Faster NTRU on ARM Cortex-M4 with TMVP-based multiplication*

İrem Keskinkurt Paksoy[1][0000−0003−2529−1150] and Murat Cenk[1][0000−0003−4941−8734]

Middle East Technical University, Ankara, Turkey
https://www.metu.edu.tr

Abstract. The Number Theoretic Transform (NTT), Toom-Cook, and Karatsuba are the most commonly used algorithms for implementing lattice-based finalists of the NIST PQC competition. In this paper, we propose Toeplitz matrix-vector product (TMVP) based algorithms for multiplication for all parameter sets of NTRU. We implement the proposed algorithms on ARM Cortex-M4. The results show that TMVP-based multiplication algorithms using the four-way TMVP formula are more efficient for NTRU. Our algorithms outperform the Toom-Cook method by up to 25.3%, and the NTT method by up to 19.8%. Moreover, our algorithms require less stack space than the others in most cases. We also observe the impact of these improvements on the overall performance of NTRU. We speed up the encryption, decryption, encapsulation, and decapsulation by up to 13.7%, 17.5%, 3.5%, and 14.1%, respectively, compared to state-of-the-art implementation.

Keywords: Lattice-based · Post-quantum · ARM Cortex-M4 · NTRU · Toeplitz · TMVP.

1 Introduction

The security of modern public-key cryptography relies on two hard-to-solve mathematical problems: The integer factorization problem and the discrete logarithm problem. There are no classical algorithms to solve these problems in polynomial time, even with very powerful computers. However, Shor's quantum computer algorithm [25] can solve these problems in polynomial time. The current asymmetric systems are still not broken because large enough quantum computers have not yet been built. In the light of the developments in this area, it is predicted that large-scale quantum computers will be built in fifteen years. Until that day comes, the classical asymmetric cryptosystems will remain secure. For this reason, research on quantum-resistant cryptosystems has been pursued for almost four decades. The studies on post-quantum cryptography (PQC) mainly focus on five main classes: lattice-based, code-based, multivariate polynomial-based, hash-based, and isogeny-based.

* The first author is partially supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) 2211-c graduate scholarship program.

NIST started PQC standardization competition process in 2017 [26] with 82 submissions, of which 69 were counted as the first round candidates. Only 26 of these candidates could make it to the second round, which started in January 2019 [1]. In July 2020, the finalists and the alternate candidates were announced [22], and the third round of the NIST PQC competition officially began. Three out of four public-key encryption (PKE)/key encapsulation mechanism (KEM) finalists and two out of three signature finalists are based on lattices.

The lattice-based finalists of the NIST PQC competition are defined on polynomial quotient rings. The key generation, encryption, and decryption algorithms of these schemes require multiplication in the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$ which is specified by different parameters for each scheme. The algorithm used for multiplication in the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$ directly affects the efficiency of the scheme. Various multiplication algorithms can be used for efficient implementations. The most commonly used algorithms are the schoolbook, Karatsuba [20], Toom-Cook [29, 8], and the Number Theoretic Transform (NTT) [9]. The paper [23] shows that multiplication algorithms based on Toeplitz matrix-vector products (TMVP) might also be good alternatives for efficient implementation of lattice-based PQC schemes. When deciding the best algorithm to use, the values of the parameters and the implementation platform must be taken into account. If the modulus q is a prime and $f(x)$ is the n-th cyclotomic polynomial satisfying $q \equiv 1 \mod n$, then NTT is the most efficient algorithm to use. One of the KEM finalists, namely Kyber [4] is an example of such schemes that utilize NTT multiplication. If the modulus q is not prime or specifically q is a power-of-two as in Saber [10, 11] and NTRU [6], then NTT cannot be used directly. In most of the implementations of such schemes, a combination of Toom-Cook, Karatsuba, and schoolbook algorithms, together with polynomial reduction, is used for multiplication in the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$. In [23], applying the proposed TMVP-based multiplication algorithm to Saber gives better results than such implementations. Shortly after this improvement, in [7], it is shown that NTT can be used when q is a power of two by making some adjustments.

Contribution: In this paper, we propose TMVP-based algorithms for multiplication in the rings $\mathbb{Z}_{2^{11}}[x]/\langle x^{509}-1\rangle$, $\mathbb{Z}_{2^{11}}[x]/\langle x^{677}-1\rangle$, $\mathbb{Z}_{2^{13}}[x]/\langle x^{701}-1\rangle$, and $\mathbb{Z}_{2^{12}}[x]/\langle x^{821}-1\rangle$ that speed up NTRU. The implementation of the proposed algorithms are faster than others in the literature for all parameter sets of NTRU.

Code: Our code is available at https://github.com/iremkp/NTRU-tmvp4-m4.git.

Structure of the paper: In Section 2, the definition and some properties of Toeplitz matrices, a brief description of the NTRU scheme, and some preliminary information required to the subject are given. The TMVP-based multiplication algorithms we propose and the results of our application to NTRU on ARM Cortex-M4 can be found in Section 3. Section 4 concludes the paper with a summary and some remarks.

2 Preliminaries

This section includes some definitions and properties to provide a background to the subject. First, we introduce Toeplitz matrices, and the formulas used to calculate Toeplitz matrix-vector products (TMVP). Then, we show how a multiplication in $\mathbb{Z}_q[x]/\langle x^n \pm 1\rangle$ can be calculated as a TMVP. After recalling one of the finalists of NIST PQC competition, NTRU, we end this section with a brief explanation of ARM Cortex-M4, which we use as the implementation platform.

2.1 Toeplitz Matrices

Toeplitz matrices are used in many cryptographic applications in the literature. In [12], the first time cryptographic use of Toeplitz matrix-vector product (TMVP) for multiplication in binary extension fields can be seen. Since then, many studies on cryptographic use of TMVP have been pursued [2, 12, 14, 15, 24, 27, 28]. Recently, in [23], TMVP is used to develop a multiplication algorithm for Saber [10, 11].

Definition 1. *Let n be a positive integer. A Toeplitz matrix is an $n \times n$ matrix in which the entries located on a line parallel to the main diagonal are constant.*

The following matrix T is an example of a 5×5 Toeplitz matrix:

$$T = \begin{pmatrix} t_0 & t_1 & t_2 & t_3 & t_4 \\ t_5 & t_0 & t_1 & t_2 & t_3 \\ t_6 & t_5 & t_0 & t_1 & t_2 \\ t_7 & t_6 & t_5 & t_0 & t_1 \\ t_8 & t_7 & t_6 & t_5 & t_0 \end{pmatrix}.$$

Since an $n \times n$ Toeplitz matrix can be determined with only $2n-1$ elements, it is sufficient to perform $2n-1$ entry additions to calculate the sum of two Toeplitz matrices, which is again a Toeplitz matrix. Moreover, every submatrix of a Toeplitz matrix is also a Toeplitz matrix. These properties allow us to calculate a TMVP more efficiently than the schoolbook matrix-vector multiplication via TMVP formulas.

2.2 TMVP formulas

There are various split formulas in the literature for efficiently compute a TMVP. We refer the reader to [2, 13, 23] for further information on TMVP formulas. In this section, we only recall the two-, three-, and four-way TMVP formulas we use in this work. We use the notation A to denote an $n \times n$ Toeplitz matrix and B for a vector of length n. For a k-split formula, we use the notations A_i and B_i to denote the n/k-dimensional partitions of A and B, respectively.

The two-way TMVP formula (TMVP-2): We can calculate an n dimensional TMVP via three $n/2$-dimensional TMVPs as follows:

$$A \cdot B = \begin{pmatrix} A_1 & A_0 \\ A_2 & A_1 \end{pmatrix} \begin{pmatrix} B_0 \\ B_1 \end{pmatrix} = \begin{pmatrix} P_0 + P_1 \\ P_0 - P_2 \end{pmatrix}, \tag{1}$$

where

$$\begin{aligned} P_0 &= A_1(B_0 + B_1), \\ P_1 &= (A_0 - A_1)B_1, \\ P_2 &= (A_1 - A_2)B_0. \end{aligned}$$

The three-way TMVP formula with six multiplications (TMVP-3$_{(6)}$): An n-dimensional TMVP

$$A \cdot B = \begin{pmatrix} A_2 & A_1 & A_0 \\ A_3 & A_2 & A_1 \\ A_4 & A_3 & A_2 \end{pmatrix} \begin{pmatrix} B_0 \\ B_1 \\ B_2 \end{pmatrix} = \begin{pmatrix} P_1 + P_4 + P_5 \\ P_2 - P_4 + P_6 \\ P_3 - P_5 - P_6 \end{pmatrix}, \tag{2}$$

where

$$\begin{aligned} P_1 &= (A_0 + A_1 + A_2)B_2, \\ P_2 &= (A_1 + A_2 + A_3)B_1, \\ P_3 &= (A_2 + A_3 + A_4)B_0, \\ P_4 &= A_1(B_1 - B_2), \\ P_5 &= A_2(B_0 - B_2), \\ P_6 &= A_3(B_0 - B_1). \end{aligned}$$

The four-way TMVP formula (TMVP-4): An n-dimensional TMVP can be calculated via seven $n/4$-dimensional TMVPs by using the TMVP-4 formula proposed in [23]. Without loss of generality, we assume that n is a multiple of four. We partition the Toeplitz matrix and the vector and compute the product as follows:

$$A \cdot B = \begin{pmatrix} A_3 & A_2 & A_1 & A_0 \\ A_4 & A_3 & A_2 & A_1 \\ A_5 & A_4 & A_3 & A_2 \\ A_6 & A_5 & A_4 & A_3 \end{pmatrix} \begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix} = \begin{pmatrix} P_1 - P_2 + 8P_3 - 8P_4 + 27P_5 + P_6 \\ P_1 + P_2 + 4P_3 + 4P_4 + 9P_5 \\ P_1 - P_2 + 2P_3 - 2P_4 + 3P_5 \\ P_0 + P_1 + P_2 + P_3 + P_4 + P_5 \end{pmatrix} \tag{3}$$

where

$$
\begin{aligned}
P_0 &= \frac{1}{12}\left(12A_6 - 4A_5 - 15A_4 + 5A_3 + 3A_2 - A_1\right) B_0,\\
P_1 &= \frac{1}{12}\left(12A_5 + 8A_4 - 7A_3 - 2A_2 + A_1\right)(B_0 + B_1 + B_2 + B_3),\\
P_2 &= \frac{1}{24}\left(-12A_5 + 16A_4 - A_3 - 4A_2 + A_1\right)(B_0 - B_1 + B_2 - B_3),\\
P_3 &= \frac{1}{24}\left(-6A_5 - A_4 + 7A_3 + A_2 - A_1\right)(B_0 + 2B_1 + 4B_2 + 8B_3),\\
P_4 &= \frac{1}{120}\left(6A_5 - 5A_4 - 5A_3 + 5A_2 - A_1\right)(B_0 - 2B_1 + 4B_2 - 8B_3),\\
P_5 &= \frac{1}{120}\left(4A_5 - 5A_3 + A_1\right)(B_0 + 3B_1 + 9B_2 + 27B_3),\\
P_6 &= \left(-12A_5 + 4A_4 + 15A_3 - 5A_2 - 3A_1 + A_0\right) B_3.
\end{aligned}
\tag{4}
$$

The Toeplitz matrices A_i are of dimesion $n/4 \times n/4$, and the vectors B_i are of dimension $n/4 \times 1$. The arithmetic complexity of the TMVP-4 and the schoolbook matrix-vector multiplication are given in Table 1. We use $M(n)$ to denote the arithmetic complexity of the corresponding algorithm for an n dimensional TMVP.

Table 1. Arithmetic complexity comparison

Schoolbook	$M(n) = 2n^2 - n$
TMVP-2	$M(n) = 3M(n/2) + 3n - 1$
TMVP-3$_{(6)}$	$M(n) = 6M(n/3) + 5n - 1$
TMVP-4	$M(n) = 7M(n/4) + 33n/2 - 21$

We use the arithmetic complexity values in Table 1 to determine the dimension at which the schoolbook algorithm is more efficient. For example,

2.3 Multiplication modulo $x^n \pm 1$ using TMVP

If $c(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$ is the product of $a(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1}$ and $b(x) = b_0 + b_1x + \cdots + b_{n-1}x^{n-1}$ modulo $x^n \pm 1$ then the coefficients c_i can be calculated via the following TMVP:

$$
\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{pmatrix}
=
\begin{pmatrix}
a_0 & \mp a_{n-1} & \mp a_{n-2} & \dots & \mp a_3 & \mp a_2 & \mp a_1 \\
a_1 & a_0 & \mp a_{n-1} & \dots & \mp a_4 & \mp a_3 & \mp a_2 \\
a_2 & a_1 & a_0 & \dots & \mp a_5 & \mp a_4 & \mp a_3 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
a_{n-2} & a_{n-3} & a_{n-4} & \dots & a_1 & a_0 & \mp a_{n-1} \\
a_{n-1} & a_{n-2} & a_{n-3} & \dots & a_2 & a_1 & a_0
\end{pmatrix}
\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{pmatrix}.
\tag{5}
$$

For large values of n schoolbook matrix-vector multiplication is not efficient to compute (5). Depending on the dimension n and the implementation platform, we prefer to use TMVP formulas recursively or iteratively. The two- and three-way formulas for TMVPs defined on integers given in [2, 13] require three and six multiplications, respectively. The three- and four-way TMVP formulas, which require five and seven multiplications, can be found in [23].

2.4 NTRU

NTRU is one of the lattice-based finalists of the NIST PQC competition. NTRU key encapsulation mechanism (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 [16, 17]. 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 CPA-DPKE are given in Algorithm 1, Algorithms 2, Algorithm 3, respectively. The details of the operations used in these algorithms can be found in [6]. In Algorithms 1, 2, and 3, n is an odd prime number and $\boldsymbol{\Phi}_i$ is the the i-th cyclotomic polynomial. Therefore, we have $\boldsymbol{\Phi}_1 = x - 1$, $\boldsymbol{\Phi}_n = x^{n-1} + x^{n-2} + \cdots + x + 1$, and $\boldsymbol{\Phi}_1\boldsymbol{\Phi}_n = x^n - 1$.

Algorithm 1 KeyGen(*seed*)

1: $(\mathbf{f},\mathbf{g}) \leftarrow \mathsf{Sample}(\text{seed})$
2: $\mathbf{f}_q \leftarrow (1/\mathbf{f}) \bmod (q, \boldsymbol{\Phi}_n)$
3: $\mathbf{h} \leftarrow (3.\mathbf{g}.\mathbf{f}_q) \bmod (q, \boldsymbol{\Phi}_1\boldsymbol{\Phi}_n)$
4: $\mathbf{h}_q \leftarrow (1/\mathbf{h}) \bmod (q, \boldsymbol{\Phi}_n)$
5: $\mathbf{f}_p \leftarrow (1/\mathbf{f}) \bmod (3, \boldsymbol{\Phi}_n)$
6: **return** $((\mathbf{f},\mathbf{f}_p,\mathbf{f}_q),\mathrm{h})$

Algorithm 3 Decrypt$((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{c})$

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

Algorithm 2 Encrypt$(\mathbf{h}, (\mathbf{r},\mathbf{m}))$

1: $\mathbf{m}' \leftarrow \mathsf{Lift}(\mathbf{m})$
2: $\mathbf{c} \leftarrow (\mathbf{r}.\mathbf{h} + \mathbf{m}') \bmod (q, \boldsymbol{\Phi}_1\boldsymbol{\Phi}_n)$
3: **return** $\mathbf{c}$

Multiplication in $\mathbb{Z}_q[x]/\langle \boldsymbol{\Phi}_1\boldsymbol{\Phi}_n \rangle = \mathbb{Z}_q[x]/\langle x^n - 1 \rangle$ is the main operation we work on to improve the efficiency of encryption and decryption algorithms. Improvements in multiplication have minimal impact on the key generation because of polynomial inversion operations. We propose TMVP-based algorithms for multiplication in the rings $\mathbb{Z}_{2^q}[x]/\langle x^n - 1 \rangle$ where n and q parameters are from NTRU. We explain the details of the algorithm and share the implementation results in Section 3.

2.5 ARM Cortex-M4

The ARM Cortex-M4 microcontroller 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 fourteen 32-bit registers available for the developer. As in many implementations [3, 5, 18, 19, 21] of the NIST PQC competition candidates, we use the STM32F407DISC1 development board. The SIMD (single instruction multiple data) instructions supported by ARM Cortex-M4 allow us to perform arithmetic operations on 16-bit halfwords of multiple registers in parallel. Since we work on NTRU, which uses integers of bit length less than 16, we can place two coefficients in one register. These instructions enable efficient implementation of schoolbook matrix-vector multiplication for small dimensions. smuadx and smlad are two examples of such instructions that we frequently use in our implementations. The operations that these instructions perform are given as follows:

$$\begin{aligned} \mathsf{smuadx\ Rd,\ Rn,\ Rm} &: \quad \mathsf{Rd} = \mathsf{Rn}_b\mathsf{Rm}_t + \mathsf{Rn}_t\mathsf{Rm}_b \\ \mathsf{smlad\ Rd,\ Rn,\ Rm,\ Rs} &: \quad \mathsf{Rd} = \mathsf{Rn}_b\mathsf{Rm}_b + \mathsf{Rn}_t\mathsf{Rm}_t + \mathsf{Rs} \end{aligned}$$

The indices b and t denote the bottom (bits $0-15$) and top (bits $16-31$) halfwords of the relevant register. So smlad calculates the sum of the product of the bottom half words and the top halfwords of Rn and Rm. Then, adds this result to Rs, and writes the final result to Rd. The contents of Rn, Rm, and Rs do not change after smlad instruction.

3 TMVP-based Multiplication for NTRU

In [23], it is shown that TMVP-based algorithms are good alternatives for efficient multiplication in the rings of the form $\mathbb{Z}_q[x]/\langle x^n \pm 1\rangle$. The algorithm proposed in [23] outperforms all other Cortex-M4 implementations that use a combination of Toom-Cook, Karatsuba, and schoolbook methods. In this work, we propose similar algorithms utilizing the TMVP-4 formula for ntruhps2048509, ntruhrss701, ntruhps2048677, and ntruhps4096821, and implement them on ARM Cortex-M4. The results show that our implementations are faster than the ones in [7, 19].

3.1 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, 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:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} a_0 & a_{n-1} & a_{n-2} & \dots & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_{n-1} & \dots & a_4 & a_3 & a_2 \\ a_2 & a_1 & a_0 & \dots & a_5 & a_4 & a_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & a_{n-4} & \dots & a_1 & a_0 & a_{n-1} \\ a_{n-1} & a_{n-2} & a_{n-3} & \dots & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{pmatrix} \tag{6}$$

where $a(x) = \sum_{i=0}^{n-1} a_i x^i, b(x) = \sum_{i=0}^{n-1} b_i x^i$, and $c(x) = \sum_{i=0}^{n-1} c_i x^i$. 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 efficient TMVP split formulas. We would not consider the schoolbook method for efficient matrix-vector multiplication for these dimensions. So, we pad these prime-dimensional TMVPs to facilitate the TMVP formulas in the literature. Our padding strategy for the Toeplitz matrix in (6) 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 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 via padding (6) we would have the following TMVP:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{m-n} \\ \vdots \\ c_{n-2} \\ c_{n-1} \\ c_n \\ \vdots \\ c_{m-2} \\ c_{m-1} \end{pmatrix} = \left(\begin{array}{cccccc|ccc} a_0 & a_{n-1} & \dots & a_2 & a_1 & & 0 & \dots & 0 \\ a_1 & a_0 & \dots & a_3 & a_2 & & a_1 & \dots & 0 \\ a_2 & a_1 & \dots & a_4 & a_3 & & a_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots & \ddots & \vdots \\ a_{m-n} & a_{m-n-1} & \dots & a_{m-n+2} & a_{m-n+1} & & a_{m-n} & \dots & a_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots & \ddots & \vdots \\ a_{n-2} & a_{n-3} & \dots & a_0 & a_{n-1} & & a_{n-2} & \dots & a_{2n-m-1} \\ a_{n-1} & a_{n-2} & \dots & a_1 & a_0 & & a_{n-1} & \dots & a_{2n-m} \\ 0 & a_{n-1} & \dots & a_2 & a_1 & & a_0 & \dots & a_{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & a_{n-1} & a_{m-n} & a_{m-n-1} & & a_{m-n-2} & \dots & a_{n-1} \\ 0 & 0 & \dots & a_{m-n+1} & a_{m-n} & & a_{m-n-1} & \dots & a_0 \end{array}\right) \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{m-n} \\ \vdots \\ b_{n-2} \\ b_{n-1} \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \tag{7}$$

Here, we assume that m is a proper dimension for using TMVP formulas. As can be seen in (7), we append $m-n$ zero entries at the end of the vector and the first row and column of the Toeplitz matrix in (6). Then, we fill the entries so that elements on a line parallel to the main diagonal are the same. After calculating the TMVP in (7) efficiently via TMVP split formulas, ignoring the last $m-n$ terms is all we need to do to obtain the result of (6). We first need to decide the dimension m before padding. The dimension of the padded TMVP mainly depends on n and the formulas we choose to use. In the following sections, we share our choices for dimensions and explain the TMVP-based multiplication algorithms we propose for every parameter set of NTRU.

3.2 TMVP-based multiplication algorithms for NTRU

As mentioned in the previous section, the dimensions of TMVPs representing the residue polynomial multiplication are prime for NTRU. Therefore, we pad these prime-dimensional TMVPs in order to get TMVPs of the selected dimension. We determine the dimensions depending on several factors, such as n, split formulas we use, and the dimension for schoolbook multiplications. We elaborately explain our way of deciding the dimension for padded TMVPs for every parameter set.

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 following TMVP:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_{507} \\ c_{508} \end{pmatrix} = \begin{pmatrix} a_0 & a_{508} & a_{507} & \dots & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_{508} & \dots & a_4 & a_3 & a_2 \\ a_2 & a_1 & a_0 & \dots & a_5 & a_4 & a_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ a_{507} & a_{506} & a_{505} & \dots & a_1 & a_0 & a_{508} \\ a_{508} & a_{507} & a_{506} & \dots & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_{507} \\ b_{508} \end{pmatrix} \tag{8}$$

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 (8) 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 dimension 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, we are free to apply TMVP-2 formulas until we reach a size of which the schoolbook is faster than TMVP formulas. According to [23], 16 is the optimal dimension for switching the multiplication algorithm from TMVP-2 to the schoolbook method. So, we decide 512 for the dimension of the padded matrix, which is given in (9).

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_{507} \\ c_{508} \\ c_{509} \\ c_{510} \\ c_{511} \end{pmatrix} = \left(\begin{array}{ccccccc:ccc} a_0 & a_{508} & a_{507} & \dots & a_3 & a_2 & a_1 & 0 & 0 & 0 \\ a_1 & a_0 & a_{508} & \dots & a_4 & a_3 & a_2 & a_1 & 0 & 0 \\ a_2 & a_1 & a_0 & \dots & a_5 & a_4 & a_3 & a_2 & a_1 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{507} & a_{506} & a_{505} & \dots & a_1 & a_0 & a_{508} & a_{507} & a_{506} & a_{505} \\ a_{508} & a_{507} & a_{506} & \dots & a_2 & a_1 & a_0 & a_{508} & a_{507} & a_{506} \\ \hdashline 0 & a_{508} & a_{507} & \dots & a_3 & a_2 & a_1 & a_0 & a_{508} & a_{507} \\ 0 & 0 & a_{508} & \dots & a_4 & a_3 & a_2 & a_1 & a_0 & a_{508} \\ 0 & 0 & 0 & \dots & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 \end{array}\right) \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_{n-2} \\ b_{n-1} \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{9}$$

As seen in (9), 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 (7). 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 (8) 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 3. 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.

Multiplication Algorithm for ntruhrss701 and ntruhps2048677: We have $q = 2^{13}$, $n = 701$, and $f(x) = x^{701} - 1$ for ntruhrss701. Like the previous one, 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 189 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 it seems like we improve the multiplication in $\mathbb{Z}_{2^{13}}[x]/\langle x^{701} - 1\rangle$. However, when we test the correctness, we observe that the algorithm fails *sometimes* in certain components of the result. We think incorrect memory usage might cause this problem, but we can not exactly point it out. We presume that for an odd-dimensional schoolbook matrix-vector multiplication (11 in this case), loading/storing from/to addresses that are not multiple of four may cause this fault. On the other hand, it is known that accessing addresses that are not divisible by four causes a performance hit. Therefore, we see no harm in increasing the dimension to find a path that requires small and even dimensional TMVPs to be calculated via the schoolbook matrix-vector multiplication. We avoid the paths that end up with an odd dimension from now on. So, we keep checking the dimensions hoping to find a path with small and even dimensional TMVPs at the end. We see that 720 is another possible dimension for an efficient computation tracing the following path:

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

We implement this algorithm which we call TMVPmul-701-720 on ARM Cortex-M4 and observe that it is faster than TMVPmul-701-704. The results in Table 3 show that TMVPmul-701-720 is faster than any other Cortex-M4 implementations in the literature.

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. 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. We eliminate this path ending with an odd dimension (19 in this case) due to the reason we explained before. Also, we believe that 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 already examine for ntruhrss701. Because of the same motive explained above, we choose 720 for the dimension of the padded matrix for ntruhps2048677 as well. Appending 43 zero entries at the end of the vector and the first row and column of the Toeplitz matrix gives us 720-dimensional padded TMVP. The multiplication algorithm for ntruhps2048677 which we denote by TMVPmul-677-720 tracing the same path as TMVPmul-701-720 is implemented on ARM Cortex-M4 and share the results in Table 3. Our implementation of TMVPmul-677-720 is the fastest compared to other Cortex-M4 implementations in the literature.

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 and eliminate those requiring large or odd-dimensional schoolbook multiplications. We determine 864 as the dimension of the padded TMVP after this elimination. The multiplication algorithm we propose for ntruhps4096821 which is referred by TMVPmul-821-864 traces the following path:

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

The results of the implementation of TMVPmul-821-864 on ARM Cortex-M4 are given in Table 3. It can be seen that TMVPmul-821-864 implementation is faster than the state-of-the-art implementations.

3.3 Implementation Results

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 simultaneously perform operations on halfwords of different registers. These instructions are called SIMD (single instruction multiple data) and enable us to implement efficient matrix-vector multiplications for small dimensions. The source code of our implementation can be found at https://github.com/iremkp/NTRU-tmvp4-m4.git. The adjustments on parameters required by both the NTT and TMVP-based methods are given in Table 2. 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 2 contain the n, q pairs used by TMVPmul-509-512, TMVPmul-677-720, TMVPmul-701-720, and TMVPmul-821-864, respectively.

Table 2. Comparison of the algorithms for polynomial multiplication

		NTT		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 2. 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 3. Our 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 3. Comparison of the algorithms for polynomial multiplication

	Cycles			Stack		
	Toom([18])	NTT([7])	TMVPmul	Toom([18])	NTT([7])	TMVPmul
ntruhps2048509	108717	101326	81238	7660	8352	7052
ntruhps2048677	182150	156235	144289	12984	100	10664
ntruhrss701	179994	155898	144228	13420	104	10668
ntruhps4096821	239133	199299	192983	15708	128	12788

The cycle count for Toom method includes the polynomial reduction.

As the results in Table 3 show, our TMVPmul-509-512 algorithm is 19.8% faster and consumes 15.5% less stack memory than the NTT method in [7]. Similarly, TMVPmul-509-512 is 25.2% faster and consumes 7.9% less stack memory than the Toom-Cook method in [18]. For the other parameter sets, we also speedup the residue polynomial multiplication but we are not able to reduce the stack usage. Compared to NTT implementation [7], we improve the ring multiplication by 7.5%, 7.6%, and 3.1% with TMVPmul-701-720,TMVPmul-677-720, and TMVPmul-821-864, respectively.

To observe the effect of the proposed algorithms on overall performance of NTRU, we use the software package accompanying the paper [7]. We integrate the assembly codes we write for the algorithms TMVPmul-509-512,TMVPmul-677-720,TMVPmul-701-720, and TMVPmul-821-864 to the implementations of ntruhps2048509, ntruhps2048677, ntruhrss701, and ntruhps4096821, respectively. Table 4 shows the results of the applications of the proposed multiplication algorithms to both the NTRU CPA-DPKE and NTRU CCA-KEM. The comparison of the results of our algorithms and the NTT method can be also seen in Table 4.

Improving polynomial multiplication does not significantly impact the performance of the NTRU CPA-DPKE key generation algorithm since the polynomial inversion operations dominate it. Nonetheless, we reduce the stack memory usage by 13.9%, 9.2%, 10.5%, and 9.5% for the algorithms given in Table 4. Although the improvement in performance of the NTRU CPA-DPKE key generation algorithm is not much, the encryption and decryption algorithms are accelerated. The percentages of the improvements are given in Table 4. Our algorithms speed up encryption by 13.7%, 4.9%, 4.1%, and 1.8%, and decryption by 17.5%, 13%, 11.6%, and 11.8% compared to [7]. The decryption algorithm of NTRU requires three multiplications, and TMVP-based algorithms can target all of these multiplications, whereas NTT targets only one of them. For this reason, the improvements are more prominent for decryption than they are for encryption. Similarly, we speed up decapsulation more than encapsulation. More or less, our application outperforms the NTT method for NTRU CCA-KEM for encapsulation and decapsulation.

Table 4. Results of application to NTRU

			NTT [7]	This work		
ntruhps2048509	CCA	KeyGen:	79665 k	78766 k	(−1.1%)	cycles
			21404	18748	(−12.4%)	bytes
		Encaps:	569 k	549 k	(−3.5%)	cycles
			14080	12780	(−9.2%)	bytes
		Decaps:	538 k	462 k	(−14.1%)	cycles
			14812	12156	(−17.9%)	bytes
	CPA	KeyGen:	79618 k	78718 k	(−1.1%)	cycles
			18972	16316	(−13.9%)	bytes
		Enc:	153 k	132 k	(−13.7%)	cycles
			11440	10140	(−11.3%)	bytes
		Dec:	434 k	358 k	(−17.5%)	cycles
			13852	11196	(−15.5%)	bytes
ntruhps2048677	CCA	KeyGen:	143731 k	142378 k	(−0.9%)	cycles
			28512	26192	(−8.1%)	bytes
		Encaps:	827 k	816 k	(−1.3%)	cycles
			9044	18264	(+101.9%)	bytes
		Decaps:	818 k	729 k	(−10.9%)	cycles
			19736	17416	(−11.8%)	bytes
	CPA	KeyGen:	143672 k	142301 k	(−0.9%)	cycles
			25280	22960	(−9.2%)	bytes
		Enc:	224 k	213 k	(−4.9%)	cycles
			4196	14760	(+251.8%)	bytes
		Dec:	676 k	588 k	(−13.0%)	cycles
			18472	16152	(−12.6%)	bytes
ntruhrss701	CCA	KeyGen:	154406 k	153508 k	(−0.6%)	cycles
			27572	24820	(−9.9%)	bytes
		Encaps:	380 k	369 k	(−2.9%)	cycles
			7412	16612	(+124.1%)	bytes
		Decaps:	871 k	787 k	(−9.6%)	cycles
			20564	17812	(−13.4%)	bytes
	CPA	KeyGen:	154377 k	153479 k	(−0.6%)	cycles
			26156	23404	(−10.5%)	bytes
		Enc:	274 k	263 k	(−4.1%)	cycles
			5716	14916	(+160.1%)	bytes
		Dec:	716 k	633 k	(−11.6%)	cycles
			19084	16332	(−14.4%)	bytes
ntruhps4096821	CCA	KeyGen:	207504 k	212377 k	(+2.3%)	cycles
			34516	31604	(−8.4%)	bytes
		Encaps:	1035 k	1026 k	(−0.9%)	cycles
			10936	21996	(+101.1%)	bytes
		Decaps:	1030 k	914 k	(−11.3%)	cycles
			23964	21044	(−12.2%)	bytes
	CPA	KeyGen:	208772 k	212270 k	(+1.7%)	cycles
			30604	27692	(−9.5%)	bytes
		Enc:	285 k	280 k	(−1.8%)	cycles
			5096	17756	(+248.4%)	bytes
		Dec:	862 k	760 k	(−11.8%)	cycles
			22348	19428	(−13.1%)	bytes

4 Conclusion

This paper showed that using the TMVP approach for multiplications in NTRU results in a better running time for all parameter sets and less stack space usage in most cases. We designed new multiplication algorithms for ntruhps2048509, ntruhps2048677, ntruhps4096821, and ntruhrss701 by combining the four-, three-, two-way TMVP formulas, and the standard (schoolbook) multiplication method. The 4-way method calls seven products of size one-fourth of the original size and improves the arithmetic complexity. Using the TMVP-4 formula leads to better implementation results on ARM Cortex M4. The running time of the proposed algorithms for NTRU enhances the best-known previous NTRU encryption, decryption, encapsulation, and decapsulation implementation results by up to 13.7%, 17.5%, 3.5%, and 14.1%, respectively. Moreover, it was revealed that the stack usage decreases in most cases.

Based on the results of this work, the proposed TMVP-based algorithms in this paper are very advantageous for NTRU. With TMVP-based algorithms, we can target every multiplication in every algorithm of NTRU, unlike the NTT method. Another practical feature of TMVP-based algorithms is that they can be diversified easily because the restriction on parameters is not that exclusive. In fact, the only restriction required is n to be a multiple of some integer depending on the TMVP formulas used. TMVP-based multiplication algorithms have never been considered for post-quantum schemes before [23]. The results show that the TMVP approach for multiplication in residue polynomial rings can be preferred to obtain high speed and efficient results for the post-quantum cryptography that uses those rings. With this work, we believe that TMVP-based algorithms attract the attention of the cryptographic community, and more research on the subject will take place in the future.

References

1. Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Liu, Y.K., Miller, C., Moody, D., Peralta, R., et al.: 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)
2. Ali, S., Cenk, M.: 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)
3. Alkim, E., Jakubeit, P., Schwabe, P.: Newhope on arm cortex-m. In: International Conference on Security, Privacy, and Applied Cryptography Engineering. pp. 332–349. Springer (2016)
4. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-kyber: a cca-secure module-lattice-based kem. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367. IEEE (2018)
5. Bos, J.W., Friedberger, S., Martinoli, M., Oswald, E., Stam, M.: Fly, you fool! faster frodo for the arm cortex-m4. IACR Cryptol. ePrint Arch. **2018**, 1116 (2018)

6. Chen, C., Danba, O., Hoffstein, J., Hülsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P., Whyte, W., Zhang, Z.: NTRU Algorithm Specifications and Supporting Documentation. In: Second PQC Standardization Conference (2019), https://ntru.org/f/ntru-20190330.pdf
7. Chung, C.M.M., Hwang, V., Kannwischer, M.J., Seiler, G., Shih, C.J., Yang, B.Y.: NTT Multiplication for NTT-unfriendly Rings. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 159–188 (2021)
8. Cook, S.A., Aanderaa, S.O.: On the minimum computation time of functions. Transactions of the American Mathematical Society **142**, 291–314 (1969)
9. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex fourier series. Mathematics of computation **19**(90), 297–301 (1965)
10. D'Anvers, J.P., Karmakar, A., Roy, S.S., Vercauteren, F.: 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)
11. D'Anvers, J.P., Karmakar, A., Roy, S.S., Vercauteren, F.: Saber: Mod-lwr based kem. Second PQC Standardization Conference, 2019, University of California, Santa Barbara, USA (2019)
12. Fan, H., Hasan, M.A.: A new approach to subquadratic space complexity parallel multipliers for extended binary fields. IEEE Transactions on Computers **56**(2), 224–233 (2007)
13. Fan, H., Hasan, M.A.: Subquadratic computational complexity schemes for extended binary field multiplication using optimal normal bases. IEEE Transactions on Computers **56**(10), 1435–1437 (2007)
14. Hasan, M.A., Meloni, N., Namin, A.H., Negre, C.: 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)
15. Hasan, M.A., Negre, C.: Multiway splitting method for toeplitz matrix vector product. IEEE Transactions on Computers **62**(7), 1467–1471 (2012)
16. Hoffstein, J.: NTRU: a new high speed public key cryptosystem. presented at the rump session of Crypto 96 (1996)
17. Hoffstein, J., Pipher, J., Silverman, J.H.: Ntru: A ring-based public key cryptosystem. In: International Algorithmic Number Theory Symposium. pp. 267–288. Springer (1998)
18. Kannwischer, M.J., Rijneveld, J., Schwabe, P.: 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
19. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: PQM4: Post-quantum crypto library for the ARM Cortex-M4, https://github.com/mupq/pqm4
20. Karatsuba, A.: Multiplication of multidigit numbers on automata. In: Soviet physics doklady. vol. 7, pp. 595–596 (1963)
21. Karmakar, A., Verbauwhede, I., 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
22. Moody, D., Alagic, G., Apon, D.C., Cooper, D.A., Dang, Q.H., Kelsey, J.M., Liu, Y.K., Miller, C.A., Peralta, R.C., Perlner, R.A., et al.: Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process (Jul 2020). https://doi.org/10.6028/nist.ir.8309, http://dx.doi.org/10.6028/NIST.IR.8309

23. Paksoy, I.K., Cenk, M.: TMVP-based Multiplication for Polynomial Quotient Rings and Application to Saber on ARM Cortex-M4. Cryptology ePrint Archive, Report 2020/1302 (2020), http://eprint.iacr.org/2020/1302.pdf
24. Pan, J.S., Lee, C.Y., Sghaier, A., Zeghid, M., Xie, J.: 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), 1614–1622 (2019)
25. Shor, P.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science. pp. 124–134 (1994). https://doi.org/10.1109/SFCS.1994.365700
26. of Standarts, N.I., Standardization, T.N.P.Q.C.: (2017), https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization
27. Taşkın, H.K., Cenk, M.: Tmvp-friendly primes for efficient elliptic curve cryptography. In: 2020 International Conference on Information Security and Cryptology (ISCTURKEY). pp. 80–87. IEEE (2020)
28. Taşkın, H.K., Cenk, M.: 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)
29. Toom, A.L.: The complexity of a scheme of functional elements realizing the multiplication of integers. In: Soviet Mathematics Doklady. vol. 3, pp. 714–716 (1963)