New TMVP-based Algorithms for Polynomial Quotient Rings and Application to Saber on ARM Cortex-M4*

İrem Keskinkurt Paksoy †
Murat Cenk ‡
Institute of Applied Mathematics, Middle East Technical University

Abstract

Lattice-based finalists of the post-quantum cryptography competition need efficient multiplication in polynomial quotient rings. The fastest multiplication algorithm known is the number theoretic transform (NTT) which requires certain restrictions on parameters. If the parameters of a scheme do not comply with these restrictions, NTT cannot be used directly. In this case, other methods like Toom-Cook, Karatsuba, and the schoolbook methods can be used. NIST also encourages the developement of the non-NTT style of multiplications in the second-round report on the PQC standardization process. This paper introduces fast new three- and four-way Toeplitz matrix-vector product (TMVP) formulas with five and seven multiplications, respectively, for non-NTT multiplications. Then, we use the new four-way TMVP formula to develop a TMVP-based multiplication algorithm for the polynomial quotient ring that Saber uses. We implement the proposed algorithm on ARM Cortex-M4 and compare the results to previous implementations that use a combination of Toom-Cook, Karatsuba, and the schoolbook methods. The TMVP-based multiplication algorithm we propose is 24.5% faster and requires 16.5% less memory, and improves the overall performance of Saber between 3.2% and 11.9% while decreasing stack memory usage between 20.5% and 44.2%. Moreover, Saber's module structure allows us to improve our algorithm by using a modified version of the block recombination method. The improved algorithm we propose speeds up the key generation, encapsulation, and decapsulation algorithms of Saber between 8.7% and 15.6% and consumes less stack memory than state-of-the-art speed optimized implementation using a non-NTT multiplication algorithm.

*The first author is partially supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) 2211-c graduate scholarship program and Council of Higher Education (YÖK) 100/2000 PhD scholarship program.

†irem@metu.edu.tr

‡mcenk@metu.edu.tr

Keywords: ARM Cortex-M4, lattice-based cryptography, polynomial multiplication, post-quantum cryptography, Saber, Toeplitz Matrix-Vector Product (TMVP)

1 Introduction

Since the beginning of the NIST post-quantum standardization competition [31], lattice-based cryptographic schemes have been compelling candidates. In July 2020 NIST announced the third round finalists [30] and three out of four PKE/KEM finalists are lattice-based schemes: CRYSTALS-Kyber [4], NTRU [20, 8], and Saber [12, 13]. These schemes are defined on polynomial rings of the form $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$ where $f(x) \in \mathbb{Z}[x]$ is a degree n polynomial. Multiplication in these rings has a major effect on the efficiency of the schemes. The most efficient polynomial multiplication algorithm that can be used in implementations varies depending on the values of the parameters n and q. The parameters of the schemes such as CRYSTALS-Kyber [4] are convenient for using the Number Theoretic Transform (NTT) [11], which is the most efficient polynomial multiplication algorithm known. The polynomial rings that Saber [12, 13] and NTRU [8, 19, 20] operate on are not NTT-friendly. In such polynomial rings, Toom-Cook [10, 33] and Karatsuba [24] were the most preferred algorithms for multiplication before the NTT method [9] for these rings was proposed. Albeit not commonly used in PQC schemes, Toeplitz matrix-vector product (TMVP) based algorithms are good alternatives for residue polynomial multiplication [34]. Toeplitz matrices appear in various cryptographic applications. For detailed information about the use of Toeplitz matrices in cryptography, we refer to the reader to [26, 27, 15, 6, 7, 1, 32]. TMVP-based algorithms include the polynomial reduction step inside the multiplication process, whereas Karatsuba and Toom-Cook methods require additional reduction. This property makes TMVP-based algorithms more advantageous in some cases, considering the high degree of the modulus polynomials and the number of multiplications performed in schemes.

In this paper, we work on improving non-NTT multiplication in the polynomial ring $\mathcal{R}_q$ with $q = 2^m$ by developing TMVP-based algorithms. The main objective of this work is to speed up the cryptographic schemes that do not originally support NTT, such as Saber and NTRU. Following this purpose, we present three- and four-way Toeplitz matrix-vector product formulas TMVP-3 and TMVP-4 that we derive from Toom-3 and Toom-4 methods, respectively, using the technique explained in [34]. Note that the previous TMVP algorithms in [16, 17, 15, 7] are generally developed for binary extension fields, and they use three multiplications for two-way and six multiplications for three-way methods. Recently, the use of TMVP for prime field multiplication has been proposed in [1] and similar to the binary case, it requires six multiplications for a three-way split. To the best of our knowledge, this is the first time that a three-way formula with five multiplications and a four-way formula with seven multiplications are proposed for residue polynomial multiplication. Moreover, we propose a

TMVP-based algorithm for efficient multiplication in the ring $\mathbb{Z}_{2^m}[x]/\langle x^{256}+1\rangle$ exploiting the TMVP-4 formula. We implement the proposed algorithm on ARM Cortex-M4. The proposed polynomial multiplication algorithm surpasses the one given in [21] that uses a combination of Toom-4, Karatsuba, and schoolbook polynomial multiplication by 24.5%. As a case study, we integrate this algorithm to Saber by replacing the assembly code for polynomial multiplication in an existing implementation with ours. We speed up the key generation, encapsulation, and decapsulation algorithms for all three variants of Saber between 3.2% and 11.9% and reduce stack usage up to 44.2% compared to [29]. Furthermore, by exploiting the module structure of Saber, we apply a technique similar to the block recombination method [17] and improve the performance of our algorithm more. Our improved algorithm achieves speedups between 8.7% and 15.6% compared to [29]. The results show that the proposed algorithms in this paper are the fastest non-NTT multiplication algorithms for Saber on Cortex-M4.

Based on the results from our application to Saber, TMVP-based residue polynomial multiplication algorithms seem to work well for the rings that are not initially NTT-friendly, and they might be good alternatives to Karatsuba and Toom-Cook methods. Apparently, lattice-based cryptosystems are very popular for post-quantum cryptography, and research on the efficiency of lattice-based schemes will continue for a long time. To the best of our knowledge, TMVP-based multiplication algorithms have not been used in post-quantum cryptography applications before. Having non-NTT style options other than Karatsuba and Toom-Cook methods for multiplication in $\mathcal{R}_q$ with $q = 2^m$ would provide diversity in research.

Our Contribution We present new three- and four-way TMVP formulas with five and seven multiplications. These formulas can be used for such multiplications over any ring. The four-way formula yields the fastest non-NTT style of multiplication algorithm for Saber's ring $\mathbb{Z}_{2^m}[x]/\langle x^{256}+1\rangle$ on Cortex-M4. Moreover, we show how we modify and use the block recombination method in [17] for Saber and achieve further improvements on the performance.

Code Availability The source codes of our applications to Saber are publicly available at `https://github.com/iremkp/Saber-tmvp4-m4`.

Organization of the Paper The remainder of this paper is organized as follows. In Section 2, we provide preliminary information and describe the notations we use throughout this paper. In Section 3, we explain the derivation of the new TMVP-3 and TMVP-4 formulas in detail, and introduce the algorithm we propose for multiplication in $\mathbb{Z}_{2^m}[x]/\langle x^{256}+1\rangle$. Moreover, in Section 3 we introduce our modified version of the block recombination method and how we use it to improve our algorithm. We also present the results of the implementation of our algorithm on ARM Cortex-M4 and the applications to Saber in Section 3. Finally, Section **??** summarizes our work and concludes the paper with future study ideas.

2 Preliminaries

This section introduces some definitions and properties to build a background. Throughout the paper, we use the notation $M_{ALG}(n)$ to state the arithmetic complexity of the algorithm ALG for dimension n, and $M(n)$ to state the arithmetic complexity of the most efficient algorithm for the computation in question for dimension n. The ring of polynomials modulo $x^n + 1$ is denoted by $\mathcal{R} = \mathbb{Z}[x]/\langle x^n + 1\rangle$ when the coefficients are integers, and is denoted by $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ when the coefficients of the polynomials are integers from $[0, q)$. The multiplication of an $n \times n$ matrix by an $n \times 1$ vector is referred to as an n-dimensional matrix-vector multiplication. We denote the componentwise multiplication of two vectors V_1 and V_2 of the same length by $V_1 \circ V_2$.

2.1 Toeplitz Matrix Vector Product

There are many cryptographic applications that utilize Toeplitz matrix-vector product (TMVP) in the literature. The use of TMVP in cryptographic computations first appeared in [15] for multiplying elements of binary extension fields. Many proposals were then suggested [1, 15, 17, 18, 32]. Recently, in [1] and [32], 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 be used to calculate the product of two polynomials modulo a polynomial as explained in [34].

2.1.1 Toeplitz Matrix

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 $i = 2, \ldots, m$ and $j = 2, \ldots n$.

Throughout this paper, we focus only on square Toeplitz matrices:

$$T = \begin{pmatrix} t_0 & t'_1 & t'_2 & \cdots & \cdots & \cdots & t'_{n\text{-}1} \\ t_1 & t_0 & t'_1 & t'_2 & \cdots & \cdots & \vdots \\ t_2 & t_1 & t_0 & t'_1 & \ddots & & \vdots \\ \vdots & t_2 & t_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & t'_1 & t'_2 \\ \vdots & & \ddots & \ddots & t_1 & t_0 & t'_1 \\ t_{n\text{-}1} & \cdots & \cdots & \cdots & t_2 & t_1 & t_0 \end{pmatrix}. \tag{1}$$

The matrix T in (1) shows the special form of an $n \times n$ Toeplitz matrix. Clearly, specifying only $2n-1$ of its elements would suffice to identify T. Therefore, the addition of two Toeplitz matrices requires only $2n - 1$ additions, while two regular matrices require n^2. Moreover, every submatrix of a Toeplitz matrix is also a Toeplitz matrix. These properties become very handy 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 n. Suppose we want to compute the product of the Toeplitz matrix T in (1) by a vector V where the transpose of V is $V^T = (v_0, v_1, \ldots, v_n)$. We may apply different splitting methods [18] to compute the following TMVP:

$$T.V = \begin{pmatrix} t_0 & t'_1 & \ldots & t'_{n\text{-}1} \\ t_1 & t_0 & \ldots & t'_{n\text{-}2} \\ \vdots & \vdots & \ddots & \vdots \\ t_{n\text{-}2} & t_{n\text{-}3} & \ldots & t'_1 \\ t_{n\text{-}1} & t_{n\text{-}2} & \ldots & t_0 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n\text{-}2} \\ v_{n\text{-}1} \end{pmatrix}. \tag{2}$$

For example, a two-way TMVP formula allows us to compute an n dimensional TMVP via three $n/2$ dimensional TMVPs. For this, we use the half size partitions T_0, T_1, T_2 of T, and V_0, V_1 of V. The n dimensional Toeplitz matrix-vector product $T.V$ can be calculated as follows:

$$T.V = \begin{pmatrix} T_1 & T_0 \\ T_2 & T_1 \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \end{pmatrix} = \begin{pmatrix} P_0 + P_1 \\ P_0 - P_2 \end{pmatrix}, \tag{3}$$

where

$$\begin{aligned} P_0 &= T_1(V_0 + V_1), \\ P_1 &= (T_0 - T_1)V_1, \\ P_2 &= (T_1 - T_2)V_0. \end{aligned}$$

We refer to the formula in (3) as TMVP$_2$. As any other TMVP split formulas, TMVP$_2$ has four independent phases. For TMVP$_2$ formula given in (3), these phases and operations that corresponds to them are as follows:

- Matrix evaluation of $T = (T_1,\, T_0 - T_1,\, T_1 - T_2)$
- Vector evaluation of $V = (V_0 + V_1, V_1, V_0)$,
- Component multiplication:

$$\begin{aligned} &(P_0, P_1, P_2) \\ =&(T_1, T_0 - T_1, T_1 - T_2) \circ (V_0 + V_1, V_1, V_0) \\ =&(T_1(V_0 + V_1), (T_0 - T_1)V_1, (T_1 - T_2)V_0) \end{aligned}$$

- Recombination of $P_0, P_1, P_2 = (P_0 + P_1, P_0 - P_2)$

The arithmetic complexity of the TMVP$_2$ formula is $M_{\text{TMVP}_2}(n) = 3M(n/2) + 3n - 1$. Similarly, a three-way TMVP formula over integers [32] allows us to compute an n dimensional TMVP via six $n/3$ dimensional TMVPs as follows:

$$\begin{pmatrix} T_2 & T_1 & T_0 \\ T_3 & T_2 & T_1 \\ T_4 & T_3 & T_2 \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \\ V_2 \end{pmatrix} = \begin{pmatrix} P_1 + P_4 + P_5 \\ P_2 - P_4 + P_6 \\ P_3 - P_5 - P_6 \end{pmatrix}, \tag{4}$$

where

$$\begin{aligned} P_1 &= (T_0 + T_1 + T_2)V_2, \\ P_2 &= (T_1 + T_2 + T_3)V_1, \\ P_3 &= (T_2 + T_3 + T_4)V_0, \\ P_4 &= T_1(V_1 - V_2), \\ P_5 &= T_2(V_0 - V_2), \\ P_6 &= T_3(V_0 - V_1). \end{aligned}$$

In this formula T_0, T_1, T_2, T_3, T_4 are $\frac{n}{3} \times \frac{n}{3}$ Toeplitz matrices and V_0, V_1, V_2 are $\frac{n}{3} \times 1$ vectors. The arithmetic complexity of the given three-way formula above which we denote by TMVP$_3$ is $M_{\text{TMVP}_3}(n) = 6M(n/3) + 5n - 1$.

The choice of the formula for efficiently calculating a TMVP differs depending on the size of the Toeplitz matrix. Different split formulas for TMVPs can be derived from any given polynomial multiplication algorithm. The technique of derivation is explained elaborately in [34]. We use the same technique in [34] to derive a three-way TMVP formula from Toom-3 algorithm and a four-way TMVP formula from Toom-4 algorithm. These three- and four-way formulas require five and seven multiplications, respectively. These formulas will be denoted by TMVP-3 and TMVP-4. We explain the derivation technique in detail and present the formulas in Section 3.

2.2 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

$$c'_i = \sum_{\substack{j+k=i \\ 0 \le j,k < n}} a_j b_k. \tag{5}$$

The product $c(x) = \sum_{i=0}^{n-1} c_i x^i$ of the polynomials $a(x)$ and $b(x)$ in $\mathcal{R}$ 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 $c_{n-1} = c'_{n-1}$. Expanding these equalities according to (5) shows that the coefficients of the polynomial $c(x)$ can be calculated via the following TMVP:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n\text{-}2} \\ c_{n\text{-}1} \end{pmatrix} = \begin{pmatrix} a_0 & \mp a_{n\text{-}1} & \dots & \mp a_2 & \mp a_1 \\ a_1 & a_0 & \dots & \mp a_3 & \mp a_2 \\ a_2 & a_1 & \dots & \mp a_4 & \mp a_3 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n\text{-}2} & a_{n\text{-}3} & \dots & a_0 & \mp a_{n\text{-}1} \\ a_{n\text{-}1} & a_{n\text{-}2} & \dots & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n\text{-}2} \\ b_{n\text{-}1} \end{pmatrix}. \tag{6}$$

Multiplication in $\mathcal{R}_q$ can be represented in the same way by assuming the coefficients $a_j, b_k \in \mathbb{Z}_q$, and arithmetic operations include reduction modulo q. The Toeplitz matrix in (6) has a more special form than the one in (2). It contains only n different entries, which are the coefficients of one of the multiplicand polynomials. Having n components instead of $2n-1$ means we can use more simplified versions of the TMVP formulas. Therefore, we can calculate this type of TMVPs even more efficiently. To be more specific, 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 and memory usage.

It should also be noted that TMVP-based multiplications have lower delay complexities than Karatsuba-like and Toom-Cook multiplications. The delay complexity comparison of Karatsuba-like algorithms with TMVP-based algorithms for binary extension fields can be seen in [7]. Therefore, using TMVP-based multiplications in hardware implementations is expected to lead to better results than Karatsuba-like and Toom-Cook multiplications.

2.3 Saber

Saber [12, 13] is a lattice-based key encapsulation mechanism (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 [3, 28]. Saber defines an IND-CPA secure public-key encryption scheme (Saber.PKE) consisting of key generation (Saber.PKE.Keygen), encryption (Saber.PKE.Enc), decryption (Saber.PKE.Dec) algorithms. It uses a version of the Fujisaki-Okamoto transformation to have an IND-CCA secure key encapsulation mechanism (Saber.KEM), which also consists of three algorithms key generation (Saber.KEM.KeyGen), encapsulation (Saber.KEM.Encaps), decapsulation (Saber.KEM.Decaps). The details of all algorithms of Saber.PKE and Saber.KEM can be found in [14].

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), and $\ell = 4$ (FireSaber) provide level 1, level 3, and level 5 security, respectively. Arithmetic operations of Saber are performed in $\mathcal{R}_q = \mathcal{R}_{2^{13}} = \mathbb{Z}_{2^{13}}[x]/\langle x^{256}+1\rangle$ 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. The rings $\mathcal{R}_q$ and $\mathcal{R}_p$ that Saber is defined on are not suitable for using the Number Theoretic Transform (NTT) directly, which is the most efficient polynomial multiplication algorithm known. Regardless of the platform, using a combination of Toom-Cook, Karatsuba, and schoolbook methods for efficient polynomial multiplication together with a polynomial reduction is the most common method when NTT is out of the question. Details of existing implementations that use Toom-Cook and Karatsuba algorithms on different platforms can be found in [21, 25, 29].

2.4 Block Recombination

TMVP formulas can be used recursively or iteratively. The recursive use of two- and three-way TMVP formulas in binary fields are given in [16]. Following this work, a new method called block recombination is proposed in [17]. For simplicity, we explain this method on the 4-dimensional TMVP (7), using TMVP$_2$ given in (3). Since we do not work in binary fields in this work, we assume (7) is defined on integers.

$$\begin{pmatrix} t_3 & t_2 & t_1 & t_0 \\ t_4 & t_3 & t_2 & t_1 \\ t_5 & t_4 & t_3 & t_2 \\ t_6 & t_5 & t_4 & t_3 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \tag{7}$$

In [17], decomposition of a TMVP formula is defined by four independent recursive calculation steps.

- *Component Matrix Formation (CMF)*: Corresponds to the recursive matrix evaluation of the Toeplitz matrix. For (7), the first step of recursion on T computes $CMF(T) = (T_1, T_0 - T_1, T_1 - T_2)$, where

  $$T_0 = \begin{pmatrix} t_1 & t_0 \\ t_2 & t_1 \end{pmatrix}, T_1 = \begin{pmatrix} t_3 & t_2 \\ t_4 & t_3 \end{pmatrix}, T_2 = \begin{pmatrix} t_5 & t_4 \\ t_6 & t_5 \end{pmatrix}.$$

  The second step computes

  $$CMF(T) = (CMF(T_1), CMF(T_0 - T_1), CMF(T_1 - T_2)).$$

  Thus, the vector

  $$\begin{aligned} CMF(T) = (&t_3, t_2 - t_3, t_3 - t_4, \\ &t_1 - t_3, t_0 - t_1 - t_2 + t_3, \\ &t_1 - t_2 - t_3 + t_4, \\ &t_3 - t_5, t_2 - t_3 - t_4 + t_5, \\ &t_3 - t_4 - t_5 + t_6). \end{aligned}$$

  of length nine is the component matrix formation of T.

- *Component Vector Formation (CVF)*: Corresponds to the recursive vector evaluation of V in (7). The first step of recursion on V computes $CVF(V) = (V_0 + V_1, V_1, V_0)$, and the second step computes $CVF(V) = (CVF(V_0 + V_1), CVF(V_1), CVF(V_0))$. Thus, the vector

$$\begin{aligned} CVF(V) = (&v_0 + v_1 + v_2 + v_3, v_1 + v_3, \\ &v_0 + v_2, v_2 + v_3, v_3, \\ &v_2, v_0 + v_1, v_1, v_0,). \end{aligned}$$

  of length nine is the component vector formation of V.

- *Component Multiplication (CM)*: Corresponds to componentwise multiplication of $CMF(T)$ and $CVF(V)$ which is the vector

$$\begin{aligned} &CMF(T) \circ CVF(V) = (t_3(v_0 + v_1 + v_2 + v_3), \\ &(t_2 - t_3)(v_1 + v_3), (t_3 - t_4)(v_0 + v_2), \\ &(t_1 - t_3)(v_2 + v_3), (t_0 - t_1 - t_2 + t_3)(v_3),) \\ &(t_1 - t_2 - t_3 + t_4)v_2, (t_3 - t_5)(v_0 + v_1), \\ &(t_2 - t_3 - t_4 + t_5)v_1, (t_3 - t_4 - t_5 + t_6)v_0) \\ &= (\underbrace{s_0, s_1, s_2}_{P_0}, \underbrace{s_3, s_4, s_5}_{P_1}, \underbrace{s_6, s_7, s_8}_{P_2}) \end{aligned}$$

  of length nine.

- *Reconstruction (R)*: Corresponds to the recombination of the product from the component multiplication of $CMF(T)$ and $CVF(V)$. Let $P = CMF(T) \circ CVF(V)$ and split $P = [P_0, P_1, P_2]$ into three equal length vectors. Then we have $R(P) = (R(P_0) + R(P_1), R(P_0) - R(P_2))$. Thus,

$$\begin{aligned} R(P) =& (y_0, y_1, y_2, y_3) \\ =& (s_0 + s_1 + s_4 + s_5, \\ & s_0 - s_2 + s_4 - s_6, \\ & s_0 + s_1 - s_6 - s_7, \\ & s_0 - s_2 - s_6 + s_8) \end{aligned}$$

In [17], they propose a technique using this decomposition to reduce the number of operations in the reconstruction step for the sum of two or more TMVPs, such as $T.B + T'.B'$.

After component multiplication step of $T.B$ and $T'.B'$ they add $CMF(T) \circ CVF(B)$ and $CMF(T') \circ CVF(B')$ componentwise before performing reconstruction step. Therefore, the computation is done with only one reconstruction step instead of performing for each multiplication.

2.5 Implementation Platform: ARM Cortex-M4

We choose 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 registers, and aside from Program Counter (PC) and Stack Pointer (SP) registers, they are all available for development. We use the STM32F4DISCOVERY development board, which is used in many implementations of PQC candidates [2, 5, 21, 22, 25]. 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 16-bit halfwords of multiple registers in one cycle. These instructions allow us to implement the schoolbook matrix-vector multiplication for small dimensions efficiently.

3 Our Work

We derive the new TMVP-3 and TMVP-4 formulas from Toom-3 and Toom-4 algorithms using the same technique given in [34], which require five and seven smaller TMVPs, respectively. We also propose a TMVP-based algorithm for multiplication in the ring $\mathcal{R}_{2^m} = \mathbb{Z}_{2^m}[x]/\langle x^{256}+1\rangle$ which utilizes our TMVP-4 formula. An important note here is that the proposed algorithm includes the polynomial reduction step and does not require additional polynomial reduction outside of polynomial multiplication, unlike Karatsuba and Toom-Cook. We implement our algorithm on the ARM Cortex-M4 microcontroller. We improve the efficiency of multiplication in $\mathcal{R}_q$ and reduce the stack usage compared to [21]. We integrate the assembly code of the proposed multiplication algorithm into an existing implementation of Saber [22]. Moreover, we optimize our multiplication algorithm and propose a non-NTT style of multiplication algorithm that is unique to Saber. We achieve improvements in the efficiency of the key generation, encapsulation, and decapsulation algorithms compared to the results given in [23] and [29]. Our applications to Saber require less stack space in some cases.

This section goes into detail about the derivation of our new three- and four-way TMVP formulas, describes the proposed multiplication algorithm, and explains how we modify the block recombination method to improve our algorithm for Saber. The benchmarking results for the applications of our algorithms to Saber are also presented in this section.

3.1 A new three-way TMVP formula with five multiplications (TMVP-3)

Let $v(x) = v_0 + v_1x + v_2x^2$ and $w(x) = w_0 + w_1x + w_2x^2$ be two polynomials in $\mathbb{Z}[x]$. The product of these polynomials $v(x)w(x) = k(x) = k_0 + k_1x + k_2x^2 + k_3x^3 + k_4x^4$ can be calculated using different methods. The coefficients of the product polynomial $k(x)$ are computed using the schoolbook method as follows:

$$\begin{aligned} k_0 &= v_0w_0, \\ k_1 &= v_0w_1 + v_1w_0, \\ k_2 &= v_0w_2 + v_1w_1 + v_2w_0, \\ k_3 &= v_1w_2 + v_2w_1, \\ k_4 &= v_2w_2. \end{aligned} \tag{8}$$

This computation requires nine multiplications and four additions. On the other hand, the evaluations of $k(x)$ at the points $\{0, 1, -1, -2, \infty\}$ leads to the following:

$$\begin{aligned} k(0) &= v(0)w(0) = v_0w_0, \\ k(1) &= v(1)w(1) = (v_0 + v_1 + v_2)(w_0 + w_1 + w_2), \\ k(\text{-}1) &= v(\text{-}1)w(\text{-}1) = (v_0 - v_1 + v_2)(w_0 - w_1 + w_2), \\ k(\text{-}2) &= v(\text{-}2)w(\text{-}2) = (v_0 - 2v_1 + 4v_2)(w_0 - 2w_1 + 4w_2), \\ k(\infty) &= v(\infty)w(\infty) = v_2w_2. \end{aligned}$$

The coefficients k_i of the product polynomial $k(x)$ are interpolated and the following equalities are obtained:

$$\begin{aligned} k_0 &= k(0), \\ k_1 &= k(0)/2 + k(1)/3 - k(\text{-}1) + k(\text{-}2)/6 - 2k(\infty), \\ k_2 &= -k(0) + k(1)/2 + k(\text{-}1)/2 - k(\infty), \\ k_3 &= -k(0)/2 + k(1)/6 + k(\text{-}1)/2 - k(\text{-}2)/6 + 2k(\infty), \\ k_4 &= k(\infty). \end{aligned} \tag{9}$$

This method is known as Toom-3. Now, we derive a three-way TMVP formula from Toom-3. To this end, we multiply each equation of both (8) and (9) that corresponds to k_i by a symbolic variable t_{4-i} for $i = 0, \ldots, 4$ and we take the sum of all equations. Then, we rearrange the terms to obtain two equations of the form $t_4k_0+t_3k_1+t_2k_2+t_1k_3+t_0k_4 = y_2w_0+y_1w_1+y_0w_2$. From (8) we get

$$\begin{aligned} y_0 &= t_2v_0 + t_1v_1 + t_0v_2, \\ y_1 &= t_3v_0 + t_2v_1 + t_1v_2, \\ y_2 &= t_4v_0 + t_3v_1 + t_2v_2, \end{aligned} \tag{10}$$

and from (9) we get

$$\begin{aligned}
y_0 =& \frac{1}{6}(v_0+v_1+v_2)\left(2t_3+3t_2+t_1\right) \\
&+\frac{1}{2}(v_0-v_1+v_2)\left(-2t_3+t_2+t_1\right) \\
&+\frac{2}{3}\left(t_3-t_1\right)\left(v_0-2v_1+4v_2\right) \\
&+\left(-2t_3-t_2+2t_1+t_0\right)v_2, \\
y_1 =& \frac{1}{6}\left(2t_3+3t_2+t_1\right)\left(v_0+v_1+v_2\right) \\
&-\frac{1}{2}\left(-2t_3+t_2+t_1\right)\left(v_0-v_1+v_2\right) \\
&-\frac{1}{3}\left(t_3-t_1\right)\left(v_0-2v_1+4v_2\right), \\
y_2 =& \frac{1}{2}\left(2t_4+t_3-2t_2-t_1\right)v_0 \\
&+\frac{1}{6}\left(2t_3+3t_2+t_1\right)\left(v_0+v_1+v_2\right) \\
&+\frac{1}{2}\left(-2t_3+t_2+t_1\right)\left(v_0-v_1+v_2\right) \\
&+\frac{1}{6}\left(t_3-t_1\right)\left(v_0-2v_1+4v_2\right).
\end{aligned} \tag{11}$$

Clearly, we may express the vector y as a TMVP using (10) where $y^T = (y_0, y_1, y_2)$, which gives the left-hand side of equation (12), whereas the right-hand side comes from (11). Finally, we have the following formula, which we refer to as TMVP-3:

$$\begin{pmatrix} t_2 & t_1 & t_0 \\ t_3 & t_2 & t_1 \\ t_4 & t_3 & t_2 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} P_1+P_2+4P_3+P_4 \\ P_1-P_2-2P_3 \\ P_0+P_1+P_2+P_3 \end{pmatrix} \tag{12}$$

where

$$\begin{aligned}
P_0 &= \frac{\left(2t_4+t_3-2t_2-t_1\right)v_0}{2}, \\
P_1 &= \frac{\left(2t_3+3t_2+t_1\right)\left(v_0+v_1+v_2\right)}{6}, \\
P_2 &= \frac{\left(-2t_3+t_2+t_1\right)\left(v_0-v_1+v_2\right)}{2}, \\
P_3 &= \frac{\left(t_3-t_1\right)\left(v_0-2v_1+4v_2\right)}{6}, \\
P_4 &= \left(-2t_3-t_2+2t_1+t_0\right)v_2.
\end{aligned}$$

Therefore, with this formulation, an n dimensional TMVP can be calculated via five smaller TMVPs of dimension $n/3$. In [34, 15], three-way formulas for TMVP which requires six TMVPs of dimension $n/3$ can be seen.

Algorithm 1 Matrix and Vector Evaluations of TMVP-3

```
1: $S_1 = v_0 + v_2$
2: $S_2 = S_1 + v_1$
3: $S_3 = S_1 - v_1$
4: $S_4 = v_0 - 2v_1$
5: $S_5 = S_4 + 4v_2$
6: $S_6 = t_3 - t_1$
7: $S_7 = t_1 + t_2$
8: $S_8 = t_4 - t_2$
9: $S_9 = S_8 + S_6/2$
10: $S_{10} = S_6/3 + S_7/2$
11: $S_{11} = S_7/2 - t_3$
12: $S_{12} = -2S_6 - t_2$
13: $S_{13} = S_{12} + t_0$
14: $S_{14} = S_6/6$
```

For the evaluation of matrices and the vectors needed to compute P_i, we use Algorithm 1. Steps 1-5 perform additions of two vectors of length $n/3$; hence $5n/3$ additions are required in total. The remaining additions in the algorithm are of two Toeplitz matrices of dimension $n/3 \times n/3$, which requires $2n/3 - 1$ additions for each. So, for steps $6 - 13$, we need $16n/3 - 8$ additions. Therefore, the evaluation given in Algorithm 1 is completed via $7n - 8$ additions.

Algorithm 2 Recombination of P_i for TMVP-3

```
1: $S_1 = P_1 + P_2$
2: $S_2 = P_1 - P_2$
3: $S_3 = S_1 + 4P_3$
4: $S_4 = S_3 + P_4$
5: $S_5 = S_2 - 2P_3$
6: $S_6 = P_0 + P_3$
7: $S_7 = S_1 + S_6$
```

For recombination of P_i, we use Algorithm 2. Since every step of Algorithm 2 requires $n/3$ additions, recombination is done via $7n/3$ additions in total. Thus, the arithmetic complexity of the new TMVP-3 formula is roughly $M_{\text{TMVP-3}}(n) = 5M(n/3) + 28n/3 - 8$. We omit the scalar multiplication operations while calculating the computational complexity of the algorithm.

3.2 A new four-way TMVP formula with seven multiplications (TMVP-4)

Let $v(x) = v_0 + v_1x + v_2x^2 + v_3x^3$ and $w(x) = w_0 + w_1x + w_2x^2 + w_3x^3$ be two polynomials in $\mathbb{Z}[x]$. The product of these polynomials $v(x)w(x) = k(x) = k_0 + k_1x + k_2x^2 + k_3x^3 + k_4x^4 + k_5x^5 + k_6x^6$ can be calculated using different methods. The coefficients k_i of the product polynomial $k(x)$ are computed using the schoolbook method as follows:

$$\begin{aligned}
k_0 &= v_0w_0,\\
k_1 &= v_0w_1 + v_1w_0,\\
k_2 &= v_0w_2 + v_1w_1 + v_2w_0,\\
k_3 &= v_0w_3 + v_1w_2 + v_2w_1 + v_3w_0,\\
k_4 &= v_1w_3 + v_2w_2 + v_3w_1,\\
k_5 &= v_2w_3 + v_3w_2,\\
k_6 &= v_3w_3.
\end{aligned} \tag{13}$$

This computation requires sixteen multiplications and nine additions. On the other hand, the evaluation of the product polynomial $k(x)$ at the points $\{0, 1, -1, 2, -2, 3, \infty\}$ leads to the following:

$$\begin{aligned}
k(0) &= v_0w_0,\\
k(1) &= (v_0 + v_1 + v_2 + v_3)(w_0 + w_1 + w_2 + w_3),\\
k(\text{-}1) &= (v_0 - v_1 + v_2 - v_3)(w_0 - w_1 + w_2 - w_3),\\
k(2) &= (v_0 + 2v_1 + 4v_2 + 8v_3)(w_0 + 2w_1 + 4w_2 + 8w_3),\\
k(\text{-}2) &= (v_0 - 2v_1 + 4v_2 - 8v_3)(w_0 - 2w_1 + 4w_2 - 8w_3),\\
k(3) &= (v_0 + 3v_1 + 9v_2 + 27v_3)(w_0 + 3w_1 + 9w_2 + 27w_3),\\
k(\infty) &= v_3w_3.
\end{aligned}$$

Interpolating the coefficients $k_i, i = 0, \ldots, 6$ of the product polynomial gives us the following equations:

$$
\begin{aligned}
k_0 &= k(0),\\
k_1 &= -k(0)/3 + k(1) - k(\text{-}1)/2 - k(2)/4\\
&\quad + k(\text{-}2)/20 + k(3)/30 - 12k(\infty),\\
k_2 &= -5k(0)/4 + 2k(1)/3 + 2k(\text{-}1)/3 - k(2)/24\\
&\quad - k(\text{-}2)/24 + 4k(\infty),\\
k_3 &= 5k(0)/12 - 7k(1)/12 - k(\text{-}1)/24 + 7k(2)/24\\
&\quad - k(\text{-}2)/24 - k(3)/24 + 15k(\infty),\\
k_4 &= k(0)/4 - k(1)/6 - k(\text{-}1)/6 + k(2)/24\\
&\quad + k(\text{-}2)/24 - 5k(\infty),\\
k_5 &= -k(0)/12 + k(1)/12 + k(\text{-}1)/24 - k(2)/24\\
&\quad - k(\text{-}2)/120 + k(3)/120 - 3k(\infty),\\
k_6 &= k(\infty).
\end{aligned}
\tag{14}
$$

This method is known as Toom-4. To derive a four-way TMVP formula, first we multiply each equation of both (13) and (14) that corresponds to k_i by a symbolic variable t_{6-i} for $i = 0, \dots, 6$. Then, we take the sum of all equations to obtain two equations of the form $t_6k_0 + t_5k_1 + t_4k_2 + t_3k_3 + t_2k_4 + t_1k_5 + t_0k_6 = y_3w_0 + y_2w_1 + y_1w_2 + y_0w_3$. From (13) we get

$$
\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} t_3 & t_2 & t_1 & t_0 \\ t_4 & t_3 & t_2 & t_1 \\ t_5 & t_4 & t_3 & t_2 \\ t_6 & t_5 & t_4 & t_3 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix}
\tag{15}
$$

and from (14) we get

$$
\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} P_1 - P_2 + 8P_3 - 8P_4 + 27P_5 + P_6 \\ P_1 + P_2 + 4P_3 + 4P_4 + 9P_5 \\ P_1 - P_2 + 2P_3 - 2P_4 + 3P_5 \\ P_0 + P_1 + P_2 + P_3 + P_4 + P_5 \end{pmatrix}
\tag{16}
$$

where

$$\begin{aligned}
P_0 &= \frac{(12t_6 - 4t_5 - 15t_4 + 5t_3 + 3t_2 - t_1)\, v_0}{12},\\
P_1 &= \frac{(12t_5 + 8t_4 - 7t_3 - 2t_2 + t_1)\,(v_0 + v_1 + v_2 + v_3)}{12},\\
P_2 &= \frac{(-12t_5 + 16t_4 - t_3 - 4t_2 + t_1)\,(v_0 - v_1 + v_2 - v_3)}{24},\\
P_3 &= \frac{(-6t_5 - t_4 + 7t_3 + t_2 - t_1)\,(v_0 + 2v_1 + 4v_2 + 8v_3)}{24},\\
P_4 &= \frac{(6t_5 - 5t_4 - 5t_3 + 5t_2 - t_1)\,(v_0 - 2v_1 + 4v_2 - 8v_3)}{120},\\
P_5 &= \frac{(4t_5 - 5t_3 + t_1)\,(v_0 + 3v_1 + 9v_2 + 27v_3)}{120},\\
P_6 &= (-12t_5 + 4t_4 + 15t_3 - 5t_2 - 3t_1 + t_0)\, v_3.
\end{aligned}$$

The equality of the right hand side of the equations (15) and (16) gives the new TMVP-4 formula. Therefore with this formulation, an n dimensional TMVP can be calculated via seven smaller TMVPs whose sizes are 1/4-th of the original one.

For the matrix and vector evaluations needed to compute P_i we use Algorithm 3. Steps 1-11 are for vector evaluations and perform vector additions of length $n/4$; hence $11n/4$ additions are required for these steps. The remaining steps of the algorithm performs $n/4$-dimensional Toeplitz matrix additions, which of each require $n/2 - 1$ addition. Therefore, steps 12-32 require $21n/2 - 21$ additions. The matrix and vector evaluations are done by performing $53n/4 - 21$ additions. For the recombination of TMVP-4, we use Algorithm 4 that performs vector additions oflength $n/4$ in each step; hence $13n/4$ additions in total. Therefore, the arithmetic complexity of TMVP-4 formula is roughly $M_{\text{TMVP-4}}(n) = 7M(n/4) + 33n/2 - 21$. We ignore scalar multiplication and shifting operations in arithmetic complexity calculations.

Algorithm 3 Matrix and Vector Evaluations of TMVP-4

```
1: $S_1 = v_0 + v_2$
2: $S_2 = v_1 + v_3$
3: $S_3 = S_1 + S_2$
4: $S_4 = S_1 - S_2$
5: $S_5 = S_1 + 3v_2$
6: $S_6 = 2(S_2 + 3v_3)$
7: $S_7 = S_5 + S_6$
8: $S_8 = S_5 - S_6$
9: $S_9 = S_5 + 5v_2$
10: $S_{10} = 3(S_1 + s_6)$
11: $S_{11} = S_9 + S_{10}$
12: $S_{12} = 4t_5 - 5t_3$
13: $S_{13} = S_{12} + t_1$
14: $S_{14} = t_4 - t_2$
15: $S_{15} = S_{13} - 5S_{14}$
16: $S_{16} = S_{15} + 2t_5$
17: $S_{17} = S_{14} + 4t_4$
18: $S_{18} = 12t_6 - 3S_{17}$
19: $S_{19} = S_{18} - S_{13}$
20: $S_{20} = t_1 + 6t_5$
21: $S_{21} = t_2 - t_0$
22: $S_{22} = 4S_{14} - S_{21}$
23: $S_{23} = S_{22} - 3S_{13}$
24: $S_{24} = 4t_5 - t_3$
25: $S_{25} = S_{13} + 2S_{24}$
26: $S_{26} = 4t_4 - t_2$
27: $S_{27} = S_{25} + 2S_{26}$
28: $S_{28} = t_5 - t_3$
29: $S_{29} = S_{13} + 2S_{28}$
30: $S_{30} = -S_{29} - S_{14}$
31: $S_{31} = S_{13} - 4S_{24}$
32: $S_{32} = S_{31} + 4S_{26}$
```

Algorithm 4 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: $S_{13} = S_{12} + 27P_5$
```

3.3 A new TMVP-based algorithm for multiplication in $\mathbb{Z}_{2^m}[x]/\langle x^{256}+1\rangle$

As explained in Section 2.2, 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 polynomial multiplication. We utilize the new TMVP-4 formula proposed in Section 3.2 to develop an efficient residue polynomial multiplication algorithm.

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 following TMVP:

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

Applying a four-way split to this TMVP, we get the equation (17).

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} A_0 & -A_3 & -A_2 & -A_1 \\ A_1 & A_0 & -A_3 & -A_2 \\ A_2 & A_1 & A_0 & -A_3 \\ A_3 & A_2 & A_1 & A_0 \end{pmatrix} \begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix} \tag{17}$$

The Toeplitz matrix in (17) has a more special form than the Toeplitz matrix in (15). The computation of P_i in the new TMVP-4 formula become even more simpler for (17) because the Toeplitz matrix has only four variables A_0, A_1, A_2, A_3 instead of seven variables $t_0, t_1, t_2, t_3, t_4, t_5, t_6$. If we rewrite the P_i evaluations of (16) for this case, we get the following:

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

where the partitions A_i are regular 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 $4n = 1024$.

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 use the same strategy in [21] and 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 masking the most significant $16 - m$ 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. This may cause a loss in the most significant bits. 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 (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 3.5, we explain how we determine 16 as the threshold value for ARM Cortex-M4 and give the results of the application to Saber.

3.4 Further Improvement 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 5 of Algorithm 1 in [14]) and inner products of two vectors (e.g., $\boldsymbol{b^T s'}$ in line 6 of Algorithm 2 in [14]) 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 (18) is used in key generation, encapsulation and decapsulation algorithms of SABER.KEM, where a_{ij} and s_j are polynomials in $\mathbb{Z}_{2^{13}}[x]/\langle x^{256}+1\rangle$.

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \end{pmatrix} = \begin{pmatrix} a_{00}s_0 + a_{01}s_1 \\ a_{10}s_0 + a_{11}s_1 \end{pmatrix} \tag{18}$$

From Section 2.2, 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 $A_{ij}B_j$ where A_{ij} is the Toeplitz matrix formed with the coefficients of the polynomial a_{ij}, and B_j is the vector representation of the coefficients of the polynomial b_j. So, the right hand side of (18) is equivalent to (19).

$$\begin{pmatrix} A_{00}S_0 + A_{01}S_1 \\ A_{10}S_0 + A_{11}S_1 \end{pmatrix} \tag{19}$$

where $A_{ij}S_j$ is a 256-dimensional TMVPs of the form (17). In the generic TMVP-based algorithm for multiplication in $\mathbb{Z}_{2^{13}}[x]/\langle x^{256}+1\rangle$ we propose in Section 3.3, 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 optimization, 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 (19). CVF_{422} denotes the component vector formation step corresponds to the vector evaluation of a layer formula (17) followed by two layers of formula (3) and is defined as follows:

$$\begin{aligned}
CVF_{422}(V) = (&CVF_{22}(V_0),\\
&CVF_{22}(V_0 + V_1 + V_2 + V_3),\\
&CVF_{22}(V_0 - V_1 + V_2 - V_3),\\
&CVF_{22}(V_0 + 2V_1 + 4V_2 + 8V_3),\\
&CVF_{22}(V_0 - 2V_1 + 4V_2 + 8V_3),\\
&CVF_{22}(V_0 + 3V_1 - 9V_2 + 27V_3),\\
&CVF_{22}(V_3)),\\
CVF_{22}(W) = (&CVF_2(W_0) + CVF_2(W_1),\\
&CVF_2(W_1),\\
&CVF_2(W_0))\\
CVF_2(B) = (&B_0 + B_1, B_1, B_0),
\end{aligned}$$

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 (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 (3) and (17). The output of $CMF_{422}(A)$ 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 3.3, computation of (19) 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 (19) 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 which we share the results in the next section.

3.5 Implementation Results

As explained in Section 3.3, 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 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 $3.10 = 30$ 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 1 shows the cycle counts of the schoolbook matrix-vector multiplication and the TMVP_2 formula for various $n = 2^t$ values.

Table 1: Schoolbook vs. TMVP-2

t	n	$SB(n)$	$TMVP2(n)$
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 $3 \times 67 = 201$. 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 [21]. To make a fair comparison, we evaluate the polynomial multiplication algorithm in [21] with the polynomial reduction step since our algorithm already includes it. As can be seen in Table 2, our algorithm for multiplication in $\mathcal{R}_{2^{16}}$ is 24.5% faster and requires 16.5% less memory than the one in [21], which uses Toom-4 and the Karatsuba algorithms.

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

Cycles		Stack	
[21]	This work	[21]	This work
37804	28520 (-24.5%)	3800	3172 (-16.5%)

In this work, we only focus on an efficient residue polynomial multiplication algorithm and an improved version of this algorithm for Saber, not on a complete implementation of Saber. So, we use the publicly available codes from [21] and [29] to compose software packages for our applications to Saber. We make some adjustments to existing codes to integrate our algorithm into these packages. We compare the results with speed optimized implementation results given in [29].

As Saber is an MLWR based scheme, it performs polynomial matrix-vector multiplications for the key generation, encapsulation, and decapsulation algorithms. Table 3 shows the cycle counts of the polynomial matrix-vector multiplication (e.g., $\boldsymbol{A^T s}$ in line 5 of Algorithm 1 in [14]) 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 3: Polynomial Matrix-Vector Multiplication

	[29](speed)	TMVP	Block Rec.
$\ell = 2$	159 k	122 k(-23.3%)	106 k(-33.3%)
$\ell = 3$	317 k	273 k (-13.9%)	231 k (-27.1%)
$\ell = 4$	528 k	483 k(-8.5%)	403 k (-23.7%)

In Table 3, the comparison of cycle counts of matrix-vector multiplication using different algorithms are given. The third and fourth columns of Table 3 represent the proposed generic TMVP-based algorithm and the improved version explained in Section 2.2 and Section 3.4, respectively. Our generic algorithm improves the polynomial matrix-vector multiplication between 8.5% and 23.3% for all values of ℓ comparing the results from [29]. Moreover, our improved algorithm outperforms the polynomial matrix-vector multiplication in [29] by 33.3%, 27.1%, 23.6% for $\ell = 2, 3, 4$, respectively.

Table 4: Results of application to Saber

		[29] (speed)	TMVP	Block Rec.	
LightSaber	KeyGen:	466 k	421 k (-9.6%)	409 k(12.2%)	cycles
		14208	7932(-44.2%)	12536(-11.8%)	bytes
	Encaps:	653 k	591 k (-9.5%)	572 k(-12.4%)	cycles
		15928	9668(-39.3%)	14248(-10.5%)	bytes
	Decaps:	678 k	597 k(-11.9%)	574 k(-15.6%)	cycles
		16672	10412(-37.5%)	14992(-10.1%)	bytes
Saber	KeyGen:	853 k	810 k(-5%)	772 k(-9.5%)	cycles
		19824	12608(-36.4%)	18144(-8.5%)	bytes
	Encaps:	1103 k	1052 k (-4.6%)	996 k(-9.7%)	cycles
		22088	14872(-32.7%)	20392(-7.7%)	bytes
	Decaps:	1127 k	1058 k (-6.1%)	995 k(-11.7%)	cycles
		23184	15968(-31.1%)	21488(-7.3%)	bytes
FireSaber	KeyGen:	1340 k	1297 k(-3.2%)	1224 k(-8.7%)	cycles
		26448	20120(-23.9%)	24776(-6.3%)	bytes
	Encaps:	1642 k	1590 k(-3.2%)	1499 k(-8.7%)	cycles
		29228	22968(-21.4%)	27592(-5.6%)	bytes
	Decaps:	1679 k	1606 k(-4.3%)	1508 k(-10.2%)	cycles
		30768	24448(-20.5%)	29072(-5.5%)	bytes

The effect of our algorithms on the overall performance of Saber is also promising. Table 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 4, both of our algorithms improve the efficiency compared to the speed optimized version in [29]. Table 4 also shows the percentage of the gain in terms of both the execution time and stack usage that our algorithms achieve. We speed up the key generation between 3.2% and 9.6%, encapsulation between 3.2% and 9.5%, decapsulation between 4.3% and 11.9% with our generic TMVP-based algorithm. The enhancement in efficiency with our improved algorithm are between 8.7% and 12.2% for key generation, 8.7% and 12.4% for encapsulation, 10.2% and 15.6% for decapsulation. Furthermore, our generic TMVP-based algorithm reduces stack memory consumption between 20.5% and 44.2% while the improved algorithm reduces between 5.5% and 11.82% compared to the speed optimized implementation in [29].

4 Conclusion/Discussion

In this work, we focus on developing non-NTT-style multiplication algorithms for post-quantum cryptographic schemes, utilizing the Toeplitz matrix-vector multiplication. We derive two new TMVP formulas, TMVP-3 and TMVP-4, which require five and seven multiplications, respectively. Moreover, we propose an algorithm for multiplication in the ring $\mathcal{R}_{2^{13}} = \mathbb{Z}_{2^{13}}[x]/\langle x^{256}+1\rangle$ which exploits the new TMVP-4 formula. Our algorithm is faster than the state-of-the-art speed optimized implementation that uses Toom-Cook and Karatsuba. Furthermore, we improve the multiplication algorithm we propose for Saber using a non-recursive version of the block recombination method [17]. We implement the proposed algorithms on the ARM Cortex-M4 microcontroller and integrate our code into an existing implementation of Saber. The results of both the generic and optimized algorithms we propose indicate that TMVP-based multiplication algorithms might be good alternatives to Toom-Cook and Karatsuba methods. Based on the outcomes of this work and our preliminary results of application to NTRU, we expect that TMVP-based multiplication algorithms exploiting TMVP-3 and TMVP-4 formulas may improve the efficiency of lattice-based post-quantum schemes that are not originally NTT-friendly.

References

[1] Shoukat Ali and Murat 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):2477–2490, 2018.

[2] Erdem Alkim, Philipp Jakubeit, and Peter Schwabe. NewHope on ARM Cortex-M. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 332–349. Springer, 2016.

[3] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 719–737. Springer, 2012.

[4] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE, 2018.

[5] Joppe W Bos, Simon Friedberger, Marco Martinoli, Elisabeth Oswald, and Martijn Stam. Fly, you fool! Faster Frodo for the ARM Cortex-M4. *IACR Cryptol. ePrint Arch.*, 2018:1116, 2018.

[6] Murat Cenk and M Anwar Hasan. Some new results on binary polynomial multiplication. *Journal of Cryptographic Engineering*, 5(4):289–303, 2015.

[7] Murat Cenk, Christophe Negre, and M Anwar Hasan. Improved three-way split formulas for binary polynomial and Toeplitz matrix vector products. *IEEE Transactions on Computers*, 62(7):1345–1361, 2012.

[8] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M Schanck, Peter Schwabe, William Whyte, and Zhenfei Zhang. NTRU: algorithm specifications and supporting documentation (2019). *URL: https://csrc. nist. gov/projects/post-quantum-cryptography/round-2-submissions. Citations in this document*, 1.

[9] Chi-Ming Marvin Chung, Vincent Hwang, Matthias J Kannwischer, Gregor Seiler, Cheng-Jhih Shih, and Bo-Yin Yang. NTT Multiplication for NTT-unfriendly Rings. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 159–188, 2021.

[10] Stephen A Cook and Stål O Aanderaa. On the minimum computation time of functions. *Transactions of the American Mathematical Society*, 142:291–314, 1969.

[11] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[12] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In *International Conference on Cryptology in Africa*, pages 282–305. Springer, 2018.

[13] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. SABER: Mod-LWR based KEM. *Second PQC Standardization Conference, 2019, University of California, Santa Barbara, USA*, 2019.

[14] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Mod-LWR based KEM (round 2 submission). 2019. `https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround2.pdf`.

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

[16] Haining Fan and M Anwar Hasan. Subquadratic computational complexity schemes for extended binary field multiplication using optimal normal bases. *IEEE Transactions on Computers*, 56(10):1435–1437, 2007.

[17] M Anwar Hasan, Nicolas Meloni, Ashkan H Namin, and Christophe Negre. Block recombination approach for subquadratic space complexity binary field multiplication based on Toeplitz matrix-vector product. *IEEE Transactions on Computers*, 61(2):151–163, 2010.

[18] M Anwar Hasan and Christophe Negre. Multiway splitting method for toeplitz matrix vector product. *IEEE Transactions on Computers*, 62(7):1467–1471, 2012.

[19] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. NTRU: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium*, pages 267–288. Springer, 1998.

[20] Andreas Hülsing, Joost Rijneveld, John Schanck, and Peter Schwabe. High-speed key encapsulation from NTRU. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 232–252. Springer, 2017.

[21] Matthias J Kannwischer, Joost Rijneveld, and Peter 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*, pages 281–301. Springer, 2019. https://github.com/mupq/polymul-z2mx-m4.

[22] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[23] Matthias J Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4. 2019.

[24] Anatolii Karatsuba. Multiplication of multidigit numbers on automata. In *Soviet physics doklady*, volume 7, pages 595–596, 1963.

[25] Angshuman Karmakar, Ingrid Verbauwhede, et al. Saber on ARM CCA-secure module lattice-based key encapsulation on ARM. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 243–266, 2018. https://github.com/KULeuven-COSIC/SABER.

[26] Hugo Krawczyk. Lfsr-based hashing and authentication. In *Annual International Cryptology Conference*, pages 129–139. Springer, 1994.

[27] Hugo Krawczyk. New hash functions for message authentication. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 301–310. Springer, 1995.

[28] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.

[29] Jose Maria Bermudo Mera, Angshuman Karmakar, and Ingrid Verbauwhede. Time-memory trade-off in Toom-Cook multiplication: an application to module-lattice based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 222–244, 2020. https://github.com/KULeuven-COSIC/TCHES2020_SABER/tree/master/Cortex-M4.

[30] Dustin Moody, Gorjan Alagic, Daniel C. Apon, David A. Cooper, Quynh H. Dang, John M. Kelsey, Yi-Kai Liu, Carl A. Miller, Rene C. Peralta, Ray A. Perlner, and et al. *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process.* Jul 2020.

[31] NIST. Post-Quantum Cryptography Standardization, 2017. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization.

[32] Halil Kemal Taşkın and Murat Cenk. Speeding up curve25519 using toeplitz matrix-vector multiplication. In *Proceedings of the Fifth Workshop on Cryptography and Security in Computing Systems*, pages 1–6, 2018.

[33] Andrei L Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. In *Soviet Mathematics Doklady*, volume 3, pages 714–716, 1963.

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