= (a_4 - 2a_6 + a_8)(b_0 - b_3),\\
P_8 &= (a_1 + a_2 - a_6 - a_7)(b_3 + b_4 + b_5),\\
P_9 &= (a_1 - a_2 - a_6 + a_7)(b_3 - b_4 + b_5),\\
P_{10} &= (a_0 - a_2 - a_5 + a_7)(b_5),\\
P_{11} &= (a_1 - a_3 - a_6 + a_8)(b_3),\\
P_{12} &= (a_4 + a_5 - a_8 - a_9)(b_0 + b_1 + b_2),\\
P_{13} &= (a_4 - a_5 - a_8 + a_9)(b_0 - b_1 + b_2),\\
P_{14} &= (a_3 - a_5 - a_7 + a_9)(b_2),\\
P_{15} &= (a_4 - a_6 - a_8 + a_{10})(b_0).
\end{aligned}
$$

The TMVP-6 formula calls sixteen TMVPs of dimension $n/6$ and requires divisions by 4.

$$
\begin{aligned}
M(n,n) =& 4M(n/3,n/2) + (13n/4 - 4)A_S + (4n/3)A_D\\
=& 4\left[4M(n/6,n/6) + (11n/6 - 4)A_S + (2n/3)A_D\right]\\
& + (13n/4 - 4)A_S + (4n/3)A_D\\
=& 16M(n/6,n/6) + (35n/3 - 20)A_S + 4nA_D
\end{aligned}
$$

If $A_S = A_D$, then we have

$$M(n) = 16M(n/6) + 47n/3 - 20.$$

2. Applying TMVP-(2×3) and TMVP(3×2) formulas consecutively gives the

following six-way TMVP formula which we denote by TMVP-6-2:

$$A.B = \begin{pmatrix} \frac{P_0+P_1-P_4-P_5}{4} + \frac{P_2-P_6+P_8+P_9}{2} + P_{10} \\ \frac{P_0-P_1-P_4+P_5}{4} + \frac{P_8-P_9}{2} \\ \frac{P_0+P_1-P_4-P_5}{4} + \frac{-P_3+P_7+P_9+P_9}{2} - P_{11} \\ \frac{P_0+P_1+P_4+P_5}{4} + \frac{P_2+P_6-P_{12}-P_{13}}{2} - P_{14} \\ \frac{P_0-P_1+P_4-P_5}{4} + \frac{-P_{12}+P_{13}}{2} \\ \frac{P_0+P_1+P_4+P_5}{4} + \frac{-P_3-P_7-P_{12}-P_{13}}{2} + P_{15} \end{pmatrix}, \tag{4.20}$$

where

$$\begin{aligned}
P_0 &= (a_3 + a_4 + a_5 + a_6)(b_0 + b_1 + b_2 + b_3 + b_4 + b_5), \\
P_1 &= (a_3 - a_4 + a_5 - a_6)(b_0 - b_1 + b_2 - b_3 + b_4 - b_5), \\
P_2 &= (a_2 - a_6)(b_1 + b_3 + b_5), \\
P_3 &= (a_3 - a_7)(b_0 + b_2 + b_4), \\
P_4 &= (a_3 + a_4 - a_5 - a_6)(b_0 + b_1 - b_2 - b_3 + b_4 + b_5), \\
P_5 &= (a_3 - a_4 - a_5 + a_6)(b_0 - b_1 - b_2 + b_3 + b_4 - b_5), \\
P_6 &= (a_2 - 2a_4 + a_6)(b_1 - b_3 + b_5), \\
P_7 &= (a_3 - 2a_5 + a_7)(b_0 - b_2 + b_4), \\
P_8 &= (a_1 + a_2 - a_5 - a_6)(b_4 + b_5), \\
P_9 &= (a_1 - a_2 - a_5 + a_6)(b_4 - b_5), \\
P_{10} &= (a_0 - a_2 - a_4 + a_6)(b_5), \\
P_{11} &= (a_1 - a_3 - a_5 + a_7)(b_4), \\
P_{12} &= (a_3 + a_4 - a_8 - a_9)(b_0 + b_1), \\
P_{13} &= (a_3 - a_4 - a_8 + a_9)(b_0 - b_1), \\
P_{14} &= (a_2 - a_4 - a_7 + a_9)(b_1), \\
P_{15} &= (a_3 - a_5 - a_8 + a_{10})(b_0).
\end{aligned}$$

This formula calls 16 TMVPs of dimension $n/6$ and requires division by 4.

$$\begin{aligned}
M(n,n) =& 4M(n/2, n/3) + (13n/4 - 4)A_S + (2n)A_D \\
=& 4\left[4M(n/6, n/6) + (5n/3 - 4)A_S + (2n/3)A_D\right] \\
& + (13n/4 - 4)A_S + (2n)A_D \\
=& 16M(n/6, n/6) + (11n - 20)A_S + (14n/3)A_D
\end{aligned}$$

If $A_S = A_D$, then we have

$$M'(n) = 16M(n/6) + 47n/3 - 20.$$

So, no matter in which order we use the TMVP-(2×3) and TMVP-(3×2) formulas, we obtain a six-way formula with sixteen $n/6$-dimensional TMVPs. Similarly, an eight-way formula composed from TMVP-(2×4) and TMVP-(4×2) would require twenty-five $n/8$-dimensional TMVPs, and a twelve-way formula composed from TMVP-(3×4) and TMVP-(4×3) would require thirty-six $n/12$-dimensional TMVPs. The number of multiplications a formula requires gives a rough idea of its efficiency. While they require more multiplications, the number of multiplication/division-by-scalar operations required by six-, eight-, and twelve-way formulas introduced in this chapter are less than the formulas derived by composing the TMVP-2, TMVP-3_5, TMVP-4 formulas.

CHAPTER 5

APPLICATION TO SABER

As explained in Section 2.2.1, polynomial multiplication modulo $x^n \pm 1$ can be expressed as a TMVP. Therefore, developing efficient algorithms for Toeplitz matrix-vector multiplication leads to efficient multiplication algorithms for polynomial quotient rings with the modulus polynomial $x^n \pm 1$. We utilize the new TMVP-4 formula proposed in Section 3 to develop an efficient residue polynomial multiplication algorithm.

5.1 TMVP-based Multiplication in $\mathbb{Z}_{2^m}[x]/\langle x^{256}+1\rangle$

Let $a(x) = \sum_{i=0}^{255} a_i x^i$ and $b(x) = \sum_{i=0}^{255} b_i x^i$ be two polynomials in the ring $\mathcal{R}_{2^m} = \mathbb{Z}_{2^m}[x]/\langle x^{256}+1\rangle$. The coefficients of the product polynomial $c(x) = \sum_{i=0}^{255} c_i x^i \in \mathcal{R}_{2^m}$ can be calculated via the TMVP given in (5.1).

$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{254} \\ c_{255} \end{pmatrix} = \begin{pmatrix} a_0 & -a_{255} & \dots & -a_2 & -a_1 \\ a_1 & a_0 & \dots & -a_3 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{254} & a_{253} & \dots & a_0 & -a_{255} \\ a_{255} & a_{254} & \dots & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{pmatrix}. \tag{5.1}$$

To calculate (5.1) efficiently, we utilize TMVP-4 formula proposed in Chapter 3, together with the formulas in the literature that we stated in Chapter 2. We calculate (5.1) via TMVP-4 as follows:

$$\begin{aligned}\begin{pmatrix}C_0\\C_1\\C_2\\C_3\end{pmatrix} &= \begin{pmatrix}A_0 & -A_3 & -A_2 & -A_1\\A_1 & A_0 & -A_3 & -A_2\\A_2 & A_1 & A_0 & -A_3\\A_3 & A_2 & A_1 & A_0\end{pmatrix}\begin{pmatrix}B_0\\B_1\\B_2\\B_3\end{pmatrix}\\ &= \begin{pmatrix}P_1 - P_2 + 8P_3 - 8P_4 + 27P_5 + P_6\\ P_1 + P_2 + 4P_3 + 4P_4 + 9P_5\\ P_1 - P_2 + 2P_3 - 2P_4 + 3P_5\\ P_0 + P_1 + P_2 + P_3 + P_4 + P_5\end{pmatrix},\end{aligned} \tag{5.2}$$

where

$$\begin{aligned}
P_0 &= \frac{(5A_0 - 15A_1 - 3A_2 + 9A_3)\,B_0}{12},\\
P_1 &= \frac{(-7A_0 + 8A_1 + 11A_2 + 2A_3)\,(B_0 + B_1 + B_2 + B_3)}{12},\\
P_2 &= \frac{(-A_0 + 16A_1 - 13A_2 + 4A_3)\,(B_0 - B_1 + B_2 - B_3)}{24},\\
P_3 &= \frac{(7A_0 - A_1 - 5A_2 - A_3)\,(B_0 + 2B_1 + 4B_2 + 8B_3)}{24},\\
P_4 &= \frac{(-5A_0 - 5A_1 + 7A_2 - 5A_3)\,(B_0 - 2B_1 + 4B_2 - 8B_3)}{120},\\
P_5 &= \frac{(-5A_0 + 3A_2)\,(B_0 + 3B_1 + 9B_2 + 27B_3)}{120},\\
P_6 &= (15A_0 + 3A_1 - 9A_2 + 5A_3)\,B_3,
\end{aligned}$$

where the partitions A_i are Toeplitz matrices of dimension 64, and B_i, C_i are vectors of length 64 for $i = 0, \ldots, 3$. The number of additions required to compute P_i decreases by $4n = 1024$ compared to (3.19), because of the special form of the Toeplitz matrix in (5.2).

Utilizing TMVP formulas allows us to split our computation into many similar computations of smaller sizes. We can use these splitting methods consecutively to reduce the dimension to a level that the schoolbook matrix-vector multiplication is more efficient than using the formulas. TMVP formulas are more efficient than schoolbook matrix-vector multiplication for large n values, but for small dimensions like $n = 2$, the schoolbook method is more efficient than TMVP formulas. The level of switching the multiplication method to the schoolbook, i.e., the threshold, might differ depending on the dimension n, the modulus q, the formula being used, and the implementation platform. The threshold must be chosen carefully depending on those factors to develop efficient algorithms.

In our case, we want to establish a TMVP-based multiplication algorithm utilizing the TMVP-4 formula for Saber and implement it on the ARM Cortex-M4 microcontroller. To make use of the benefits of SIMD instructions, we place the components of the matrices (or equivalently, the coefficients of the polynomials) into registers pairwise. It means that we operate on modulo 2^{16}. In other words, we develop an algorithm for multiplication in $\mathcal{R}_{2^{16}}$, and then we apply a modular reduction to obtain a result in $\mathcal{R}_{2^m}$ for $m < 16$, which can be done easily by deleting the most significant $16 - m$ bits or equivalently taking the least significant m bits. One caveat of working in $\mathbb{Z}_{2^{16}}$ is the division by powers of two. Since 2 has no inverse in $\mathbb{Z}_{2^{16}}$, shifting right by r bits is the only way of performing a division by 2^r after reduction modulo 2^{16}. This may cause a loss in the most significant bits of the final result. A formula that requires a division by 2^r can work correctly if $m + r \leq 16$ for the modulus 2^m. To be more precise, for the modulus $q = 2^{13} = 2^m$ of Saber, we have $r \leq 3$; that is, any method that requires a division by 2^r with $r \leq 3$ works correctly. So, we can afford to lose at most three bits. We already start our multiplication algorithm with a layer of TMVP-4 formula, which requires divisions by 2^3 and obtain seven TMVPs of dimension 64. For these 64-dimensional TMVP computations, we can not use a formula that contains a division by a power of two because we lose the maximum number of bits we can by applying the TMVP-4 formula. It leaves us only two options: the two-way TMVP formula TMVP-2 given in (2.3) or the schoolbook matrix-vector multiplication since none of them require a division by a power of two.

At this point, we must determine the dimension for which the schoolbook matrix-vector multiplication is faster than the TMVP_2 formula. For this, we implement the schoolbook matrix-vector multiplication and the TMVP_2 formula for small dimensions and compare their cycle counts. Since $n = 256$ and we use only four- and two-way split methods, we restrict our search to powers of two. In Section 7.1, we explain how we determine 16 as the threshold value for ARM Cortex-M4 and give the results of the application to Saber. Thus, we develop a multiplication algorithm for Saber, which uses a layer of TMVP-4 followed by two layers of TMVP-2, and completes the multiplication step with 63 schoolbook matrix-vector multiplications of dimension 16. After applying the recombination steps according to the path we use, we obtain the result of (5.1).

$$\mathbf{256} \xrightarrow{\text{TMVP-4}} \mathbf{64} \xrightarrow{\text{TMVP-2}} \mathbf{32} \xrightarrow{\text{TMVP-2}} \mathbf{16}$$

5.2 Improved multiplication for Saber

For large dimensions using TMVP formulas is more efficient than the schoolbook method, whereas the schoolbook is superior to TMVP formulas for small dimensions. Since Saber is a scheme based on the MLWR problem, it requires matrix-vector multiplications (e.g., $\boldsymbol{A^T s}$ in line 4 of Algorithm 3) and inner products of two vectors (e.g., $\boldsymbol{b^T s'}$ in line 4 of Algorithm 4) with polynomial components. The components of the matrices and the vectors are from $\mathcal{R}_q$ or $\mathcal{R}_p$. For example, for $\ell = 2$, the matrix-vector multiplication in (5.3) is used in both encryption and decryption algorithms, where a_{ij} and s_j are polynomials in $\mathbb{Z}_{2^{13}}[x]/\langle x^{256}+1\rangle$.

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \end{pmatrix} = \begin{pmatrix} a_{00}s_0 + a_{01}s_1 \\ a_{10}s_0 + a_{11}s_1 \end{pmatrix}. \tag{5.3}$$

From Section 2.2.1, we know that each multiplication $a_{ij}s_j$ in $\mathbb{Z}_{2^{13}}[x]/\langle x^{256}+1\rangle$ can be represented as a TMVP $A_{ij}S_j$ where A_{ij} is the Toeplitz matrix formed with the coefficients of the polynomial a_{ij}, and S_j is the vector representation of the coefficients of the polynomial s_j. So, the right hand side of (5.3) is equivalent to the following:

$$\begin{pmatrix} A_{00}S_0 + A_{01}S_1 \\ A_{10}S_0 + A_{11}S_1 \end{pmatrix}, \tag{5.4}$$

where $A_{ij}S_j$ is a 256-dimensional TMVPs of the form (5.2). In the generic TMVP-based algorithm for multiplication in $\mathbb{Z}_{2^{13}}[x]/\langle x^{256}+1\rangle$ we propose in Section 5.1, we perform the evaluation, multiplication, and recombination steps for each $A_{ij}S_j$ calculation and we obtain the final result with 256 additions. In our improved algorithm, we use a non-recursive version of the block recombination method to reduce the number of operations, expecting to increase efficiency.

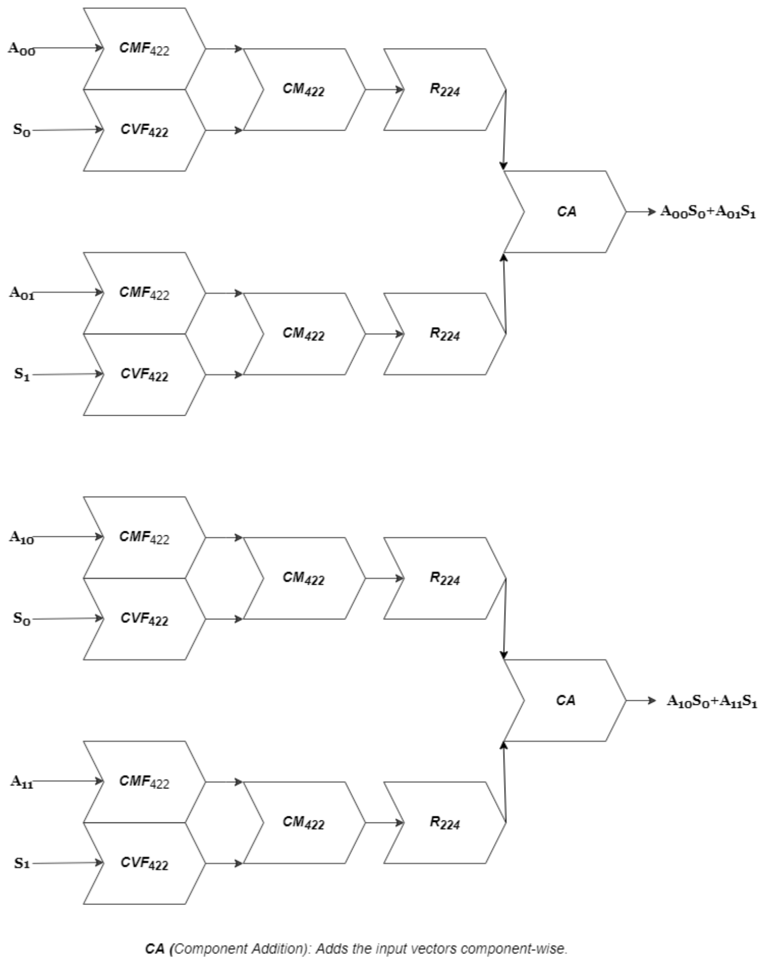
Let us explain our modified version of the block recombination method on the TMVPs in (5.4). We use the notation $Op_{i_1 i_2 \dots i_t}$ to denote the consecutive calculations of the operation *Op* corresponding to TMVP-i_1, TMVP-$i_2, \dots,$ TMVP-i_t formulas, respectively. For example, CVF_{422} denotes the component vector formation step corresponds to the vector evaluations ($\mathbf{E}_{\mathsf{vctr}}$) of a layer formula (5.2) followed by two layers of formula (2.3) and is defined as follows:

$$
\begin{aligned}
\mathit{CVF}_{422}(V) = (&\mathit{CVF}_{22}(V_0),\\
&\mathit{CVF}_{22}(V_0 + V_1 + V_2 + V_3),\\
&\mathit{CVF}_{22}(V_0 - V_1 + V_2 - V_3),\\
&\mathit{CVF}_{22}(V_0 + 2V_1 + 4V_2 + 8V_3),\\
&\mathit{CVF}_{22}(V_0 - 2V_1 + 4V_2 + 8V_3),\\
&\mathit{CVF}_{22}(V_0 + 3V_1 - 9V_2 + 27V_3),\\
&\mathit{CVF}_{22}(V_3)),\\
\mathit{CVF}_{22}(W) = (&\mathit{CVF}_{2}(W_0) + \mathit{CVF}_{2}(W_1),\\
&\mathit{CVF}_{2}(W_1),\\
&\mathit{CVF}_{2}(W_0))\\
\mathit{CVF}_{2}(B) = (&B_0 + B_1, B_1, B_0),
\end{aligned}
$$

where

$$
V = \begin{pmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{pmatrix}, W = \begin{pmatrix} W_0 \\ W_1 \end{pmatrix}, B = \begin{pmatrix} B_0 \\ B_1 \end{pmatrix}
$$

In the above definition, CVF_2 denotes the vector evaluation corresponds to a layer of formula (2.3) and CVF_{22} denotes two consecutive CVF_2 evaluations. After applying CVF_{422} to a vector of length 256, we end up with 63 vectors of length 16. Component matrix formation CMF_{422} and reconstruction R_{224} are defined similarly by using the formulas (2.3) and (5.2). The output of $\mathit{CMF}_{422}(A)$ for a 256-dimensional Toeplitz matrix A is 63 Toeplitz matrices of dimension 16×16. The component multiplication step in our optimized algorithm is componentwise multiplication of two vectors of length 63 which have 16-dimensional Toeplitz matrices and vectors of length 16 as their components, respectively. We refer to this operation as CM_{422} which require 63 TMVPs of dimension 16 which are calculated via schoolbook matrix-vector multiplication. With our generic algorithm proposed in Section 5.1, computation of (5.4) requires 4 CMF_{422}, 4 CVF_{422}, 4 CM_{422}, 4 R_{224}, and 256 additions. With our optimized algorithm that utilizes an altered version of the block recombination method, the computation of (5.4) requires 4 CMF_{422}, 2 CVF_{422}, 4 CM_{16}, 126 additions, and 2 R_{224}. Therefore, our optimization reduces the number of operations. We also observe this improvement in our implementation, the results of which we share in the next chapter.



A00
S0
CMF422
CVF422
CM422
R224
CA
A00S0+A01S1
A01
S1
A10
A11
A10S0+A11S1
CA (Component Addition): Adds the input vectors component-wise.

Figure 5.1: Diagram of the Generic TMVP-based Algorithm

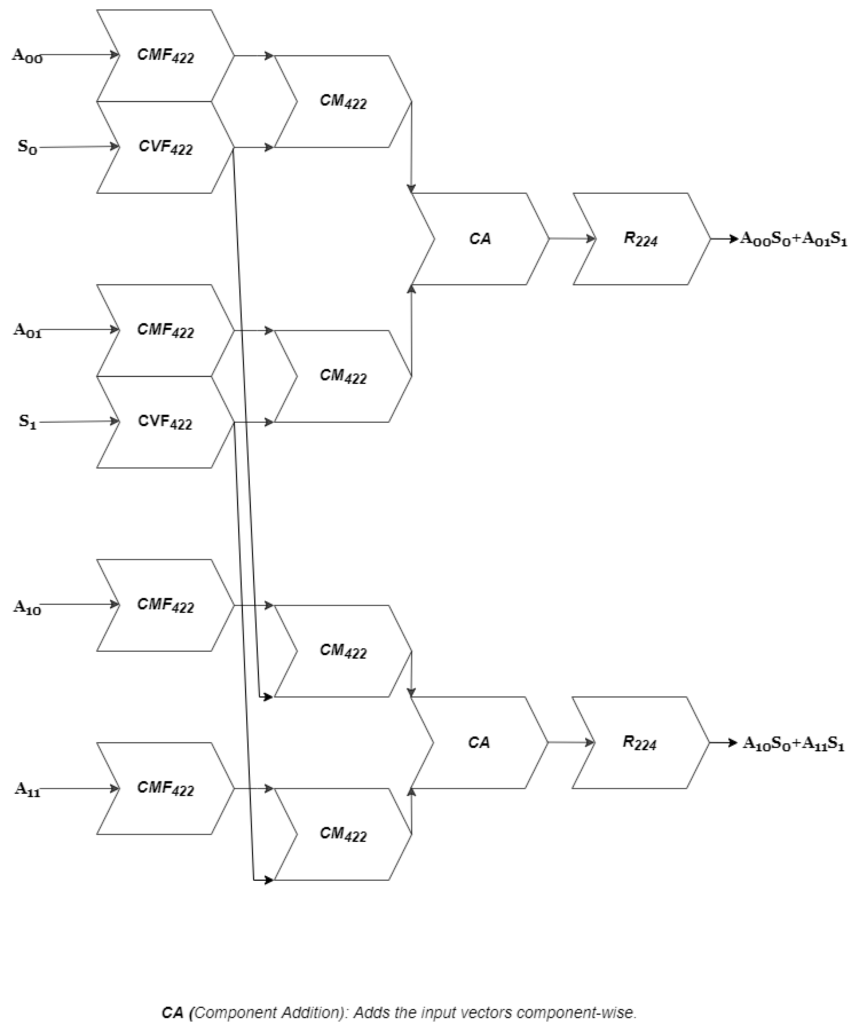


Figure 5.2: Diagram of the Improved TMVP-based Algorithm

CHAPTER 6

APPLICATION TO NTRU

In this chapter, we propose algorithms utilizing the TMVP-4 formula for all parameter sets of NTRU. Unlike Saber, NTRU uses $x^n - 1$ as the modulus polynomial where n takes one of the prime values from $\{509, 677, 701, 821\}$. The TMVPs representing the polynomial multiplication modulo $x^n - 1$ are also n-dimensional. Using a split formula is not possible when the dimension of TMVP is prime. In the following section, we explain how we pad the prime dimensional TMVPs to enable using the formulas.

6.1 Padding prime-dimensional TMVPs

NTRU uses multiplication in the ring $\mathbb{Z}_q[x]/\langle x^n - 1\rangle$ for key generation, encryption and decryption algorithms. As we mention in Section 2.2.1, we can represent the multiplication $c(x) = a(x)b(x)$ in $\mathbb{Z}_q[x]/\langle x^n - 1\rangle$ as the following TMVP:

$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} a_0 & a_{n-1} & \dots & a_2 & a_1 \\ a_1 & a_0 & \dots & a_3 & a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & \dots & a_0 & a_{n-1} \\ a_{n-1} & a_{n-2} & \dots & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{pmatrix} \tag{6.1}$$

where $a(x) = \sum_{i=0}^{n-1} a_i x^i, b(x) = \sum_{i=0}^{n-1} b_i x^i$, and $c(x) = \sum_{i=0}^{n-1} c_i x^i$. The $n \times n$ Toeplitz matrix in (6.1) also has the cyclic property. A Toeplitz matrix with this property can be specified by only n of its components instead of $2n-1$, which allows to use simplified versions of TMVP formulas. For all parameter sets of NTRU, the modulus q is a power of two and the dimension n is a prime. The prime-dimension prevents us using the split formulas directly. It would not be convenient to use the schoolbook method for efficient matrix-vector multiplication for these dimensions. So, we pad these prime-dimensional TMVPs to facilitate the TMVP formulas. Our padding strategy for the Toeplitz matrix in (6.1) is adding as many zeros as needed to the first row and the first column until we attain the targeted dimension and complete the rest of the entries in such a way that preserves the Toeplitz matrix structure. On the other hand, we append just as many zero entries at the end of the vector. For example, if we decide to obtain an m-dimensional TMVP from (6.1) by padding, we would have the following TMVP:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{m-n} \\ \vdots \\ c_{n-2} \\ c_{n-1} \\ \hline c_n \\ \vdots \\ c_{m-2} \\ c_{m-1} \end{pmatrix} = \left(\begin{array}{ccccc|ccc} a_0 & a_{n-1} & \cdots & a_2 & a_1 & 0 & \cdots & 0 \\ a_1 & a_0 & \cdots & a_3 & a_2 & a_1 & \cdots & 0 \\ a_2 & a_1 & \cdots & a_4 & a_3 & a_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m-n} & a_{m-n-1} & \cdots & a_{m-n+2} & a_{m-n+1} & a_{m-n} & \cdots & a_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-2} & a_{n-3} & \cdots & a_0 & a_{n-1} & a_{n-2} & \cdots & a_{2n-m-1} \\ a_{n-1} & a_{n-2} & \cdots & a_1 & a_0 & a_{n-1} & \cdots & a_{2n-m} \\ \hline 0 & a_{n-1} & \cdots & a_2 & a_1 & a_0 & \cdots & a_{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a_{n-1} & a_{m-n} & a_{m-n-1} & a_{m-n-2} & \cdots & a_{n-1} \\ 0 & 0 & \cdots & a_{m-n+1} & a_{m-n} & a_{m-n-1} & \cdots & a_0 \end{array}\right) \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{m-n} \\ \vdots \\ b_{n-2} \\ b_{n-1} \\ \hline 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \tag{6.2}$$

Here, we assume that the dimension m is a proper composite value for using TMVP formulas. As seen in (6.2), we append $m - n$ zero entries at the end of the vector and the first row and column of the Toeplitz matrix in (6.1). Then, we fill the entries so that elements on a line parallel to the main diagonal are the same. Therefore, we obtain a TMVP of a targeted dimension with our padding technique. The padding technique we suggest gives us a proper dimensional TMVP, but it cannot preserve the cyclic feature of the matrix. Although this situation prevents us from using the simplified version of TMVP formulas, this does not hinder our algorithms from performing efficiently. After padding, we calculate the TMVP in (6.2) efficiently via TMVP split formulas and obtain the resulting vector of length m. Finally, ignoring the last $m - n$ entries of this vector gives us the result of (6.1), which is a vector of length n. Prior to all of these steps, we first need to decide the dimension m before padding. We decide the suitable dimensions depending on several factors, such as n, split formulas we use, and the size for small schoolbook multiplications. Since we want to utilize the

TMVP-4 formula, the first condition on the dimension is to be divisible by four. In the following sections, we elaborately explain our choices for dimensions of padded TMVPs for every parameter set of NTRU.

6.2 TMVP-based Multiplication for NTRU

In the following sections, we present the algorithms we propose for ntruhps2048509, ntruhrss701, ntruhps2048677, and ntruhps4096821. First, we pad the TMVP, and then we apply various TMVP formulas consecutively until we reach the threshold dimension at which the schoolbook matrix-vector multiplication is more reasonable than the formulas.

6.2.1 Multiplication Algorithm for ntruhps2048509

For this parameter set of NTRU, we have $q = 2^{11}$ and $n = 509$ with the modulus polynomial $x^{509} - 1$. So, ntruhps2048509 requires multiplication in the ring $\mathbb{Z}_{2^{11}}[x]/\langle x^{509} - 1\rangle$ which can be calculated via the TMVP in (6.3).

$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{507} \\ c_{508} \end{pmatrix} = \begin{pmatrix} a_0 & a_{508} & \cdots & a_2 & a_1 \\ a_1 & a_0 & \cdots & a_3 & a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{507} & a_{506} & \cdots & a_0 & a_{508} \\ a_{508} & a_{507} & \cdots & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{507} \\ b_{508} \end{pmatrix} \tag{6.3}$$

We would not consider the schoolbook matrix-vector multiplication algorithm as an option for this dimension. So, we should pad both the Toeplitz matrix and the vector in (6.3) to obtain a suitable dimension for using the TMVP-4 formula followed by other TMVP formulas, which yields small dimensional TMVPs. Since we want to use the TMVP-4 formula, we start checking the options for dimensions with the smallest multiple of four that exceeds 509, which is 512. The TMVP-4 formula yields seven 128-dimensional TMVPs when it is applied to a 512-dimensional TMVP. Since 128 is a power of two and applying another TMVP-4 formula is out of the question, we are free to apply as many layers of TMVP-2 formulas as needed until we reach a size in which the schoolbook is faster than TMVP formulas. According to our implementation results on ARM Cortex-M4 (Section 7), we choose the threshold as 16, as we do for Saber.

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_{507} \\ c_{508} \\ \hdashline c_{509} \\ c_{510} \\ c_{511} \end{pmatrix} = \left(\begin{array}{cccccccc:ccc} a_0 & a_{508} & a_{507} & \cdots & a_3 & a_2 & a_1 & & 0 & 0 & 0 \\ a_1 & a_0 & a_{508} & \cdots & a_4 & a_3 & a_2 & & a_1 & 0 & 0 \\ a_2 & a_1 & a_0 & \cdots & a_5 & a_4 & a_3 & & a_2 & a_1 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ a_{507} & a_{506} & a_{505} & \cdots & a_1 & a_0 & a_{508} & & a_{507} & a_{506} & a_{505} \\ a_{508} & a_{507} & a_{506} & \cdots & a_2 & a_1 & a_0 & & a_{508} & a_{507} & a_{506} \\ \hdashline 0 & a_{508} & a_{507} & \cdots & a_3 & a_2 & a_1 & & a_0 & a_{508} & a_{507} \\ 0 & 0 & a_{508} & \cdots & a_4 & a_3 & a_2 & & a_1 & a_0 & a_{508} \\ 0 & 0 & 0 & \cdots & a_5 & a_4 & a_3 & & a_2 & a_1 & a_0 \end{array}\right) \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_{n-2} \\ b_{n-1} \\ \hdashline 0 \\ 0 \\ 0 \end{pmatrix} \tag{6.4}$$

As seen in (6.4), we add three zero entries at the end of the first row and the first column of the Toeplitz matrix and complete the rest accordingly to preserve the Toeplitz structure. Similarly, we pad the vector with three zero entries as well. We use the TMVP-4 formula followed by three layers of the TMVP-2 formula and end up with 7.3.3.3=189 schoolbook matrix-vector multiplications of dimension 16.

$$\mathbf{512} \xrightarrow{\text{TMVP-4}} \mathbf{128} \xrightarrow{\text{TMVP-2}} \mathbf{64} \xrightarrow{\text{TMVP-2}} \mathbf{32} \xrightarrow{\text{TMVP-2}} \mathbf{16}$$

We use the name TMVPmul-509-512 to refer the algorithm following the path given above. TMVPmul-509-512 performs 189 schoolbook matrix-vector multiplications and gives us 189 vectors of length 16. Then, we recombine these vectors according to the formulas we use, and obtain $(c_0, c_1, \ldots, c_{508}, c_{509}, c_{510}, c_{511})$ of length 512 as the result of (6.4). Omitting the last three terms $c_{509}, c_{510}, c_{511}$ gives us $(c_0, c_1, \ldots, c_{508})$ of length 509 which is the result of (6.3) that we were looking for at the beginning. The results of the Cortex-M4 implementation of the algorithm TMVPmul-509-512 are given in Table 7.2. The results show that TMVPmul-509-512 is the most efficient algorithm for ntruhps2048509 compared to any other algorithm in the literature. For the algorithms in the next sections, we skip some detailed explanations we give in this section to prevent unnecessary repetitions.

6.2.2 Multiplication Algorithm for ntruhrss701

We have $q = 2^{13}$, $n = 701$, and $f(x) = x^{701} - 1$ for ntruhrss701. Just like in the previous case, we start checking the dimensions with the smallest multiple of four that is larger than 701, which is 704. After applying the TMVP-4 formula to a 704-dimensional TMVP, we have seven 176-dimensional TMVPs. Since 176 is a multiple of 16, we can apply four layers of the TMVP-2 formula and end up with 567 TMVPs of dimension 11 to be calculated via the schoolbook method. We named this algorithm TMVPmul-701-704.

$$\mathbf{704} \xrightarrow{\text{TMVP-4}} \mathbf{176} \xrightarrow{\text{TMVP-2}} \mathbf{88} \xrightarrow{\text{TMVP-2}} \mathbf{44} \xrightarrow{\text{TMVP-2}} \mathbf{22} \xrightarrow{\text{TMVP-2}} \mathbf{11}$$

We implement the algorithm TMVPmul-701-704 and improve the performance of multiplication in $\mathbb{Z}_{2^{13}}[x]/\langle x^{701} - 1\rangle$. However, it is known that accessing addresses that are not divisible by four causes a performance hit. Therefore, we see no harm in checking the higher dimensions to find a path that requires small and even dimensional TMVPs to be calculated via the schoolbook matrix-vector multiplication. So, we keep searching for the dimensions with small and even dimensional TMVPs at the end to see whether the it works faster or not.

$$\mathbf{708} \xrightarrow{\text{TMVP-4}} \mathbf{177} \xrightarrow{\text{TMVP-3}_6} \mathbf{59}$$
$$\mathbf{712} \xrightarrow{\text{TMVP-4}} \mathbf{178} \xrightarrow{\text{TMVP-2}} \mathbf{89}$$
$$\mathbf{716} \xrightarrow{\text{TMVP-4}} \mathbf{179}$$

As can be seen above, for the values 708, 712, and 716 there do not exist a path that ends up with a small dimension. Finally, we see that 720 is another possible dimension for an efficient computation tracing the following path:

$$\mathbf{720} \xrightarrow{\text{TMVP-4}} \mathbf{180} \xrightarrow{\text{TMVP-3}_6} \mathbf{60} \xrightarrow{\text{TMVP-3}_6} \mathbf{20} \xrightarrow{\text{TMVP-2}} \mathbf{10}$$

To make the first assessment of this algorithm which we call TMVPmul-701-720, we implement the 10-dimensional schoolbook Toeplitz matrix-vector product and compare the performance of two algorithms roughly on paper. The cycle counts of 10- and 11-dimensional schoolbook Toeplitz matrix-vector multiplications are 110 and 147.

Therefore, the TMVPmul-701-704 algorithm takes $567 \cdot 147 = 83349$ cycles and the TMVPmul-701-720 algorithm takes $756 \cdot 110 = 83160$ cycles for performing schoolbook multiplications. Since these values are very close to each other, we implement the TMVPmul-701-720 algorithm to see whether it is faster than TMVPmul-701-704 or not. The TMVPmul-701-704 algorithm is slightly faster than the TMVPmul-701-720 but it consumes a little bit more stack memory. Therefore, both algorithms can be preferred for efficient implementations. We use the results of TMVPmul-701-704 for comparisons in Section 7.2. The results show that TMVPmul-701-704 is faster than any other Cortex-M4 implementations in the literature.

6.2.3 Multiplication Algorithm for ntruhps2048677

For ntruhps2048677 the parameters are given as $q = 2^{11}$, $n = 677$, and $f(x) = x^{677} - 1$. Following the same strategy, we start checking the dimensions with the smallest multiple of four exceeding 677 and eliminate those requiring schoolbook matrix-vector multiplications with dimensions larger than 16.

$$\begin{aligned}
&\mathbf{680} \xrightarrow{\text{TMVP-4}} \mathbf{170} \xrightarrow{\text{TMVP-2}} \mathbf{85}\\
&\mathbf{684} \xrightarrow{\text{TMVP-4}} \mathbf{171} \xrightarrow{\text{TMVP-3}_6} \mathbf{57} \xrightarrow{\text{TMVP-3}_6} \mathbf{19}\\
&\mathbf{688} \xrightarrow{\text{TMVP-4}} \mathbf{172} \xrightarrow{\text{TMVP-2}} \mathbf{86} \xrightarrow{\text{TMVP-2}} \mathbf{43}\\
&\mathbf{692} \xrightarrow{\text{TMVP-4}} \mathbf{173}\\
&\mathbf{696} \xrightarrow{\text{TMVP-4}} \mathbf{174} \xrightarrow{\text{TMVP-3}_6} \mathbf{58} \xrightarrow{\text{TMVP-2}} \mathbf{29}\\
&\mathbf{700} \xrightarrow{\text{TMVP-4}} \mathbf{175}
\end{aligned}$$

680 is the first one we try, which yields seven 170-dimensional TMVPs after applying the TMVP-4 formula. For 170, the only option is TMVP-2 which yields three 85-dimensional TMVPs. 85 is not a multiple of two or three, and it is too large for the dimension of a TMVP to a compute via schoolbook algorithm. So, we continue checking with 684, which yields 19-dimensional TMVPs after a layer of TMVP-4 followed by two layers of TMVP-3_6. We eliminate this path because 19 is not a small enough dimension for an efficient schoolbook matrix-vector multiplication considering the implementation platform we use. For the dimensions 688, 692, 696, and 700, a path does not exist that ends up with small enough TMVPs via a combination

of four-, three-, and two-way formulas. The next dimension is 704, which we have already examined for ntruhrss701. Appending 27 zero entries at the end of the vector and the first row and column of the Toeplitz matrix gives us 704-dimensional padded TMVP. The multiplication algorithm for ntruhps2048677 which we denote by TMVPmul-677-704 tracing the same path as TMVPmul-701-704 is implemented on ARM Cortex-M4 and share the results in Table 7.2. Because of the same motive explained above, we also implement the algorithm that uses 720 for the dimension of the padded matrix for ntruhps2048677 as well. We pad the TMVP similarly with 43 zero entries and obtain a 720-dimensional padded TMVP. We denote this algorithm by TMVPmul-677-720 tracing the same path as TMVPmul-701-720. The results show that our implementation of TMVPmul-677-704 is the fastest compared to other Cortex-M4 implementations in the literature.

In this work, for TMVPmul-677-720 we prefer to use the three-way TMVP formula that requires six multiplications (TMVP-3_6) in (2.4). Here we should note that, since the modulus of ntruhps2048677 is 2^{11}, the maximum value of i is 5, as explained in Section 2.5. The three-way TMVP formula with five multiplications (TMVP-3_5) can also be preferred for the TMVPmul-677-720 algorithm. TMVP-3_5 has lower arithmetic complexity, but it requires divisions by two. Applying two layers of TMVP-3_5 formula (requires divisions by 2^2) after a layer of TMVP-4 (requires divisions by 2^3) requires divisions by 2^5, i.e., shifting right by 5 bits. Since we make our calculations over $\mathbb{Z}_{2^{16}}$, this may cause losing the most significant five bits of some coefficients. Fortunately, ntruhps2048677 can afford these losses because reducing the coefficients modulo 2^{11} (i.e., masking the result with 0x07ff) removes those lost bits and provides the correct result.

6.2.4 Multiplication Algorithm for ntruhps4096821

For this parameter set of NTRU, we have $q = 2^{12}$ and $n = 821$ with the modulus polynomial $x^{821} - 1$. Starting with the nearest multiple of four which is 824, we check possible paths. For 824 and 828, we cannot find a path.

$$\mathbf{824} \xrightarrow{\text{TMVP-4}} \mathbf{206} \xrightarrow{\text{TMVP-2}} \mathbf{103}$$
$$\mathbf{828} \xrightarrow{\text{TMVP-4}} \mathbf{207} \xrightarrow{\text{TMVP-}3_6} \mathbf{69} \xrightarrow{\text{TMVP-}3_6} \mathbf{23}$$

For 832 we find the following path:

$$\mathbf{832} \xrightarrow{\text{TMVP-4}} \mathbf{208} \xrightarrow{\text{TMVP-2}} \mathbf{104} \xrightarrow{\text{TMVP-2}} \mathbf{52} \xrightarrow{\text{TMVP-2}} \mathbf{26} \xrightarrow{\text{TMVP-2}} \mathbf{13}$$

We implement this algorithm that we call TMVPmul-821-832, and we observe that it is faster than Toom but a little slower than NTT. Similar to the previous cases, we also check for another path that ends up with a small dimension. We see that 864 is another suitable dimension for the padded matrix. The multiplication algorithm we propose for ntruhps4096821 which is referred by TMVPmul-821-864 traces the following path:

$$\mathbf{864} \xrightarrow{\text{TMVP-4}} \mathbf{216} \xrightarrow{\text{TMVP-}3_6} \mathbf{72} \xrightarrow{\text{TMVP-}3_6} \mathbf{24} \xrightarrow{\text{TMVP-2}} \mathbf{12}$$

Before implementing TMVPmul-821-864 completely, we implement 12-dimensional schoolbook TMVP to compare the algorithms roughly. 12- and 13-dimensional schoolbook TMVP calculations take 162 and 221 cycles, respectively. Therefore, TMVPmul-821-832 takes $567 \cdot 221 = 125307$ cycles, whereas TMVPmul-821-864 takes $756 \cdot 162 = 122472$ cycles. We implement TMVPmul-821-864 and observe that it is slightly faster than TMVPmul-821-832. Unlike the previous cases, the larger dimensional padded matrix leads to a more efficient implementation because of the even-dimensional schoolbook matrix-vector multiplications. The benchmark results of TMVPmul-821-864 on ARM Cortex-M4 are given in Table 7.2. Note that, for a similar reason we explained for ntruhps2048677 in the previous section, a layer of TMVP-3_5 formula may be preferred instead one of the TMVP-3_6 in TMVPmul-821-864.

CHAPTER 7

IMPLEMENTATION AND BENCHMARKS

In this section, we share the cycle counts of the implementations of the schoolbook matrix-vector multiplication and the TMVP-2 formula for small dimensions. The benchmarking results for the application of our algorithms to Saber and NTRU are also presented in this section.

7.1 Implementation Results for Saber

As explained in Section 5.1, to complete our multiplication algorithm, we need to determine the maximum value of the dimension for which the schoolbook method is faster than the TMVP-2 formula. For this, we compare the cycle counts of the schoolbook method and the TMVP-2 formula for TMVPs of dimension $n = 2^t$ for small t values.

For $t = 1$, we implement both the schoolbook matrix-vector multiplication and the TMVP-2 formula, which require 10 and 16 clock cycles, respectively. We implement the schoolbook matrix-vector multiplication and observe that it requires 23 clock cycles for $t = 2$. We know that for $t = 2$, the TMVP-2 formula calls three schoolbook matrix-vector multiplication of dimension 2, which would take more than $3 \cdot 10 = 30$ cycles. We do not implement the TMVP-2 formula for $t = 2$ and conclude that the schoolbook method is preferable for this dimension. A similar observation shows that also for $t = 3$, the schoolbook is faster. Table 7.1 shows the cycle counts of the schoolbook matrix-vector multiplication and the TMVP-2 formula for various $n = 2^t$ values.

Table 7.1: Schoolbook vs. TMVP-2

t	n	SB(n)	TMVP-2(n)
1	2	10	16
2	4	23	$> 3 \times 10 = 30$
3	8	67	$> 3 \times 23 = 69$
4	16	280	401
5	32	1313	1082

For $t = 4$, schoolbook method takes 280 cycles which is not less than $3 \cdot 67 = 201$. So, we implement the TMVP-2 method, which requires 401 clock cycles. Finally, for $t = 5$, we implement both algorithms and observe that the TMVP-2 formula is faster than the schoolbook method. So, $t = 4$ is the maximum value that the schoolbook matrix-vector multiplication is faster than the TMVP-2 formula for the dimension 2^t. Hence, 16 is the threshold. Now that we determine the threshold, we know how many layers of the TMVP-2 formula we apply before switching the multiplication method to the schoolbook.

So, our TMVP-based algorithm for multiplication in $\mathcal{R}_{2^{13}} = \mathbb{Z}_{2^{13}}[x]/\langle x^{256} + 1\rangle$ uses the TMVP-4 formula to split the computations into seven 64-dimensional TMVPs. Then, to each of these seven TMVPs, we apply the TMVP-2 formula twice successively and end up with 16-dimensional TMVPs. We perform sixty-three schoolbook matrix-vector multiplications in total and recombine their results according to the formulas to obtain the final result. We implement this algorithm on the ARM Cortex-M4 to compare the results with [31]. To make a fair comparison, we evaluate the polynomial multiplication algorithm in [31] with the polynomial reduction step since our algorithm already includes it. As can be seen in Table 7.2, our algorithm for multiplication in $\mathcal{R}_{2^{16}}$ is 24.5% faster and requires 16.5% less memory than the one in [31], which uses Toom-4 and the Karatsuba algorithms.

Table 7.2: Multiplication in $\mathcal{R}_{2^{16}}$

Cycles			Stack		
[31]	This work	Imp.	[31]	This work	Imp.
37804	28520	24.5%	3800	3172	16.5%

In this work, we only focus on an efficient residue polynomial multiplication algorithm and optimization of this algorithm for Saber, not on a complete implementation of Saber. So, we use the publicly available codes from [31], [32], and [38] to compose software packages for our applications to Saber. We make some adjustments to existing codes to integrate our algorithm into these packages. The source codes of our application are publicly available at `https://github.com/iremkp/Saber-tmvp4-m4`. We compare the results with the ones given in [38] and [33].

As Saber is an MLWR-based scheme, it performs polynomial matrix-vector multiplications for the key generation, encapsulation, and decapsulation algorithms. Table 7.3 shows the cycle counts of the polynomial matrix-vector multiplication (e.g., $\boldsymbol{A^T s}$ in line 5 of Algorithm 1 in [23]) that Saber public-key encryption scheme use for the key generation and encryption algorithms. Here, the matrix $\boldsymbol{A}$ is of dimension $\ell \times \ell$ and the vector $\boldsymbol{s}$ is of dimension $\ell \times 1$. They both have polynomial components from the ring $\mathcal{R}_q$.

Table 7.3: Polynomial Matrix-Vector Multiplication

	[33]	TMVP	[38]	Block Rec.	
$\ell = 2$	162 k	122 k	159 k	106 k	cycles
$\ell = 3$	361 k	273 k	317 k	231 k	cycles
$\ell = 4$	646 k	483 k	528 k	403 k	cycles

In Table 7.3, the comparison of cycle counts of matrix-vector multiplication using different algorithms are given. The third and fifth columns of Table 7.3 represent the generic TMVP-based algorithm and the optimized algorithm explained in Section 2.2.1 and Section 5.2, respectively. Our generic algorithm improves the polynomial matrix-vector multiplication approximately 25% for all values of ℓ comparing the results from [33]. Moreover, our optimized algorithm outperforms the polynomial matrix-vector multiplication in [38] by 33.3%, 27.1%, 23.6% for $\ell = 2, 3, 4$, respectively.

The effect of our algorithms on the overall performance of Saber is also promising. Table 7.4 shows the cycle counts and the stack usage of different implementations on ARM Cortex-M4 microcontroller of the key generation, encapsulation, and decapsulation algorithms of LightSaber ($\ell = 2$), Saber ($\ell = 3$), and FireSaber ($\ell = 4$). As can be seen in Table 7.4, our optimized algorithm is the fastest compared to [38] and [33]. Table 7.5 shows the percentage of the gain in terms of the execution time that our

Table 7.4: Results of application to Saber

		[33]	[38] (speed)	TMVP	Block Rec.	
LightSaber	KeyGen:	$460\ k$	$466\ k$	$421\ k$	$409\ k$	cycles
		9656	14208	7932	12536	bytes
	Encaps:	$651\ k$	$653\ k$	$591\ k$	$572\ k$	cycles
		11392	15928	9668	14248	bytes
	Decaps:	$679\ k$	$678\ k$	$597\ k$	$574\ k$	cycles
		12136	16672	10412	14992	bytes
Saber	KeyGen:	$896\ k$	$853\ k$	$810\ k$	$772\ k$	cycles
		13256	19824	12608	18144	bytes
	Encaps:	$1161\ k$	$1103\ k$	$1052\ k$	$996\ k$	cycles
		15544	22088	14872	20392	bytes
	Decaps:	$1204\ k$	$1127\ k$	$1058\ k$	$995\ k$	cycles
		16640	23184	15968	21488	bytes
FireSaber	KeyGen:	$1449\ k$	$1340\ k$	$1297\ k$	$1224\ k$	cycles
		20144	26448	20120	24776	bytes
	Encaps:	$1787\ k$	$1642\ k$	$1590\ k$	$1499\ k$	cycles
		23008	29228	22968	27592	bytes
	Decaps:	$1853\ k$	$1679\ k$	$1606\ k$	$1508\ k$	cycles
		24592	30768	24448	29072	bytes

algorithms achieve. We speed up the key generation between 3.2% and 10.5%, encapsulation between 3.2% and 11%, and decapsulation between 4.3% and 13.3% with our generic TMVP-based algorithm. The improvements in efficiency with our optimized algorithm are between 8.7% and 15.5% for key generation, 8.7% and 16.1% for encapsulation, 9.7% and 18.7% for decapsulation.

Our TMVP-based multiplication algorithm consumes less memory than [33] and the speed-optimized version in [38]. The percentage of improvements in memory utilization can be seen in Table 7.6. Our optimized algorithm for Saber using the block recombination method requires less memory than the speed-optimized version in [38]. Although our optimized algorithm is the fastest, it consumes more stack memory than [33].

Table 7.5: Speed ups

			[33]	[38](speed)
TMVP	LightSaber	KeyGen:	-8.4%	-9.6%
		Encaps:	-9.2%	-9.5%
		Decaps:	-12.0%	-11.9%
	Saber	KeyGen:	-9.6%	-5.0%
		Encaps:	-9.4%	-4.6%
		Decaps:	-12.1%	-6.1%
	FireSaber	KeyGen:	-10.5%	-3.2%
		Encaps:	-11.0%	-3.2%
		Decaps:	-13.3%	-4.3%
Block Rec.	LightSaber	KeyGen:	-11.1%	-12.2%
		Encaps:	-12.1%	-12.4%
		Decaps:	-15.8%	-15.6%
	Saber	KeyGen:	-13.8%	-9.5%
		Encaps:	-14.2%	-9.7%
		Decaps:	-17.4%	-11.7%
	FireSaber	KeyGen:	-15.5%	-8.7%
		Encaps:	-16.1%	-8.7%
		Decaps:	-18.6%	-10.2%

7.2 Implementation Results of NTRU

In the previous section, we present the TMVP-based algorithms for multiplication in $\mathbb{Z}_q[x]/\langle x^n - 1\rangle$ for different values of n and q. We implement these algorithms on ARM Cortex-M4, a recommended platform by NIST for evaluating post-quantum cryptographic schemes on microcontrollers. The digital signal processing (DSP) instructions that the Cortex-M4 microprocessor supports allow us to perform operations on halfwords of different registers simultaneously. These instructions are called SIMD (single instruction multiple data) and enable us to implement efficient matrix-vector multiplications for small dimensions. The coefficients of the polynomials are less than 2^{13} for all parameter sets of NTRU. Since we build the TMVPs with these coefficients, the entries of the Toeplitz matrices and vectors in TMVPs are also less than 2^{13}. Therefore, we place two entries into one register to make use of these instructions. The source code of our implementation can be found at `https://github.com/iremkp/NTRU-tmvp4-m4.git`.

Table 7.6: Improvements in memory utilization

			[33]	[38](speed)
TMVP	LightSaber	KeyGen:	-17.9%	-44.2%
		Encaps:	-15.1%	-39.3%
		Decaps:	-14.2%	-37.5%
	Saber	KeyGen:	-4.9%	-36.4%
		Encaps:	-4.3%	-32.7%
		Decaps:	-4.0%	-31.1%
	FireSaber	KeyGen:	-0.1%	-23.9%
		Encaps:	-0.2%	-21.4%
		Decaps:	-0.5%	-20.5%
Block Rec.	LightSaber	KeyGen:	+29.8%	-11.8%
		Encaps:	+25.0%	-10.5%
		Decaps:	+23.5%	-10.1%
	Saber	KeyGen:	+36.9%	-8.5%
		Encaps:	+31.2%	-7.7%
		Decaps:	+29.1%	-7.3%
	FireSaber	KeyGen:	+22.9%	-6.3%
		Encaps:	+19.9%	-5.6%
		Decaps:	+15.4%	-5.5%

The adjustments on parameters required by both the NTT and TMVP-based methods are given in Table 7.7. The first two columns contain the original n and q parameters of NTRU, whereas the values in the middle two columns are used to enable the NTT method for NTRU. The last two columns of Table 7.7 contain the n, q pairs used by the TMVPmul-509-512, TMVPmul-677-720, TMVPmul-701-720, and TMVPmul-821-864 algorithms, respectively.

Table 7.7: Parameter values

		NTT [19]		TMVPmul	
n	q	n	q	n	q
509	2048	1024	1043969	512	2048
677	2048	1536	1389569	720	2048
701	8192	1536	5747201	720	8192
821	4096	1725	3365569	864	4096

Unlike the NTT method, TMVP-based algorithms do not entail a modification on modulus q as can be seen in Table 7.7. In fact, the modifications on n required by TMVP-based multiplication algorithms seem negligible compared to the NTT method. We think that being able to keep the parameters relatively smaller has a remarkable effect on the performance of the TMVP-based algorithms. The performance results of our multiplication algorithms and the others in the literature are given in Table 7.8. The results in Table 7.8 are obtained utilizing the benchmarking software from [19]. The first column contains the results of the TMVP-based algorithms we propose. The second column shows the results of the state-of-the-art NTT multiplication from [32], whereas the last column shows the Toom results from [19]. The codes are compiled with arm-none-eabi-gcc version 10.3.1. The cycle counts are the average of the results of 100 executions.

Table 7.8: Comparison of the algorithms for polynomial multiplication

	Cycles		
	TMVPmul	NTT [32]	Toom [19]
ntruhps2048509	**81054**	103271	108717
ntruhps2048677	**142109**	147810	182150
ntruhrss701	**142252**	148190	179994
ntruhps4096821	192996	**182153**	239018
	Stack		
	TMVPmul	NTT [32]	Toom [19]
ntruhps2048509	**7028**	8332	9696
ntruhps2048677	**11300**	12376	12980
ntruhrss701	**11296**	12372	13408
ntruhps4096821	**12776**	13964	15696

The cycle count for Toom method includes the polynomial reduction.

As the results in Table 7.8 show, TMVP-based algorithms reduce the stack usage, and except for the TMVPmul-821-864, they all improve the ring multiplication. Our TMVPmul-509-512 algorithm is 21.5% faster and consumes 15.6% less stack memory than the NTT method. Similarly, TMVPmul-509-512 is 25.4% faster and consumes 27.5% less stack memory than the Toom-Cook method. We reduce the stack usage by 8.7% compared to NTT with TMVPmul-701-704 and TMVPmul-677-704 algorithms and improve the ring multiplication by 3.9% and 4.2%, respectively. These algorithms are also 22% and 26.5% faster and 12.9% and 15.7% more memory efficient than the Toom method. While TMVPmul-821-864 algorithm consumes 8.5%

less stack memory, it is 5.9% slower than NTT. On the other hand, TMVPmul-821-864 is 19.2% faster and requires 18.6% less memory than the Toom method.

To observe the effect of the proposed TMVP-based algorithms on the overall performance of NTRU and compare it with the others, we use the benchmarking software [32] (commit c37e541). We integrate the assembly codes that we write for the algorithms TMVPmul-509-512, TMVPmul-677-704, TMVPmul-701-704, and TMVPmul-821-864 to the implementations of ntruhps2048509, ntruhps2048677, ntruhrss701, and ntruhps4096821, respectively. We do the same for Toom results with the assembly codes from [19]. Table 7.9 shows the benchmark results of the applications of the proposed TMVP-based multiplication algorithms to both NTRU CPA-DPKE and NTRU CCA-KEM. The comparison of the results of our algorithms and the others can also be seen in Table 7.9. The negative percentages in the parentheses represent the improvements that TMVP algorithms achieve compared to the corresponding algorithm. For the ones that do not improve the performance (NTT implementation of encapsulation and encryption algorithms of ntruhps4096821) we use a positive sign to denote the percentage of increase in the cycle counts of the TMVP method.

Improving polynomial multiplication impacts the performance of the key generation algorithm of NTRU CPA-DPKE. Compared the NTT, our algorithm speed up the key generation of NTRU CPA-DPKE by 12.5%, 9.2%, 9.4%, and 7.6%, and reduce the stack memory usage by 14.1%, 6.7%, 8.1%, and 9.4% for $n = 509$, $n = 677$, $n = 701$, and $n = 821$, respectively. The encryption and decryption algorithms are also accelerated in most cases. The percentages of the improvements can be found in Table 7.9. The key generation, encryption, and decryption algorithms of NTRU CPA-DPKE require five, one, and three ring multiplications, respectively. TMVP-based algorithms can target all of these multiplications, whereas the NTT method in [32] targets three in key generation, one in encryption, and one in decryption, respectively. For this reason, the improvements are more prominent for key generation and decryption than they are for encryption. We also achieve improvements in NTRU CCA-KEM schemes. Similar to NTRU CPA-DPKE, we improve the performance of key generation and decapsulation more than we do for encapsulation. Our application outperforms the NTT method for NTRU CCA-KEM in all cases except for the encryption and encapsulation of ntruhps4096821 and reduces the stack usage in all cases. Furthermore, we report the size evaluation results of our application and others in Table 7.10. We share the improvement percentages of our code in parentheses compared to the corresponding implementation. We do not include *.data* and *.bss* values to the table since they are all zero. The size evaluation results show that we reduce the flash footprint between 8.9% and 17%.

Table 7.9: Results of application to NTRU

			This work	NTT	Toom	
ntruhps2048509 (Sec.Level:1)	CCA KEM	KeyGen:	**2517** k	2870 k (-12.3%)	2887 k (-12.8%)	cycles
			18676	21344 (-12.5%)	21344 (-12.5%)	bytes
		Encaps:	**544** k	566 k (-3.9%)	572 k (-4.9%)	cycles
			12756	14060 (-9.3%)	15424 (-17.3%)	bytes
		Decaps:	**460** k	539 k (-14.7%)	545 k (-15.6%)	cycles
			12132	14800 (-18%)	14800 (-18%)	bytes
	CPA DPKE	KeyGen:	**2474** k	2828 k (-12.5%)	2845 k (-13%)	cycles
			16244	18912 (-14.1%)	18912 (-14.1%)	bytes
		Enc:	**132** k	154 k (-14.3%)	160 k (-17.5%)	cycles
			10116	11420 (-11.4%)	12784 (-20.9%)	bytes
		Dec:	**357** k	434 k (-17.7%)	441 k (-19.1%)	cycles
			11172	13840 (-19.3%)	13840 (-19.3%)	bytes
ntruhps2048677 (Sec.Level:3)	CCA KEM	KeyGen:	**4172** k	4592 k (-9.1%)	4692 k (-11.1%)	cycles
			26772	28460 (-5.9%)	28640 (-5.9%)	bytes
		Encaps:	**807** k	812 k (-1%)	848 k (-4.8%)	cycles
			18900	19976 (-5.4%)	20580 (-8.2%)	bytes
		Decaps:	**719** k	806 k (-10.8%)	842 k (-14.6%)	cycles
			18052	19732 (-8.5%)	19732 (-8.5%)	bytes
	CPA DPKE	KeyGen:	**4119** k	4536 k (-9.2%)	4643 k (-11.3%)	cycles
			23540	25228 (-6.7%)	25228 (-6.7%)	bytes
		Enc:	**210** k	216 k (-2.8%)	251 k (-16.3%)	cycles
			15396	16472 (-6.5%)	17076 (-9.8%)	bytes
		Dec:	**580** k	667 k (-13%)	703 k (-17.5%)	cycles
			16788	18468 (-9.1%)	18468 (-9.1%)	bytes
ntruhrss701 (Sec.Level:3)	CCA KEM	KeyGen:	**3803** k	4204 k (-9.5%)	4304 k (-11.6%)	cycles
			25384	27512 (-7.7%)	27512 (-7.7%)	bytes
		Encaps:	**362** k	368 k (-1.6%)	401 k (-9.7%)	cycles
			17240	18316 (-5.9%)	19352 (-10.9%)	bytes
		Decaps:	**779** k	862 k (-9.6%)	895 k (-13%)	cycles
			18448	20560 (-10.3%)	20560 (-10.3%)	bytes
	CPA DPKE	KeyGen:	**3784** k	4178 k (-9.4%)	4278 k (-11.5%)	cycles
			23968	26096 (-8.1%)	26096 (-8.1%)	bytes
		Enc:	**261** k	266 k (-1.9%)	299 k (-12.7%)	cycles
			15544	16620 (-6.5%)	17656 (-12%)	bytes
		Dec:	**626** k	708 k (-11.6%)	741 k (-15.5%)	cycles
			16968	19080 (-11.1%)	19080 (-11.1%)	bytes
ntruhps4096821 (Sec.Level:5)	CCA KEM	KeyGen:	**5632** k	6071 k (-7.2%)	6239 k (-9.7%)	cycles
			32272	35208 (-8.3%)	35208 (-8.3%)	bytes
		Encaps:	1020 k	**1011** k ($+0.1\%$)	1067 k (-4.4%)	cycles
			22224	23412 (-5.1%)	25144 (-11.6%)	bytes
		Decaps:	**917** k	1011 k (-9.3%)	1067 k (-14.1%)	cycles
			21360	24280 (-12%)	24280 (-12%)	bytes
	CPA DPKE	KeyGen:	**5549** k	6007 k (-7.6%)	6174 k (-10.1%)	cycles
			28360	31296 (-9.4%)	31296 (-9.4%)	bytes
		Enc:	280 k	**268** k ($+4.5\%$)	324 k (-13.6%)	cycles
			17984	19172 (-6.2%)	20904 (-14%)	bytes
		Dec:	**747** k	845 k (-11.6%)	900 k (-17%)	cycles
			19744	22664 (-12.9%)	22664 (-12.9%)	bytes

Table 7.10: Results of size evaluations

Scheme	Implementation	.text (bytes)
ntruhps2048509	TMVP	**161272**
	NTT	191912 (16%)
	Toom	181128 (11%)
ntruhps2048677	TMVP	**239120**
	NTT	281692 (15.1%)
	Toom	267820 (10.7%)
ntruhrss701	TMVP	**219560**
	NTT	264688 (17%)
	Toom	250024 (12.2%)
ntruhps4096821	TMVP	**315748**
	NTT	370184 (14.7%)
	Toom	346504 (8.9%)

.data and .bss values are not included in the table since they are zero for all parameter sets and implementations.

CHAPTER 8

CONCLUSION

We derive the new TMVP-3 and TMVP-4 formulas from Toom-3 and Toom-4 algorithms using Winograd's technique [43], which require five and seven smaller TMVPs, respectively. Moreover, we propose a TMVP-based algorithm for multiplication in $\mathcal{R}_{2^m+} = \mathbb{Z}_{2^m}[x]/\langle x^n + 1\rangle$ for $n = 256$, and in $\mathcal{R}_{2^m-} = \mathbb{Z}_{2^m}[x]/\langle x^n - 1\rangle$ for $n = 509, 677, 701, 821$ which utilize our TMVP-4 formula. An important note here is the proposed algorithms provide results in the target rings, thus they do not require additional polynomial reduction outside of polynomial multiplication, unlike Karatsuba and Toom-Cook. We implement our algorithms on the ARM Cortex-M4 microcontroller. We improve the efficiency of multiplication in $\mathcal{R}_{q^+}$ and $\mathcal{R}_{q^-}$, and reduce the stack usage compared to all Cortex-M4 implementations that use the Karatsuba, Toom, and schoolbook methods. In some cases, our algorithms are faster and consume less memory than the NTT implementations [32]. We integrate the assembly codes of the proposed multiplication algorithms to existing implementations of Saber, and NTRU [31, 32]. We achieve improvements in stack memory usage and performance of the key generation, encapsulation, and decapsulation algorithms compared to the results of Karatsuba, Toom implementations given in [33] and [38]. Our application to NTRU outperforms the NTT method in [32] for $n = 509, 677, 701$, and reduces stack usage for all parameter sets. Furthermore, our NTRU applications have smaller code sizes than the NTT method for all.

REFERENCES

[1] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*, US Department of Commerce, National Institute of Standards and Technology, Jul 2020.

[2] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, *Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process*, US Department of Commerce, National Institute of Standards and Technology, 2019.

[3] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dand, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*, US Department of Commerce, National Institute of Standards and Technology, Jul 2022.

[4] S. Ali and M. Cenk, Faster residue multiplication modulo 521-bit Mersenne prime and an application to ECC, IEEE Transactions on Circuits and Systems I: Regular Papers, 65(8), pp. 2477–2490, 2018.

[5] E. Alkım, P. Jakubeit, and P. Schwabe, NewHope on ARM Cortex-M, in *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pp. 332–349, Springer, 2016.

[6] N. Aragon, P. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Guneysu, C. A. Melchor, et al., BIKE: bit flipping key encapsulation, 2017.

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

[8] R. Azarderakhsh, M. Campagna, C. Costello, L. Feo, B. Hess, A. Jalali, D. Jao, B. Koziel, B. LaMacchia, P. Longa, et al., Supersingular isogeny key encapsulation, submission to the NIST post-quantum standardization project, 152, pp. 154–155, 2017.

[9] 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.

[10] D. J. Bernstein, T. Chou, T. Lange, I. von Maurich, R. Misoczki, R. Niederhagen, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, et al., Classic McEliece: conservative code-based cryptography, NIST submissions, 2017.

[11] D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. Van Vredendaal, NTRU Prime., IACR Cryptol. ePrint Arch., 2016, p. 461, 2016.

[12] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn, SPHINCS: practical stateless hash-based signatures, in *Annual international conference on the theory and applications of cryptographic techniques*, pp. 368–397, Springer, 2015.

[13] D. J. Bernstein and B.-Y. Yang, Fast constant-time gcd computation and modular inversion, IACR transactions on cryptographic hardware and embedded systems, pp. 340–398, 2019.

[14] J. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila, Frodo: Take off the ring! practical, quantum-secure key exchange from LWE, in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 1006–1018, 2016.

[15] J. W. Bos, S. Friedberger, M. Martinoli, E. Oswald, and M. Stam, Fly, you fool! Faster Frodo for the ARM Cortex-M4., IACR Cryptol. ePrint Arch., 2018, p. 1116, 2018.

[16] A. Casanova, J.-C. Faugere, G. Macario-Rat, J. Patarin, L. Perret, and J. Ryckeghem, *GeMSS: a great multivariate short signature*, Ph.D. thesis, UPMC-Paris 6 Sorbonne Universités; INRIA Paris Research Centre, MAMBA Team, 2017.

[17] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha, Picnic: A family of post-quantum secure digital signature algorithms.

[18] C. Chen, O. Danba, J. Hoffstein, A. Hülsing, J. Rijneveld, J. M. Schanck, P. Schwabe, W. Whyte, and Z. Zhang, NTRU Algorithm Specifications and Supporting Documentation, in *Second PQC Standardization Conference*, 2019, `https://ntru.org/f/ntru-20190330.pdf`.

[19] C.-M. M. Chung, V. Hwang, M. J. Kannwischer, G. Seiler, C.-J. Shih, and B.-Y. Yang, NTT Multiplication for NTT-unfriendly Rings, IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 159–188, 2021.

[20] J. Ding, M.-S. Chen, M. Kannwischer, J. Patarin, A. Petzoldt, D. Schmidt, and B.-Y. Yang, Rainbow. Submission to the NIST Post-Quantum Cryptography Standardization Project (2020).

[21] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, Crystals-dilithium: A lattice-based digital signature scheme, IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 238–268, 2018.

[22] 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.

[23] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, Saber: Mod-LWR based KEM (round 2 submission), 2019, `https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround2.pdf`.

[24] J.-P. D'Anvers, A. Karmakar, F. Vercauteren, and S. S. Roy, SABER: Mod-LWR based KEM, Second PQC Standardization Conference, 2019, University of California, Santa Barbara, USA, 2019.

[25] H. Fan and M. A. Hasan, A new approach to subquadratic space complexity parallel multipliers for extended binary fields, IEEE Transactions on Computers, 56(2), pp. 224–233, 2007.

[26] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, Falcon: Fast-Fourier lattice-based compact signatures over NTRU, Submission to the NIST's post-quantum cryptography standardization process, 36(5), 2018.

[27] 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.

[28] M. A. Hasan and C. Negre, Multiway splitting method for Toeplitz matrix vector product, IEEE Transactions on Computers, 62(7), pp. 1467–1471, 2012.

[29] J. Hoffstein, NTRU: a new high speed public key cryptosystem, presented at the rump session of Crypto 96, 1996.

[30] J. Hoffstein, J. Pipher, and J. H. Silverman, NTRU: A ring-based public key cryptosystem, in *International Algorithmic Number Theory Symposium*, pp. 267–288, Springer, 1998.

[31] M. J. Kannwischer, J. Rijneveld, and P. Schwabe, Faster Multiplication in $\mathbb{Z}_{2^m}[x]$ on Cortex-M4 to Speed up NIST PQC Candidates, in *International Conference on Applied Cryptography and Network Security*, pp. 281–301, Springer, 2019, `https://github.com/mupq/polymul-z2mx-m4`.

[32] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, PQM4: Post-quantum crypto library for the ARM Cortex-M4, `https://github.com/mupq/pqm4`.

[33] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4, 2019.

[34] A. Karmakar, I. Verbauwhede, et al., Saber on ARM CCA-secure module lattice-based key encapsulation on ARM, IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 243–266, 2018, `https://github.com/KULeuven-COSIC/SABER`.

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

[36] C.-L. T. Li, *Implementation of Polynomial Modular Inversion in Latticebased cryptography on ARM*, Master's thesis, National Taiwan University, 2021.

[37] C. A. Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, E. Persichetti, G. Zémor, and I. Bourges, Hamming quasi-cyclic (HQC), NIST PQC Round, 2(4), p. 13, 2018.

[38] 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, `https://github.com/KULeuven-COSIC/TCHES2020_SABER/tree/master/Cortex-M4`.

[39] J.-S. Pan, C.-Y. Lee, A. Sghaier, M. Zeghid, and J. Xie, Novel systolization of subquadratic space complexity multipliers based on toeplitz matrix–vector product approach, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 27(7), pp. 1614–1622, 2019.

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

[41] H. K. Taşkın and M. Cenk, Speeding up Curve25519 using Toeplitz Matrix-vector Multiplication, in *Proceedings of the Fifth Workshop on Cryptography and Security in Computing Systems*, pp. 1–6, 2018.

[42] 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.

[43] S. Winograd, *Arithmetic Complexity of Computations*, Society For Industrial & Applied Mathematics, U.S., 1980.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Keskinkurt Paksoy, İrem
Nationality: Turkish (TC)
Date and Place of Birth: 1982, Bursa
Marital Status: Married

EDUCATION

Degree	**Institution**	**Year of Graduation**
M.S., Cryptography	Middle East Technical University	2017
B.S., Mathematics	Hacettepe University	2005
High School	Anıttepe High School	1999

PROFESSIONAL EXPERIENCE

Year	**Place**	**Enrollment**
2020-	Middle East Technical University	Expert
2005-2015	Various Private Education Institutes	Mathematics Teacher

PUBLICATIONS

International Journal Publications

- İrem Keskinkurt Paksoy, Murat Cenk, *TMVP-based Multiplication for Polynomial Quotient Rings and Application to Saber on ARM Cortex-M4*, submitted to a peer-reviewed journal.

- İrem Keskinkurt Paksoy, Murat Cenk, *Faster NTRU on ARM Cortex-M4 with TMVP-based multiplication*, accepted by IEEE Transactions on Circuits and Systems-I: Regular Papers.

Proceedings/Posters

- Erdem Başer, Timur Hülagü, Ersan Akyıldız, Adnan Bilgen, Murat Cenk, İrem Keskinkurt-Paksoy, A. Sevtap Selcuk-Kestel, *Data Sharing Under Confidentiality*, Ninth Biennial IFC Conference Bulletin, 2019, Basel, Switzerland.

Invited Talks/Presentations

- İrem Keskinkurt Paksoy, *Implementation of Saber using TMVP*, TÜBİTAK BİLGEM & ASELSAN 2nd Post-Quantum Cryptography Workshop, November, 2020, Ankara.
- İrem Keskinkurt Paksoy, *Implementation of NTRU using TMVP-based multiplication*, TÜBİTAK BİLGEM & ASELSAN 3rd Post-Quantum Cryptography Workshop, March, 2022, Ankara.