IACR Communications in Cryptology
ISSN 3006-5496, Vol. 1, No. 2, 63 pages.

https://doi.org/10.62056/a0ivr-10k

Check for updates

# A Survey of Polynomial Multiplications for Lattice-Based Cryptosystems

Vincent Hwang

Max Planck Institute for Security and Privacy, Bochum, Germany

**Abstract.** We survey various mathematical tools used in software works multiplying polynomials in

$$\frac{\mathbb{Z}_q[x]}{\langle x^n - \alpha x - \beta \rangle}.$$

In particular, we survey implementation works targeting polynomial multiplications in lattice-based cryptosystems Dilithium, Kyber, NTRU, NTRU Prime, and Saber with instruction set architectures/extensions Armv7-M, Armv7E-M, Armv8-A, and AVX2.

There are three emphases in this paper: (i) modular arithmetic, (ii) homomorphisms, and (iii) vectorization. For modular arithmetic, we survey Montgomery, Barrett, and Plantard multiplications. For homomorphisms, we survey (a) various homomorphisms such as Cooley–Tukey FFT, Good–Thomas FFT, Bruun's FFT, Rader's FFT, Karatsuba, and Toom–Cook; (b) various algebraic techniques for adjoining nice properties to the coefficient rings, including localization, Schönhage's FFT, Nussbaumer's FFT, and coefficient ring switching; and (c) various algebraic techniques related to the polynomial moduli, including twisting, composed multiplication, evaluation at $\infty$, truncation, incomplete transformation, striding, and Toeplitz matrix-vector product. For vectorization, we survey the relations between homomorphisms and vector arithmetic.

We then go through several case studies: We compare the implementations of modular multiplications used in Dilithium and Kyber, explain how the matrix-to-vector structure was exploited in Saber, and review the design choices of transformations for NTRU and NTRU Prime with vectorization. Finally, we outline several interesting implementation projects.

**Keywords:** Lattice-based cryptography · Polynomial multiplication · Modular arithmetic · Homomorphism · Vectorization

# 1 Introduction

Lattice-based cryptosystems have gained more popularity due to their balancing performance and the Post-Quantum Cryptography Standardization by the National Institute of Standards and Technology [NIS]. Among the commonly used building blocks of lattice-based cryptosystems, polynomial multiplication is one of the operations dominating the performance cycles. In this paper, we survey various implementation aspects of polynomial multiplications in the ring

$$\frac{\mathbb{Z}_q[x]}{\langle x^n - \alpha x - \beta \rangle}.$$

In particular, we survey polynomial multiplications in Dilithium, Kyber, NTRU, NTRU Prime, and Saber. All polynomial multiplications takes the form $\mathbb{Z}_q[x]/\langle x^n - \alpha x - \beta \rangle$: We

E-mail: vincentvbh7@gmail.com (Vincent Hwang)

Received: 2023-12-27 Accepted: 2024-06-04

have $\mathbb{Z}_{8380417}[x]/\langle x^{256}+1\rangle$ in Dilithium, $\mathbb{Z}_{3329}[x]/\langle x^{256}+1\rangle$ in Kyber, $\mathbb{Z}_{2^k}[x]/\langle x^n-1\rangle$ with prime $n$ in NTRU, $\mathbb{Z}_q[x]/\langle x^p-x-1\rangle \cong \mathbb{F}_{q^p}$ in NTRU Prime, and $\mathbb{Z}_{2^{13}}[x]/\langle x^{256}+1\rangle$ in Saber. We refer to [ABD+20a, ABD+20b, CDH+20, BBC+20, DKRV20] for the specifications.

## 1.1 Why This Paper

Many works in the literature argue the computational complexity of integer and polynomial multiplications. In practice, practitioners implemented specific approaches and justified the merit of the ideas with numerical evidence. There are no systematic and definite ways to evaluate the practical implications of the ideas since hardware gradually evolves, resulting in floating combinations of implementation considerations. People usually implement the ideas at the assembly-optimized level to determine the best approaches on target hardware. The objective of this paper is to summarize the justifications/implications of the numerical evidence in recent implementation works so when practitioners encounter similar implementational considerations on new platforms in the future, this paper can serve as a collection of practical techniques and guide them toward a convergence of highly optimized implementations on their desired platforms.

## 1.2 Constant-Time Computation

In cryptography, we seek for constant-time computations avoiding timing side-channel attacks. A program is said to be constant-time if (i) the computation is independent from the secret values, (ii) all the memory access are independent from the secret values, and (iii) the instruction timings are independent from the input secret values. In more details, we do not allow secret-dependent branches. This can be overcome with logical operations. We also do not allow looking up a table with a secret index [Ber05]. In this paper, all the mathematical techniques follow the constant-time principal, completely removing the timing side-channel concerns (i) and (ii). As for the concrete implementations, one needs to know the microarchitectural characteristic to determine if (iii) is honored. In the case studies, we will briefly review mitigation techniques when some frequently used multiplication instructions are not constant-time.

## 1.3 Emphases

This paper is written with three emphases: (i) modular arithmetic, (ii) homomorphisms of algebraic structures, and (iii) vectorization.

### 1.3.1 Modular Arithmetic

We survey various modular arithmetic computing representatives of elements in $\mathbb{Z}_q$. Let $\texttt{R}$ be a power of two with exponent a power of two and $q \le \texttt{R}$. We call $\log_2 \texttt{R}$ the width or precision of arithmetic. We only need the cases $\texttt{R} = 2^{16}, 2^{32}$ in this paper. If $q$ is a power of two, then reduction modulo $q$ can be implemented as reduction modulo $\texttt{R}$ and logical and `&`.

If $q$ is not a power of two, then there are two cases: $q$ is an even number with an odd factor, or $q$ is an odd number. We leave the discussion of even $q$ with an odd factor to future work since it is not used in the interested implementations of this paper.

Let's assume $q$ is odd. For $a, b \in \mathbb{Z}_{\texttt{R}}$, there are many ways to compute $c \in \mathbb{Z}_{\texttt{R}}$ with $c \equiv ab \pmod{q}$. The requirement $c \in \mathbb{Z}_{\texttt{R}}$ is to ensure that everything we have at the end can be passed to successive computations with the same width of arithmetic. In practice, we prefer $c \in \mathbb{Z}_{\texttt{B}}$ with $q \le \texttt{B} \le \texttt{R}$ for $\texttt{B}$ reasonably close to $q$. Montgomery multiplication [Mon85] achieves $\texttt{B} = 2q$ and Plantard multiplication [Pla21] achieves

$\mathtt{B} = q + 1$. Both modular multiplications come with multiplicative forms by design. Barrett reduction [Bar86] effectively achieves $\mathtt{B} = 2q$ with $b = 1$, and $\mathtt{B} = q$ while replacing $\mathtt{R}$ with sufficiently large $2^k\mathtt{R}$. [BHK+22b] introduced Barrett multiplication – a multiplicative form of Barrett reduction – and showed that its range is the same as Montgomery multiplication. They introduced the notion "integer approximation" $[\![\,]\!]$ mapping a real number to an integer with a difference bounded by 1, and defined $\bmod^{[\![\,]\!]}$ as

$$\forall a \in \mathbb{Z}, q \left[\!\!\left[ \frac{a}{q} \right]\!\!\right] = a - a \bmod{}^{[\![\,]\!]} q.$$

[BHK+22b] established a correspondence between Montgomery and Barrett multiplications. While Montgomery multiplication is considered exact, Barrett multiplication encompasses various multiplication instructions by interpreting them as high products (multiplication instructions returning the high parts) with integer approximations. Recently, [HKS23] showed that relaxing the condition on $[\![\,]\!]$ enables efficient Barrett multiplication on microcontrollers with limited multiplication instructions. Our survey of modular multiplication is built around the generalization of integer approximations by [HKS23]. We survey integer approximation in Section 2.1, Montgomery multiplication in Section 2.2, Barrett multiplication in Section 2.3, Plantard multiplication in Section 2.4, and compare the modular multiplications in Section 2.5. See Table 1 for an overview.

Table 1: Content of modular multiplications in Section 2.

| Section | Topic | Content |
|---|---|---|
| Section 2.1 | Integer approximation | Integer approximation relating Montgomery, Barrett, and Plantard multiplications. |
| Section 2.2 | Montgomery | Modular multiplication with long/high exact multiplications. |
| Section 2.3 | Barrett | Modular multiplication with approximate high multiplication. |
| Section 2.4 | Plantard | Modular multiplication with long multiplication and middle product. |
| Section 2.5 | Comparisons | Comparisons between Montgomery, Barrett, and Plantard multiplications. |

### 1.3.2 Homomorphisms of Algebraic Structures

This paper involves several notions of algebraic structures and their homomorphisms. An algebraic structure is a set $\mathcal{A}$ of elements equipped with finitely many operations on $\mathcal{A}$. In this paper, there are always identity elements for the operations. Homomorphisms are structure-preserving maps between two algebraic structures – a homomorphism $\eta : \mathcal{A} \to \mathcal{B}$ must satisfy that

$$\forall a, b \in \mathcal{A}, \eta(a \cdot_{\mathcal{A}} b) = \eta(a) \cdot_{\mathcal{B}} \eta(b)$$

for $\cdot_{\mathcal{A}}$ and $\cdot_{\mathcal{B}}$ same type of operations. We call $\eta$ a monomorphism if it is injective. Common algebraic structures are rings, modules, and associative algebras. Associative algebras are algebraic structures that are modules and rings at the same time. For simplicity, we call associative algebra an algebra.

Let $R$ be a unital commutative ring. This paper surveys various algebra homomorphisms implementing the polynomial ring multiplication of $R[x]/\langle x^n - \alpha x - \beta \rangle$ as an algebra. Since algebra homomorphisms are ring and module homomorphisms by definition, we can view them in both ways. In this paper, we always view algebra homomorphisms as module homomorphisms. Suppose we find a way to decompose an algebra homomorphism $\eta$ into a composition of module homomorphisms:

$$\cdots \circ \eta_{i+2} \circ \eta_{i+1} \circ \eta \circ \cdots \circ \eta_{j+2} \circ \eta_{j+1} \circ \eta_j \circ \cdots .$$

We identify the series of module homomorphisms resulting in ring homomorphisms. Such series enable us to multiply the homomorphic images of multiplicands. In practice, this is an interactive process with the target platform – we first write an algebra homomorphism as a composition of module homomorphisms, implement a series of module homomorphisms giving a ring homomorphism, and decide if we want to implement the remaining module homomorphisms, or halt and multiply the images. Therefore, thoroughly exploring the efficiency of module homomorphisms in practice is crucial.

We first review the notations for polynomial rings in Section 3.1 and the definition of discrete Fourier transform (DFT) in Section 3.2. We then survey various "basic homomorphisms," including Cooley–Tukey fast Fourier transform (FFT) in Section 3.3, Good–Thomas FFT in Section 3.4, Bruun-like FFTs in Section 3.5, Rader's FFT in Section 3.6, and Toom–Cook in Section 3.7. Finally, we compare them in Section 3.8. We also use number-theoretic transform (NTT) as a synonym of FFT. See Table 2 for an overview.

Table 2: Basic homomorphisms surveyed in Section 3.

| Section | Topic | Content |
|---|---|---|
| Section 3.1 | Notations | Notations for polynomial rings. |
| Section 3.2 | DFT | Definition of DFT. |
| Section 3.3 | Cooley–Tukey | Factorizing into binomial polynomials. |
| Section 3.4 | Good–Thomas | Factorizing into polynomials with coprime dimensions. |
| Section 3.5 | Bruun | Factorizing into trinomial polynomials. |
| Section 3.6 | Rader | Prime-size factorization. |
| Section 3.7 | Toom–Cook | Evaluation at integers. |
| Section 3.8 | Comparisons | Comparisons between various basic homomorphisms. |

In practice, the target polynomial ring does not always exhibit nice properties defining the "basic homomorphisms." Section 4 surveys various coefficient ring injections adjoining the defining structures, including localization in Section 4.1, Schönhage's and Nussbaumer's FFTs in Section 4.2, and coefficient ring switching in Section 4.3. Finally, we compare localization, Schönhage, Nussbaumer, and coefficient ring switching in Section 4.4. See Table 3 for an overview.

Table 3: Coefficient ring injections surveyed in Section 4.

| Section | Topic | Content |
|---|---|---|
| Section 4.1 | Localization | Adjoin inverses. |
| Section 4.2 | Schönhage/Nussbaumer | Adjoin roots of unity. |
| Section 4.3 | Coefficient ring switching | Adjoin inverses and roots of unity. |
| Section 4.4 | Comparisons | Comparisons between localization, Schönhage, Nussbaumer, and coefficient ring switching. |

We also survey generic optimizations that are closely related to the shape of polynomial modulus, including embedding in Section 5.1, twisting and composed multiplication in Section 5.2, truncation in Section 5.3, incomplete transformation and striding in Section 5.4, and Toeplitz matrix-vector product (TMVP) in Section 5.5. See Table 4 for an overview.

Table 4: Generic optimizations related to the shape of polynomial moduli. We will survey the techniques in Section 5.

| Section | Topic | Content |
|---|---|---|
| Section 5.1 | Embedding/<br>Evaluation at $\infty$ | Decrement the degree of polynomial modulus by one when $\infty$ is not involved. |
| Section 5.2 | Twisting/<br>Composed multiplication | Convert $R[x]/\langle \boldsymbol{g}(x) \rangle$ into $R[y]/\langle x - \zeta y, \boldsymbol{g}(\zeta y) \rangle$ . |
| Section 5.3 | Truncation | Convert a transformation modulo a polynomial modulus into a transformation modulo its factor. |
| Section 5.4 | Incomplete transformation/<br>Striding | Transformation over a substructure. |
| Section 5.5 | TMVP | Convert a transformation multiplying size-$n$ polynomials into a polynomial multiplication for $R[x]/\langle x^n - \alpha x - \beta \rangle$ resulting in small-dimensional TMVPs. |

### 1.3.3 Vectorization

Vectorization is another important topic for highly-optimized assembly implementations. Common vector instruction sets are Neon on Arm Cortex-A processors and SSE/AVX/AVX2/AVX512 on Intel processors. Usually, vector instructions perform a wide variety of permutations and vector-by-vector arithmetic, including additions, subtractions, multiplications, shift operations, and variants. We survey the formalization of vectorization by [Hwa24], including vectorization-friendliness in Section 6.2 and permutation-friendliness in Section 6.3. We also give a short guide for designing vectorized transformations in Section 6.4 based on [Hwa24]. See Table 5 for an overview.

Table 5: Formalization of vectorization.

| Section | Topic | Content |
|---|---|---|
| Section 6.2 | Vectorization-friendliness | Determine mapping to vector-by-vector arithmetic. |
| Section 6.3 | Permutation-friendliness | Determine mapping to vector-by-vector arithmetic and permutations. |
| Section 6.4 | Guide | Determine efficient vectorization with permutation, vector-by-vector, and vector-by-scalar arithemtic. |

## 1.4 Applications of the Techniques and More

To illustrate how the techniques are applied in the literature, we go through some case studies of polynomial multiplications for the lattice-based cryptosystems Dilithium, Kyber, Saber, NTRU, and NTRU Prime. Section 7.1 reviews Barrett and Montgomery multiplications for the NTT/iNTT in Dilithium, Section 7.2 reviews Montgomery and Plantard multiplications for the NTT/iNTT in Kyber, Section 7.3 reviews homomorphism caching for the matrix-vector multiplication in Saber, Section 7.4 reviews the Toeplitz matrix-vector product for NTRU, and Section 7.5 reviews the vectorization of FFTs for NTRU Prime. See Table 6 for an overview.

We then briefly review the advances of the techinques and their applications to Dilithium, Kyber, Saber, NTRU, and NTRU Prime in Section 8, and provide some interesting

Table 6: Overview of case studies.

| Section | Scheme | Content |
|---|---|---|
| Section 7.1 | Dilithium | Barrett and Montgomery multiplications. |
| Section 7.2 | Kyber | Montgomery and Plantard multiplications. |
| Section 7.3 | Saber | Homomorphism caching. |
| Section 7.4 | NTRU | TMVP. |
| Section 7.5 | NTRU Prime | Vectorization. |

directions for implementation projects in Section 9.

## 1.5 How to Read This Paper

This paper is long and condensed. We recommend the readers to pick up specific sections based on their needs with the aid of the introductory section.

## 1.6 Artifact

We provide C implementations for several techniques reviewed in this paper. The source code is available at https://github.com/Polynomial-Multiplications-for-Lattices/Polynomial-Multiplications-for-Lattices.

As hardware gradually improve, we eventually need to transfer the knowledge to incoming platforms. The major goal of the repository is to provide platform-independent implementations for techniques that have found practical importance for easing the development of platform-specific optimizations for future platforms. We expect the repository to be updated in a rolling fashion.

## 1.7 Related Works

There are many survey works targeting polynomial multiplications. We recommend [Win80, Nus82, DV90, Ber01, Ber08] for the underlying mathematical ideas, and [LZ22] for applications to lattice-based cryptography.

## 1.8 Assumed Knowledge

This paper assumes that readers have some basic understandings of commutative algebra. We list the following key words and corresponding references: rings from [Jac12a, Section 2] and [Bou89, Section 8, Chapter I], modules from [Jac12a, Section 3] and [Bou89, Section 1, Chapter II], dual modules from [Jac12b, Example 11, Section 1.3] and [Bou89, Section 2, Chapter II], tensor products of modules from [Jac12b, Section 3.7] and [Bou89, Section 3, Chapter II], associative algebras from [Jac12a, Section 7], [Jac12b, Section 3.9], and [Bou89, Sections 1 and 2, Chapter III], and tensor products of algebras from [Jac12b, Section 3.9] and [Bou89, Section 4].

# 2 Modular Arithmetic

We first survey various modular arithmetic. Section 2.1 generalizes integer approximations for unifying the modular arithmetic used in relevant works. Section 2.2 reviews Montgomery multiplication, Section 2.3 reviews Barrett multiplication, and Section 2.4 reviews Plantard multiplication.

## 2.1 Integer Approximations

For a real number $\delta > 0$ and an integer-valued function $\llbracket\rrbracket : \mathbb{R} \to \mathbb{Z}$, we call $\llbracket\rrbracket$ a $\delta$-integer-approximation [BHK+22b, HKS23] if

$$\forall r \in \mathbb{R}, |\llbracket r \rrbracket - r| \leq \delta.$$

To avoid clutter, we call $\llbracket\rrbracket$ an integer approximation as long as there is a $\delta$ such that $\llbracket\rrbracket$ is a $\delta$-integer-approximation. Furthermore, for a positive integer $q \in \mathbb{Z}_{>0}$, we define the corresponding modular reduction $\bmod^{\llbracket\rrbracket} q : \mathbb{Z} \to \mathbb{Z}$ as

$$\forall z \in \mathbb{Z}, z \bmod^{\llbracket\rrbracket} q = z - \left\llbracket \frac{z}{q} \right\rrbracket q$$

and $\left|\bmod^{\llbracket\rrbracket} q\right| = \max_{z \in \mathbb{Z}} \left|z \bmod^{\llbracket\rrbracket} q\right|$. By definition, we have

$$\forall z \in \mathbb{Z}, \begin{cases} \left\llbracket \frac{z}{q} \right\rrbracket q = z - z \bmod^{\llbracket\rrbracket} q, \\ z \equiv z \bmod^{\llbracket\rrbracket} q \pmod{q}. \end{cases}$$

We illustrate the idea with two examples: the floor function $\lfloor\rfloor$ and the rounding function $\lfloor\rceil \coloneqq r \mapsto \left\lfloor r + \frac{1}{2} \right\rfloor$.

**The floor function $\lfloor\rfloor$.** The floor function $\lfloor\rfloor$ maps a real number to the largest integer lower-bounding the real number. Therefore, for an $r \in \mathbb{R}$, we have $r - 1 < \lfloor r \rfloor \leq r \longrightarrow |\lfloor r \rfloor - r| \leq 1$ and find $\lfloor\rfloor$ a 1-integer-approximation. This function is commonly accompanied by unsigned arithmetic. We denote the corresponding modulo reduction as $\bmod^{\lfloor\rfloor} = \bmod^{+}$ in this case.

**The rounding function $\lfloor\rceil$.** For the round function $\lfloor\rceil$ and an $r \in \mathbb{R}$, since $\lfloor r \rceil = \left\lfloor r + \frac{1}{2} \right\rfloor$ and $r - \frac{1}{2} < \left\lfloor r + \frac{1}{2} \right\rfloor \leq r + \frac{1}{2}$, we find $|\lfloor r \rceil - r| \leq \frac{1}{2}$ and $\lfloor\rceil$ a $\frac{1}{2}$-integer-approximation. If $\lfloor\rceil$ is used for signed arithmetic, we denote the corresponding modulo reduction as $\bmod^{\lfloor\rceil} = \bmod^{\pm}$.

In this paper, we provide a unified view of Montgomery, Barrett, and Plantard multiplication using the pair $\left(\llbracket\rrbracket, \bmod^{\llbracket\rrbracket} q\right)$. Usually, two pairs of integer approximations $\left(\llbracket\rrbracket_0, \bmod^{\llbracket\rrbracket_0} q\right)$ and $\left(\llbracket\rrbracket_1, \bmod^{\llbracket\rrbracket_1} \mathtt{R}\right)$ are involved where $\left(\llbracket\rrbracket_0, \bmod^{\llbracket\rrbracket_0} q\right)$ refers to the one we really want and $\left(\llbracket\rrbracket_1, \bmod^{\llbracket\rrbracket_1} \mathtt{R}\right)$ refers to the practically efficient one.

## 2.2 Montgomery Arithmetic

Let $a, b$ be integers. We wish to compute $ab \bmod^{\llbracket\rrbracket_0} q$ for a $\bmod^{\llbracket\rrbracket_0} q$ with odd $q$. Montgomery multiplication [Mon85, Sei18] computes a representative of $ab \bmod^{\llbracket\rrbracket_1} q$ with possible scaling. Observe that $ab + \left(ab\left(-q^{-1}\right) \bmod^{\llbracket\rrbracket_1} \mathtt{R}\right) q$ is equivalent to 0 modulo $\mathtt{R}$ and $ab$ modulo $q$[1], we have

$$\frac{ab + \left(ab\left(-q^{-1}\right) \bmod^{\llbracket\rrbracket_1} \mathtt{R}\right) q}{\mathtt{R}} \equiv ab\mathtt{R}^{-1} \pmod{q}.$$

To see why this is a reduction, we bound the range as follows:

$$\left|\frac{ab + \left(ab\left(-q^{-1}\right) \bmod^{\llbracket\rrbracket_1} \mathtt{R}\right) q}{\mathtt{R}}\right| \leq \frac{|ab| + \left|\bmod^{\llbracket\rrbracket_1} \mathtt{R}\right| q}{\mathtt{R}}.$$

[1] Since $\mathtt{R} \perp q$, $c \equiv 0 \pmod{\mathtt{R}}$ and $c \equiv ab \pmod{q}$ solve to $c = ab + \left(ab\left(-q^{-1}\right) \bmod^{\llbracket\rrbracket_1} \mathtt{R}\right) q$ by the divided-difference form of the Chinese remainder theorem [CHK+21, Theorem 1]. This was pointed out by [Wan23, YJX24].

There are many ways to mitigate the scaling. A generic way is to perform an additional Montgomery multiplication with $b = \mathtt{R}^2 \bmod {}^{\llbracket\rrbracket_0} q$ for some $\operatorname{mod}^{\llbracket\rrbracket_0} q$. If $b$ is known in prior, we can precompute $b\mathtt{R} \bmod {}^{\llbracket\rrbracket_0} q$ and compute

$$\frac{a\left(b\mathtt{R} \bmod {}^{\llbracket\rrbracket_0} q\right) + \left(a\left(b\mathtt{R} \bmod {}^{\llbracket\rrbracket_0} q\right)\left(-q^{-1}\right) \bmod {}^{\llbracket\rrbracket_1} \mathtt{R}\right) q}{\mathtt{R}} \equiv ab \pmod{q}.$$

Since $b\mathtt{R} \bmod {}^{\llbracket\rrbracket_0} q$ is now bounded by $\left|\operatorname{mod}^{\llbracket\rrbracket_0} q\right|$, we have the following bound:

$$\begin{aligned} & \left|\frac{a\left(b\mathtt{R} \bmod {}^{\llbracket\rrbracket_0} q\right) + \left(a\left(b\mathtt{R} \bmod {}^{\llbracket\rrbracket_0} q\right)\left(-q^{-1}\right) \bmod {}^{\llbracket\rrbracket_1} \mathtt{R}\right) q}{\mathtt{R}}\right| \\ \leq \quad & \frac{|a|\left|\operatorname{mod}^{\llbracket\rrbracket_0} q\right| + \left|\operatorname{mod}^{\llbracket\rrbracket_1} \mathtt{R}\right| q}{\mathtt{R}}. \end{aligned}$$

For unsigned arithmetic with $\operatorname{mod}^{\llbracket\rrbracket_1} \mathtt{R} = \operatorname{mod}^{+} \mathtt{R}$ and $\operatorname{mod}^{\llbracket\rrbracket_0} q = \operatorname{mod}^{+} q$, the range is

$$\frac{|a|\left|\operatorname{mod}^{\llbracket\rrbracket_0} q\right| + \left|\operatorname{mod}^{\llbracket\rrbracket_1} \mathtt{R}\right| q}{\mathtt{R}} \leq q\left(1 + \frac{|a|}{\mathtt{R}}\right).$$

For signed arithmetic with $\operatorname{mod}^{\llbracket\rrbracket_1} \mathtt{R} = \operatorname{mod}^{\pm} \mathtt{R}$ and $\operatorname{mod}^{\llbracket\rrbracket_0} q = \operatorname{mod}^{\pm} q$, the resulting range is

$$\frac{|a|\left|\operatorname{mod}^{\llbracket\rrbracket_0} q\right| + \left|\operatorname{mod}^{\llbracket\rrbracket_1} \mathtt{R}\right| q}{\mathtt{R}} \leq \frac{q}{2}\left(1 + \frac{|a|}{\mathtt{R}}\right).$$

**Historical review.** [Mon85] proposed the unsigned Montgomery multiplication, and [Sei18] later proposed the signed variant along with the subtractive variant:

$$\frac{ab - \left(abq^{-1} \bmod {}^{\pm}\mathtt{R}\right) q}{\mathtt{R}}.$$

The benefit of the subtractive variant is that $(ab \bmod {}^{\pm}\mathtt{R}) - \left(\left(abq^{-1} \bmod {}^{\pm}\mathtt{R}\right) q \bmod {}^{\pm}\mathtt{R}\right) = 0$ whereas $(ab \bmod {}^{\pm}\mathtt{R}) + \left(\left(ab - q^{-1} \bmod {}^{\pm}\mathtt{R}\right) q \bmod {}^{\pm}\mathtt{R}\right) = 0$ or $\mathtt{R}$ as integers. The former implies the following computation:

$$\left\lfloor \frac{ab}{\mathtt{R}} \right\rfloor - \left\lfloor \frac{\left(abq^{-1} \bmod {}^{\pm}\mathtt{R}\right) q}{\mathtt{R}} \right\rfloor.$$

This replaces double-size products with high products. See [KAK96, KA98] for the multi-limb versions.

## 2.3 Barrett Arithmetic

Let $\llbracket\rrbracket_0, \llbracket\rrbracket_1$ be integer approximations. Barrett multiplication computes

$$ab - \left\llbracket \frac{a\left\llbracket \frac{b\mathtt{R}}{q} \right\rrbracket_0}{\mathtt{R}} \right\rrbracket_1 q \equiv ab \pmod{q}.$$

Obviously, this is a representative of $ab \bmod q$. The only question is if the resulting range falls into the data width. [BHK<sup>+</sup>22b] showed the following correspondence

$$ab - \left\llbracket \frac{a\left\llbracket \frac{b\mathtt{R}}{q} \right\rrbracket_0}{\mathtt{R}} \right\rrbracket_1 q = \frac{a\left(b\mathtt{R} \operatorname{mod}^{\llbracket\rrbracket_0} q\right) + \left(a\left(b\mathtt{R} \operatorname{mod}^{\llbracket\rrbracket_0} q\right)\left(-q^{-1}\right) \operatorname{mod}^{\llbracket\rrbracket_1} \mathtt{R}\right) q}{\mathtt{R}}$$

and obtained the bound

$$\left| ab - \left\llbracket \frac{a \left\llbracket \frac{b\mathtt{R}}{q} \right\rrbracket_0}{\mathtt{R}} \right\rrbracket_1 q \right| \le \frac{|a| \left| \bmod^{\llbracket\rrbracket_0} q \right| + \left| \bmod^{\llbracket\rrbracket_1} \mathtt{R} \right| q}{\mathtt{R}}.$$

In Appendix A, we prove the correspondence for principal ideal domains where $\mathbb{Z}$ and $\mathbb{F}[x]$ are popular special cases.

**Comparing Montgomery and Barrett multiplications.** Since the absolute value of the result is smaller than $\frac{\mathtt{R}}{2}$ for signed arithmetic ($\mathtt{R}$ for unsigned arithmetic) in practice, we only need to compute $ab \bmod {}^{\pm}\mathtt{R}$ ($ab \bmod {}^{+}\mathtt{R}$ for unsigned arithmetic) instead of the full product. Same observation holds for $\left\llbracket \frac{a \left\llbracket \frac{b\mathtt{R}}{q} \right\rrbracket_0}{\mathtt{R}} \right\rrbracket_1 q$. Therefore, Barrett multiplication only requires one to compute a high product implementing $\left\llbracket \frac{a \left\llbracket \frac{b\mathtt{R}}{q} \right\rrbracket_0}{\mathtt{R}} \right\rrbracket_1$ and two low-products multiplying in $\bmod^{\pm}\mathtt{R}$ or $\bmod^{+}\mathtt{R}$. On the other hand, one has to compute two full products (or high products for the subtractive variant) and one low-product for Montgomery multiplication. [BHK+22b] saved one subtraction with Barrett multiplication since there is a subtractive variant for low-product and not high product.

**Historical review.** For unsigned arithmetic, [Bar86] proposed the case $b = 1$, and [Sho] proposed Barrett multiplication for generic $b$. The signed version and its correspondence to Montgomery multiplication was discovered by [BHK+22b]. Interestingly, [Dhe03] proposed the finite field version. Appendix A proves the correspondence for principal ideal domains, and the impact for finite fields is left for future investigation. Recently, [BHK+22a, Section 2.4] improved the output range for $b \neq 1$ while replacing $\mathtt{R}$ for some $2^k\mathtt{R}$, and [HKS23] furthered the approximation nature of $\llbracket\rrbracket_1$ and improved the modular multiplications on microcontrollers.

## 2.4 Plantard Arithmetic

Recently, [Pla21] proposed an unsigned modular multiplication essentially with precision $2\log_2 \mathtt{R}$. The signed versions were later proposed by [HZZ+22, AMOT22]. For multiplying an integer $a$ by a constant $b$ known in prior, Montgomery multiplication results in the bound $\frac{|a|\left| \bmod^{\llbracket\rrbracket_0} q\right| + \left| \bmod^{\llbracket\rrbracket_1} \mathtt{R}\right| q}{\mathtt{R}}$. If we replace the precision $\log_2 \mathtt{R}$ with $2\log_2 \mathtt{R}$ and compute with

$$\frac{a\left(b\mathtt{R}^2 \bmod^{\llbracket\rrbracket_0} q\right) + \left(a\left(b\mathtt{R}^2 \bmod^{\llbracket\rrbracket_0} q\right)\left(-q^{-1}\right) \bmod^{\llbracket\rrbracket_1} \mathtt{R}^2\right) q}{\mathtt{R}^2},$$

we have the bound

$$\frac{|a| \left| \bmod^{\llbracket\rrbracket_0} q \right| + \left| \bmod^{\llbracket\rrbracket_1} \mathtt{R}^2 \right| q}{\mathtt{R}^2}.$$

For signed arithmetic with $\left| \bmod^{\llbracket\rrbracket_1} \mathtt{R}^2 \right| \le \frac{\mathtt{R}^2}{2}$ and $\left| \bmod^{\llbracket\rrbracket_0} q \right| \le \frac{q}{2}$, the bound is $\frac{q}{2}\left(1 + \frac{|a|}{\mathtt{R}^2}\right)$. In practice, since $|a| \le \mathtt{R}$ and $q < \mathtt{R}$, the result is stricly smaller than $\frac{q}{2}$, and hence an integer in $\left\{-\frac{q-1}{2}, \dots, 0, \dots, \frac{q-1}{2}\right\}$.

We borrow the integer-approximation view from [HKS23] and proceed with [Pla21]'s innovation for implementing the above observation. Suppose we find two integer approximations $[\![]\!]_2$ and $[\![]\!]_3$ implementing:

$$\begin{aligned} & \frac{c + \left(c\left(-q^{-1}\right) \bmod^{[\![]\!]_1} \mathtt{R}^2\right) q}{\mathtt{R}^2} \\ = \quad & \left[\!\!\left[ \frac{c + \left(c\left(-q^{-1}\right) \bmod^{[\![]\!]_1} \mathtt{R}^2\right) q}{\mathtt{R}^2} - \frac{c + \left(c\left(-q^{-1}\right) \bmod^{[\![]\!]_1} \mathtt{R}^2 \bmod^{[\![]\!]_2} \mathtt{R}\right) q}{\mathtt{R}^2} \right]\!\!\right]_3 \end{aligned}$$

for all $c \in \mathbb{Z}_{\mathtt{B}}$ with $\mathtt{B}$ sufficiently close to $\mathtt{R}^2$, we claim the following:

$$\frac{c + \left(c\left(-q^{-1}\right) \bmod^{[\![]\!]_1} \mathtt{R}^2\right) q}{\mathtt{R}^2} = \left[\!\!\left[ \frac{\left[\!\!\left[\frac{c\left(-q^{-1}\right) \bmod^{[\![]\!]_1} \mathtt{R}^2}{\mathtt{R}}\right]\!\!\right]_2 q}{\mathtt{R}} \right]\!\!\right]_3 .$$

The proof is left as an exercise[2]. If $c = ab$, we can instead precompute $b\left(-q^{-1}\right) \bmod^{[\![]\!]_1} \mathtt{R}^2$, and apply one middle product followed by one high product. While $z \mapsto \left[\!\left[\frac{zq}{\mathtt{R}}\right]\!\right]_3$ is the usual high product multiplying numbers of precision $\log_2 \mathtt{R}$, the high product $z \mapsto \left[\!\!\left[\frac{zb\left(-q^{-1}\right) \bmod^{[\![]\!]_1} \mathtt{R}^2}{\mathtt{R}}\right]\!\!\right]_2$ requires one to multiply $a$ by a number with precision $2\log_2 \mathtt{R}$. [HZZ+22] identified the use case in Armv7E-M implementing the multiplication instructions `smulw{b, t}`[3], and [AMOT22, Source code 1] implemented the idea when only multiplication instructions with precision $2\log_2 \mathtt{R}$ are available.

## 2.5 Comparisons

We briefly review the required multiplication instructions with precision $\log_2 \mathtt{R}$. We categorize multiplication instructions into three groups:

- Low multiplications: `mullo` computes the lower $\log_2 \mathtt{R}$ bits of the product, `mlalo` computes the lower $\log_2 \mathtt{R}$ bits of the product and accumulate them to a register with $\log_2 \mathtt{R}$-bit precision, and `mlslo` subtract the product from the register with $\log_2 \mathtt{R}$-bit precision.

- High multiplications: `mulhi` computes the upper $\log_2 \mathtt{R}$ bits of the product within a reasonable approximation, `mlahi` is the accumulative variant, and `mlshi` is the subtractive variant.

- Long multiplications: `mull` computes the full-size product with $2\log_2 \mathtt{R}$ bits of precision, `mlal` is the accumulative variant, and `mlsl` is the subtractive variant.

See Table 7 for an overview.

**Montgomery multiplication.** For Montgomery multiplication, we need one `mull`, one `mullo`, and one `mlal` (in this order) as seen in Section 2.2. As for the subtractive variant, we need on `mulhi`, one `mullo`, and one `mlshi`.

**Barrett multiplication.** For Barrett multiplication, we need one `mullo`, one `mulhi`, and one `mlslo` since we only need the lower $\log_2 \mathtt{R}$ bits of the difference of the lower products (cf. Section 2.3).

[2] Hint: cancel out the terms $\frac{c}{\mathtt{R}}$, write the remaing as a multiple of $\frac{q}{\mathtt{R}}$, and rewrite the rest with $[\![]\!]_2$.

[3] `w` stands for a word and `{b, t}` stands for the bottum or the top half-word.

**Plantard multiplication.** For Plantard multiplication, we need a middle product computing the middle $\log_2 \mathtt{R}$ bits of a product of a $\log_2 \mathtt{R}$-bit number and a $2\log_2 \mathtt{R}$-bit number. Middle product can be implemented with one `mulhi` and one `mlalo`. As for the multiplication followed by $[\![\,]\!]_3$, experiment shows that we have to add up the full-size product and certain constant with absolute value $\leq \frac{\mathtt{R}}{2}$ prior to applying $[\![\,]\!]_3$, so we count the last multiplication as an `mlal` (cf. Section 2.4).

Table 7: Overview of required multiplication instructions of Montgomery, Barrett, and Plantard multiplications. Montgomery (acc.) stands for the accumulative variant and Montgomery (sub.) stands for the subtractive variant. In practice, if the corresponding multiplication instructions are already supported by the target platform and run in constant-time, then one implements the modular multiplications with them. On the other hand, if some multiplication instructions are missing or not constant-time, then one should emulate them with the multiplication instructions supported by the target platform. This usually incurs large overhead. We'll review examples for 32-bit modular multiplications in Section 7.1.

| | `mullo` | `mlalo` | `mlslo` | `mulhi` | `mlshi` | `mull` | `mlal` |
|---|---|---|---|---|---|---|---|
| Montgomery (acc.) | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| Montgomery (sub.) | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| Barrett | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| Plantard | 0 | 1 | 0 | 1 | 0 | 0 | 1 |

# 3 Basic Homomorphisms

We survey several homomorphisms that are frequently used for constructing large transformations. Section 3.2 reviews discrete Fourier transform, Section 3.3 reviews Cooley–Tukey FFT, Section 3.4 reviews Good–Thomas FFT, Section 3.5 reviews Bruun's FFT, Section 3.6 reviews Rader's FFT, Section 3.7 reviews Karatsuba and Toom–Cook, and Section 3.8 compares the domains, images, and defining conditions.

## 3.1 Notations

For a ring $R$, we denote $R[x]$ the polynomial ring with indeterminate $x$ and coefficients in $R$. For a polynomial $\boldsymbol{g} \in R[x]$, we denote $\langle \boldsymbol{g} \rangle := \boldsymbol{g}R[x] \subset R[x]$ the ideal generated by $\boldsymbol{g}$ and $R[x]/\langle \boldsymbol{g} \rangle$ the quotient ring. If $\boldsymbol{g} = x^n$ for a positive integer $n$, we also denote the quotient ring $R[x]/\langle x^n \rangle$ as $R[x]_{<n}$.

**Maps and homomorphisms.** For two sets $S_0$ and $S_1$, we denote $S_0 \to S_1$ the signature of a map from $S_0$ to $S_1$. If the map is injective, we write $S_0 \hookrightarrow S_1$; and if the map is surjective, we write $S_0 \twoheadrightarrow S_1$. If the map is injective and surjective, we call it bijective and write $S_0 \cong S_1$. For two rings/modules/algebras $S_0$ and $S_1$ and a map $f : S_0 \to S_1$, we call $f$ ring/module/algebra homomorphism if it preserves the underlying operations defining the algebraic structure. For a ring/module/algebra homomorphism, we call it monomorphism if it is injective, epimorphism if it is surjective, and isomorphism if it is bijective.

**Products.** For a positive integer $n$, and rings $R_0, \ldots, R_{n-1}$, we denote $\prod_{0 \leq i < n} R_i$ for the product ring of $R_0, \ldots, R_{n-1}$. Its elements are denoted as $n$-tuples. When the context is clear, we simply write $\prod_i$ where $i$ runs over all possible values.

## 3.2 Discrete Fourier Transform

We first review the discrete Fourier transform (DFT). Essentially, DFT is a special case of the Chinese remainder theorem (CRT) for polynomial rings. For a ring $R$, a positive integer $n$, and an $n$-th root of unity $\omega_n \in R$. We call $\omega_n$ principal $n$-th root of unity if

$$\forall j = 1, \dots, n-1, \sum_{0 \le i < n} \omega_n^{ij} = 0.$$

The size-$n$ DFT refers to the following isomorphism:

$$\begin{cases} \dfrac{R[x]}{\langle x^n - 1 \rangle} & \to & \displaystyle\prod_{0 \le i < n} \dfrac{R[x]}{\langle x - \omega_n^i \rangle} \\ \boldsymbol{a}(x) & \mapsto & \left( \boldsymbol{a}\left(\omega_n^i\right) \right)_{0 \le i < n} \end{cases}$$

with the inverse

$$\begin{cases} \displaystyle\prod_{0 \le i < n} \dfrac{R[x]}{\langle x - \omega_n^i \rangle} & \to & \dfrac{R[x]}{\langle x^n - 1 \rangle} \\ (\hat{a}_i)_{0 \le i < n} & \mapsto & \displaystyle\sum_{0 \le i < n} \boldsymbol{r}_i \hat{a}_i \end{cases}$$

where $\boldsymbol{r}_i \coloneqq \frac{1}{n} \sum_{0 \le j < n} \omega_n^{-ij} x^j$. The correctness follows from the definition of principal $n$-th root of unity.

For an invertible element $\zeta \in R$, discrete weighted transform (DWT) generalizes DFT into an isomorphism between $R[x]/\langle x^n - \zeta^n \rangle$ and $\prod_{0 \le i < n} R[x]/\langle x - \zeta\omega_n^i \rangle$ where the $\boldsymbol{r}_i \coloneqq \frac{1}{n} \sum_{0 \le j < n} \zeta^{-j} \omega_n^{-ij} x^j$ in the inversion map [CF94]. We call it cyclic when $\zeta^n = 1$ and negacyclic when $\zeta^n = -1$.

In summary, we need three conditions for defining an invertible DWT for $R[x]/\langle x^n - \zeta^n \rangle$:

- The positive integer $n$ must be invertible in $R$. Notice that positive integers are encoded as repeat additions of the identity of $R$, and negative integers are encoded as repeat additions of the additive inverse of the identity of $R$.

- The element $\zeta$ must be invertible in $R$.

- There must exist a principal $n$-th root of unity. When $n$ is a power of two, the condition is equivalent to $\omega_n^{\frac{n}{2}} = -1 \in R$ [Für09]. In Appendix B, we show that the condition $\Phi_n(\omega_n) = 0$ suffices where $\Phi_n$ is the $n$-th cyclotomic polynomial, the unique irreducible polynomial in $\mathbb{Z}[x]$ that is a divisor of $x^n - 1$ and not a divisor of $x^d - 1$ for any positive integer $d < n$.

**Historical review of the conditions.** For defining a DFT of size-$n$, [Pol71] showed that $n$ must be a divisor of $q-1$ if $R = \mathbb{F}_q$ and $p-1$ if $R = \mathbb{Z}_{p^k}$ for a prime $p$. The latter says that for $R = \mathbb{Z}_m$ with prime factorization $m = \prod_i p_i^{d_i}$, $n$ must divide $\gcd(p_i - 1)$ [Pol71, AB74]. [DV78b, Theorem 4] showed the condition when $R$ is a product of local rings[4], and [Für09, Section 2] showed that a principal $n$-th root of unity suffices. The cyclotomic condition was used in [SS71] and stated in [Für09] for a power-of-two $n$. The proof in [Für09] naturally generalizes to a prime-power $n$. For the general case, we can't find such a statement in the literature and therefore, present it in Appendix B.

[4] A local ring is a ring with a unique maximal left/right-ideal.

### 3.3 Cooley–Tukey Fast Fourier Transform

For the DFT implementing $R[x]/\langle x^n - \zeta^n\rangle \cong \prod_{0\leq i<n} R[x]/\left\langle x - \zeta\omega_n^i\right\rangle$, Cooley–Tukey FFT improves the computation when $n$ admits a factorization $\prod_{0\leq j<h} n_j$. We define

$$\boldsymbol{g}_{i_0,\ldots,i_{h-1}} \coloneqq x - \zeta\omega_n^{\sum_j i_j \prod_{l<j} n_l}$$

for all $0 \leq i_j < n_j$ and find $x^n - \zeta^n = \prod_{i_0,\ldots,i_{h-1}} \boldsymbol{g}_{i_0,\ldots,i_{h-1}}$. Since all the $\boldsymbol{g}_{i_0,\ldots,i_{h-1}}$'s are coprime, we have the following series of isomorphisms:

$$\frac{R[x]}{\langle x^n - \zeta^n\rangle} = \frac{R[x]}{\left\langle \prod_{i_0,\ldots,i_{h-1}} \boldsymbol{g}_{i_0,\ldots,i_{h-1}}\right\rangle} \cong \prod_{i_0} \frac{R[x]}{\left\langle \boldsymbol{g}_{i_0,\ldots,i_{h-1}}\right\rangle} \cong \cdots \cong \prod_{i_0,\ldots,i_{h-1}} \frac{R[x]}{\left\langle \boldsymbol{g}_{i_0,\ldots,i_{h-1}}\right\rangle}.$$

**Real-world example(s).** In Dilithium, one is asked to implement the radix-2 FFT defined on $\mathbb{Z}_{8380417}[x]/\left\langle x^{256}+1\right\rangle$. Since $x^{256}+1 = \Phi_{512}(x)$, the defining condition is the same for $\mathbb{Z}_{8380417}[x]/\left\langle x^{512}-1\right\rangle$. Observe that $8380417 = 2^{13}\cdot 3\cdot 11\cdot 31 + 1$, we can define a cyclic FFT with transformation size a divisor of $2^{13}\cdot 3\cdot 11\cdot 31$. This gives the isomorphism $\mathbb{Z}_{8380417}[x]/\left\langle x^{512}-1\right\rangle \cong \prod_i \mathbb{Z}_{8380417}[x]/\left\langle x - \omega_{512}^i\right\rangle$ and hence $\mathbb{Z}_{8380417}[x]/\left\langle x^{256}+1\right\rangle \cong \prod_i \mathbb{Z}_{8380417}[x]/\left\langle x - \omega_{512}^{2i+1}\right\rangle$ by choosing $\zeta = \omega_{512}$ (any odd power of $\omega_{512}$ works) and $\omega_{256} = \omega_{512}^2$.

### 3.4 Good–Thomas FFT

Good–Thomas FFT exploits the factorization of $n = \prod_{0\leq j<d} n_j$ when $n_j$'s are coprime to each other [Goo58]. For a cyclic size-$n$ DFT implementing $R[x]/\langle x^n - 1\rangle \cong \prod_{0\leq i<n} R[x]/\left\langle x - \omega_n^i\right\rangle$, we define principal $n_j$-th root of unity $\omega_{n_j}$ as $\omega_n^{e_j}$ for all $j$ where $(e_j)_{0\leq j<d}$ is the unique tuple of positive integers satisfying $1 \equiv \sum_{0\leq j<n} e_j \pmod{n}$ so $\omega_n = \omega_n^{\sum_{0\leq j<d} e_j} = \prod_{0\leq j<d} \omega_{n_j}$. If we rewrite the result of DFT as

$$\sum_{0\leq i<n} a_i\omega_n^i = \sum_{i_0,\ldots,i_{d-1}} a_i \prod_{0\leq j<d} \omega_{n_j}^{i_j} = \sum_{i_0}\cdots\sum_{i_{d-1}} a_i \prod_{0\leq j<d} \omega_{n_j}^{i_j}$$

where $i_j = i \bmod n_j$, we find that the right-hand side is a multi-dimensional cyclic DFT. In the language of polynomial rings, the cyclic size-$n$ DFT is implemented as the following multi-dimensional cyclic DFT:

$$\begin{aligned}
\frac{R[x]}{\langle x^n - 1\rangle} &\cong \frac{R[x_0,\ldots,x_{d-1}]}{\left\langle x - \prod_j x_j, x_0^{n_0}-1,\ldots,x_{d-1}^{n_{d-1}}-1\right\rangle} \\
&\cong \prod_{i_0,\ldots,i_{d-1}} \frac{R[x_0,\ldots,x_{d-1}]}{\left\langle x - \prod_j x_j, x_0 - \omega_{n_0}^{i_0},\ldots,x_{d-1} - \omega_{n_{d-1}}^{i_{d-1}}\right\rangle} \\
&\cong \prod_i \frac{R[x]}{\langle x - \omega_n^i\rangle}.
\end{aligned}$$

The overall asymptotic run-time is the same as Cooley–Tukey, but we save linearly number of multiplications in $R$. We review the algebraic view of Good–Thomas FFT in Appendix C, and vector-radix FFT in Appendix D further improving the multi-dimensional transformation.

**Real-world example(s).** In [AHY22], they computed the products in $R[x]/\left\langle x^{1536}-1\right\rangle$ via Good–Thomas. They first introduced $x^4 \sim x_0x_1, x_0^3 \sim 1$, and $x_1^{128} \sim 1$ for vectorization-friendliness, and observed that vectorization-friendliness implies a flexible code-size optimization while permuting for Good–Thomas [AHY22, Sections 3.2 and 3.3]. We will formally review the notion of vectorization-friendliness in Section 6.2.

## 3.5 Bruun-Like Fast Fourier Transforms

After the introduction of Cooley–Tukey FFT over complex numbers, many works proposed several optimizations if the input coefficients are real. [Bru78] proposed Bruun's FFT for the power-of-two case, [DH84] proposed split-radix FFT, [Bra84] proposed fast Hartley transform for the discrete Hartley transform (DHT) [Har42][5], [Mur96] generalized Bruun's FFT to arbitrary even sizes, and [JF07, Ber07, LVB07] improved the split-radix FFT.

This section reviews the works [Bru78, Mur96] over complex numbers for historical reasons. However, the actual use case relevant to us are the factorization of cyclotomic polynomials over finite fields [BC87, BGM93, Mey96]. See [TW13, BMGVdO15, WYF18, WY21] for recent progresses on this topic.

**The complex case.** Let $n_j = |\mathcal{I}_j|$, $n = \prod_j n_j$, $\xi, \zeta \in \mathbb{C}$ be invertible elements, and $\omega_n \in \mathbb{C}$ a principal $n$-th root of unity. Bruun's FFT [Bru78, Mur96] chooses $\boldsymbol{g}_{i_0,\dots,i_{h-1}}$ as follows:

$$\boldsymbol{g}_{i_0,\dots,i_{h-1}} = x^2 - \left(\xi\omega_n^{\sum_j i_j \prod_{l<j} n_l} + \xi^{-1}\omega_n^{-\sum_j i_j \prod_{l<j} n_l}\right)\zeta x + \zeta^2.$$

If $\boldsymbol{g}_{i_0,\dots,i_{h-1}}$'s are coprime (namely, $\xi \neq \xi^{-1}$ in the complex case), we have a fast transformation for the ring $R[x]/\left\langle x^{2n} - (\xi^n + \xi^{-n})\,\zeta^n x^n + \zeta^{2n}\right\rangle$ since $\prod_{i_0,\dots,i_{h-1}} \boldsymbol{g}_{i_0,\dots,i_{h-1}} = x^{2n} - (\xi^n + \xi^{-n})\,\zeta^n x^n + \zeta^{2n}$. For $\zeta = 1, \xi = \omega_{4n} \in \mathbb{C}$, this implements the isomorphism $\mathbb{C}[x]/\left\langle x^{2n}+1\right\rangle \cong \prod_i \mathbb{C}[x]/\left\langle x - \omega_{4n}^{1+2i}\right\rangle$ if we further split into linear factors.

**The finite field cases.** In this paper, we are interested in the case $R = \mathbb{F}_q$ with $q \equiv 3 \pmod 4$ which relies on the following theorem from [BGM93]:

**Theorem 1** ([BGM93])**.** *Let $q \equiv 3 \pmod 4$ be a prime and $2^w$ be the highest power of $q+1$. For $k < w$, $x^{2^k}+1$ factors into irreducible trinomials $x^2 + \gamma x + 1 \in \mathbb{F}_q[x]$. For $k \geq w$, $x^{2^k}+1$ factors into irreducible trinomials $x^{2^{k-w+1}} + \gamma x^{2^{k-w}} - 1 \in \mathbb{F}_q[x]$.*

**Real-world example(s).** For the NTRU Prime parameter sets `ntrulpr761/sntrup761`, [HLY24] introduced a fast transformation (`Good-Schönhage-Bruun`) leading to computing in $\mathbb{Z}_{4591}[x]/\left\langle x^{32}+1\right\rangle$. Since $4591 \equiv 3 \pmod 4$ and $4591 + 1 = 287 \cdot 2^4$, we can split $\mathbb{Z}_{4591}[x]/\left\langle x^{32}+1\right\rangle$ into polynomial rings modulo trinomials of the form $x^4 + \gamma x^2 - 1$. [HLY24] split into rings of the form $\mathbb{Z}_{4591}[x]/\left\langle x^8 + \alpha x^4 + 1\right\rangle$ for efficiency reasons.

## 3.6 Rader's Fast Fourier Transform

Let $n$ be a positive integer, $\mathcal{I} = \{0, \dots, n-1\}$, and $\omega_n \in R$ be a principal $n$-th root of unity. If $n$ is an odd prime, Rader's FFT computes the map $\boldsymbol{a} \mapsto \left(\boldsymbol{a}(\omega_n^i)\right)_{i\in\mathcal{I}}$ via a size-$(n-1)$ cyclic convolution [Rad68]. See Appendix E for generalization.

We explain the idea for an odd prime $n$. Let $\mathcal{I}^* \coloneqq \{1, \dots, n-1\}$ be an index set, $(a_j)_{j\in\mathcal{I}} \coloneqq \boldsymbol{a}$, and $(\hat{a}_i)_{i\in\mathcal{I}} \coloneqq \left(\boldsymbol{a}(\omega_n^i)\right)_{i\in\mathcal{I}}$. Since $n$ is prime, there is a $g \in \mathcal{I}$ with $\left\{g^k \in \mathcal{I} \middle| k \in \mathbb{Z}_{n-1}\right\} = \mathcal{I}^*$ where the powers $g^k$ are reduced modulo $n$. We introduce the reindexing $j \in \mathcal{I}^* \mapsto -\log_g j \in \mathbb{Z}_{n-1}$ and $i \in \mathcal{I}^* \mapsto \log_g i \in \mathbb{Z}_{n-1}$ where $\log_g$ is the discrete logarithm, and split the computation $(a_j)_{j\in\mathcal{I}} \mapsto (\hat{a}_i)_{i\in\mathcal{I}}$ into $\hat{a}_0 = \sum_{j\in\mathcal{I}} a_j$ and $\hat{a}_i = a_0 + \sum_{j\in\mathcal{I}^*} a_j \omega_n^{ij}$ for $i \in \mathcal{I}^*$. For the cases $i \in \mathcal{I}^*$, we move $a_0$ to the left-hand side, and rewrite it as

$$\hat{a}_{g^{\log_g i}} - a_0 = \sum_{j\in\mathcal{I}^*} a_j \omega_n^{ij} = \sum_{-\log_g j \in \mathbb{Z}_{n-1}} a_{g^{\log_g j}} \omega_n^{g^{\log_g i + \log_g j}}.$$

[5] One can derive DFT and DHT from each other with linearly number of arithmetic during post-processing. Therefore, improvement for one of them transfers to the other one.

We can now compute $\left(\hat{a}_{g^k} - a_0\right)_{k \in \mathbb{Z}_{n-1}}$ as the size-$(n-1)$ cyclic convolution of $\left(a_{g^{-k}}\right)_{k \in \mathbb{Z}_{n-1}}$ and $\left(\omega_n^{g^k}\right)_{k \in \mathbb{Z}_{n-1}}$. We give an example for $n = 5$ and $g = 2$:

$$\left(\hat{a}_{2^k} - a_0\right)_{k \in \mathbb{Z}_5^*} = \begin{pmatrix} a_1\omega_5^2 + a_2\omega_5^4 + a_3\omega_5^1 + a_4\omega_5^3 \\ a_1\omega_5^4 + a_2\omega_5^3 + a_3\omega_5^2 + a_4\omega_5^1 \\ a_1\omega_5^3 + a_2\omega_5^1 + a_3\omega_5^4 + a_4\omega_5^2 \\ a_1\omega_5^1 + a_2\omega_5^2 + a_3\omega_5^3 + a_4\omega_5^4 \end{pmatrix} = \begin{pmatrix} a_{2^4}\omega_5^{2^1} + a_{2^3}\omega_5^{2^4} + a_{2^2}\omega_5^{2^3} + a_{2^1}\omega_5^{2^2} \\ a_{2^4}\omega_5^{2^2} + a_{2^3}\omega_5^{2^1} + a_{2^2}\omega_5^{2^4} + a_{2^1}\omega_5^{2^3} \\ a_{2^4}\omega_5^{2^3} + a_{2^3}\omega_5^{2^2} + a_{2^2}\omega_5^{2^1} + a_{2^1}\omega_5^{2^4} \\ a_{2^4}\omega_5^{2^4} + a_{2^3}\omega_5^{2^3} + a_{2^2}\omega_5^{2^2} + a_{2^1}\omega_5^{2^1} \end{pmatrix}.$$

**Real-world example(s).** The case $(n, g) = (17, 3)$ was used for multiplying over $\mathbb{Z}_{4591}$. [ACC+21] multiplied in $\mathbb{Z}_{4591}[x]/\left\langle x^{1530} - 1\right\rangle$ on Cortex-M4, [HLY24] multiplied in $\mathbb{Z}_{4591}[x]/\left\langle x^{1632} - 1\right\rangle$ with Armv8.0-A Neon, and [Hwa24] multiplied in $\mathbb{Z}_{4591}[x]/\left\langle \Phi_{17}(x^{96})\right\rangle$ with Armv8.0-A Neon and Intel AVX2. Observe $1530 = 17 \cdot 90$ and $1632 = 17 \cdot 96$, their implementations relied on the size-17 cyclic FFT $\mathbb{Z}_{4591}[x]/\left\langle x^{17} - 1\right\rangle \cong \prod_i \mathbb{Z}_{4591}[x]/\left\langle x - \omega_{17}^i\right\rangle$, and are implemented with Rader's FFT. We will shortly review how [Hwa24] applied Rader's FFT for $\mathbb{Z}_{4591}[x]/\left\langle \Phi_{17}(x^{96})\right\rangle$ in Section 5.3.

## 3.7 Karatsuba and Toom–Cook

Let $\mathcal{I} = \{0, \dots, 2n-2\}$ and $\{s_i\}_{i \in \mathcal{I}} \subset \mathbb{Z}$ be a finite set. Karatsuba [KO62] and Toom–Cook [Too63] compute the size-$(2n-1)$ product of two size-$n$ polynomials with the maps $R[x]_{<n} \hookrightarrow R[x]/\left\langle \prod_{i \in \mathcal{I}} (x - s_i)\right\rangle \cong \prod_{i \in \mathcal{I}} R[x]/\langle x - s_i\rangle$. [KO62] proposed the case $n = 2$ with the point set $\{0, 1, \infty\}$, [Too63] chose $n \geq 2$ and $\{s_i\} \subset \mathbb{Z}$, and [Win80] extended the choice of $\{s_i\}$ to $\mathbb{Q} \cup \{\infty\}$. Let $c \in \mathbb{Z}$. Evaluating $x$ at $c^{-1}$ means mapping a polynomial $\boldsymbol{a}(x)$ to $c^{\deg(\boldsymbol{a})}\boldsymbol{a}(c^{-1})$ instead of $\boldsymbol{a}(c^{-1})$. We will review the idea of evaluating at $\infty$ in Section 5.1, and localization adjoining the inverses of integers in Section 4.1.

## 3.8 Comparisons

We briefly compare Cooley–Tukey, Good–Thomas, Bruun, Rader, and Toom–Cook. Table 8 summarizes the domains and images, and Table 9 summarizes the defining conditions.

**Cooley–Tukey vs Good–Thomas.** Both Cooley–Tukey and Good–Thomas relies on a factorization of the dimension $n$, the existence of a principal $n$-th root of unity, and the existence of $n^{-1}$ in the coefficient ring. While Cooley–Tukey works for arbitrary factorization of $n$, Good–Thomas relies on a coprime factorization. As for the shape of polynomial modulus, Cooley–Tukey is definable on $R[x]/\langle x^n - \zeta^n\rangle$, and Good–Thomas reviewed in Section 3.4 is definable only on $R[x]/\langle x^n - 1\rangle$. Generally speaking, if the order of $\zeta$ is coprime to $n$, we can also define Good–Thomas on $R[x]/\langle x^n - \zeta^n\rangle$ via truncation as illustrated in [HVDH22, Sections 3.5 and 3.6]. See Appendix C for more explanations. If both approaches are definable, Good–Thomas saves linearly number of multiplications in $R$.

**Cooley–Tukey vs Bruun.** While Cooley–Tukey factorizes into polynomial rings with binomial moduli, Bruun factorizes into polynomial rings with trinomial moduli. If the coefficient ring is a finite field or finite ring, Bruun works in some cases where Cooley–Tukey doesn't since factorizing into binomials implies factorizing into trinomials but the converse doesn't always hold. The downside of Bruun is the increased number of arithmetic during the transformation.

**Rader vs others.** Rader converts size-$n$ cyclic transformation into a size-$(n-1)$ cyclic convolution with linear pre- and post-processing when $n$ is an odd prime. Other approaches rely on a factorization of $n$, implying that $n$ must be composite.

**Toom–Cook vs others.** Cooley–Tukey, Good–Thomas, Bruun, and Rader are isomorphisms where the dimensions are preserved during the transformation. On the hand, Toom–Cook is a monomorphism where the dimension becomes larger after the transformation. For the definability, Toom–Cook requires the existences of the inverses of some integers. This is generally more favorable than the FFTs in this section since one can always go for localization (cf. Section 4.1) for constructing the inverses of integers, which, in practice, amounts to replacing the coefficient ring with a slightly larger one. FFTs require the existence of a principal $n$-th root of unity and the inverse $n^{-1}$ where the former only exists in certain coefficient ring (cf. Section 3.2).

Table 8: Overview of the domains and images for Cooley–Tukey, Good–Thomas, Bruun, Rader, and Toom–Cook.

| Approach | Domain | Image |
|---|---|---|
| Cooley–Tukey | $\frac{R[x]}{\langle x^n - \zeta^n \rangle}$ | $\prod_i \frac{R[x]}{\langle x - \zeta\omega_n^i \rangle}$ |
| Good–Thomas | $\frac{R[x]}{\langle x^n - 1 \rangle}$ | $\prod_i \frac{R[x]}{\langle x - \omega_n^i \rangle}$ |
| Bruun | $\frac{R[x]}{\langle x^{2n} - (\xi^n + \xi^{-n})\zeta^n x^n + \zeta^{2n} \rangle}$ | $\prod_i \frac{R[x]}{\langle x^2 - \left(\xi\omega_n^i + (\xi\omega_n^i)^{-1}\right)\zeta x + \zeta^2 \rangle}$ |
| Rader | $\frac{R[x]}{\langle x^n - 1 \rangle}$ | $\prod_i \frac{R[x]}{\langle x - \omega_n^i \rangle}$ |
| Toom–Cook | $R[x]_{<n}$ | $\prod_{i=0,\dots,2n-2} \frac{R[x]}{\langle x - s_i \rangle}$ |

Table 9: Overview of the defining conditions of Cooley–Tukey, Good–Thomas, Bruun, Rader, and Toom–Cook.

| Approach | Requirements | Comment |
|---|---|---|
| Cooley–Tukey | $\exists \omega_n, \zeta, \zeta^{-1}, n^{-1} \in R$ | Fairly flexible. |
| Good–Thomas | $\exists \omega_n, n^{-1} \in R$ | $\exists$ coprime factorization of $n$. |
| Bruun | $\exists \xi\omega_n^i + \left(\xi\omega_n^i\right)^{-1}, \zeta, \zeta^{-1}, n^{-1} \in R$ | More flexible than Cooley–Tukey. |
| Rader | $\exists \omega_n, n^{-1} \in R$ | Odd prime $n$. |
| Toom–Cook | Inverses of integers. | Fairly flexible. |

# 4 Coefficient Ring Injections

This section reviews existing techniques and benefits of a coefficient ring injection:

$$R \hookrightarrow R'.$$

We focus on the structural implications. There are two questions we wish to answer in this section.

- What if there are no inverses of some integers defining a correct transformation?
- What if there are no principal roots of unity defining a correct transformation?

Section 4.1 reviews localization at non-zero integers for adjoining inverses, Section 4.2 reviews Schönhage's and Nussbaumer's FFTs adjoining symbolic principal roots of unity, Section 4.3 reviews an alternative approach switching to $R'$ with suitable inverses and principal roots of unity, and Section 4.4 compares the cost of coefficient ring injections. See Table 10 for an overview.

Table 10: Overview of coefficient ring injection techniques.

| Technique | Inverse | Principal root of unity |
|---|---|---|
| Localization | ✓ | - |
| Schönhage/Nussbaumer | - | ✓ |
| Coeff. ring switching | ✓ | ✓ |

## 4.1 Localization

Let $n \in \mathbb{Z}$ be non-invertible in $R$. Localization formulates "division by an integer $n^k$ in $R$." We quote the following from [Jac12b, Section 7.2] for the propose of localization:

> Given a (commutative) ring $R$ and a subset $S$ of $R$, to construct a ring $R_S$ and a homomorphism $\lambda_S$ of $R$ into $R_S$ such that every $\lambda_S(s), s \in S$, is invertible in $R_S$, and the pair $(R_S, \lambda_S)$ is universal for such pairs in the sense that if $\eta$ is any homomorphism of $R$ into a ring $R'$ such that every $\eta(s)$ is invertible, then there exists a unique homomorphism $\tilde{\eta} : R_S \to R'$ such that $\eta = \tilde{\eta} \circ \lambda_S$[6].

The ring $R_S$ is also commonly denoted as $S^{-1}R$. We leave the formal treatment to Appendix F and explain with a small example.

Suppose we want to compute $c_0 + c_1 x = (a_0 + a_1 x)(b_0 + b_1 x)$ in $\mathbb{Z}_{2^{15}}[x]/\langle x^2 - 1\rangle$ with "Cooley–Tukey FFT". We compute $a_0 + a_1 x \mapsto (a_0 + a_1, a_0 - a_1)$ and $b_0 + b_1 x \mapsto (b_0 + b_1, b_0 - b_1)$, point-multiply them, and perform an add-sub pair. The result is $((a_0 + a_1)(b_0 + b_1) \pm (a_0 - a_1)(b_0 - b_1)) = 2(a_0 b_0 + a_1 b_1, a_0 b_1 + a_1 b_0)$. It remains to "divide by two". Localization means the following monomorphisms:

$$\frac{\mathbb{Z}_{2^{15}}[x]}{\langle x^2 - 1\rangle} \hookrightarrow \prod \frac{\mathbb{Z}_{2^{16}}[x]}{\langle x \pm 1\rangle} \hookrightarrow \frac{\mathbb{Z}_{2^{16}}[x]}{\langle x^2 - 1\rangle}.$$

Since we know that the result is a 2-multiple of the desired one, we can extract the result by maintaining the set of 2-multiples as in $\mathbb{Z}_{2^{16}}$.

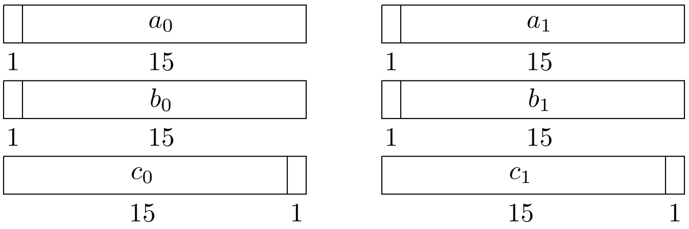


Figure 1: Localization for $\mathbb{Z}_{2^{15}}$ in $\mathbb{Z}_{2^{16}}$. We store the 15-bit values $a_0, a_1, b_0, b_1$ as halfwords (little endian in the Figure). For the 15-bit values $c_0, c_1$, we compute the 16-bit values $2c_0$ and $2c_1$ and extract the $c_0$ and $c_1$ by shifting.

[6] The last sentence actually ends with "such that the diagram [Figure] is commutative". We replace the description with the desired composition.

**Real-world example(s).** Recall that for Toom-3 with the point set $\{0, \pm 1, 2, \infty\}$, we have to apply

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 2 & 4 & 8 & 16 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & -\frac{1}{3} & -\frac{1}{6} & 2 \\ -1 & \frac{1}{2} & \frac{1}{2} & 0 & -1 \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{6} & \frac{1}{6} & -2 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

If we work over the ring $\mathbb{Z}_{2^{11}}[x]$ used in `ntruhps2048677`, then we have to maintain the values in $\mathbb{Z}_{2^{12}}$ for adjoining $2^{-1}$. Another example is Toom-5. If we choose $\{0, \pm 1, \pm 2, \pm 3, 4, \infty\}$ as the point set for evaluation, we must adjoin $16^{-1}$. [CCHY24] showed that one can instead switch to $\left\{0, \pm 1, \pm 2, \pm \frac{1}{2}, 3, \infty\right\}$ requiring only $8^{-1}$.

It should be noted that localization need not to adjoin the inverses uniformly in practice. For example, if we apply Toom-4 with the point set $\{s_i\} = \{0, \pm 1, \pm 2, 3, \infty\}$, then we only need to implement the following monomorphism:

$$\frac{\mathbb{Z}_{2^k}[x]}{\langle \prod_i s_i \rangle} \hookrightarrow \frac{\mathbb{Z}_{2^{k+2}}[x]}{\langle x \rangle} \times \frac{\mathbb{Z}_{2^{k+2}}[x]}{\langle x-1 \rangle} \times \frac{\mathbb{Z}_{2^{k+3}}[x]}{\langle x+1 \rangle} \times \frac{\mathbb{Z}_{2^{k+3}}[x]}{\langle x-2 \rangle} \times \frac{\mathbb{Z}_{2^{k+3}}[x]}{\langle x+2 \rangle} \times \frac{\mathbb{Z}_{2^{k+3}}[x]}{\langle x-3 \rangle} \times \frac{\mathbb{Z}_{2^k}[x]}{\langle x-\infty \rangle}.$$

This implies one can apply more aggressive transformations to some subproblems by working over $\mathbb{Z}_{2^{k+2}}$ and $\mathbb{Z}_{2^k}$ instead of $\mathbb{Z}_{2^{k+3}}$. The non-uniform property of localization with Toom–Cook does not seem to appear in the literature, but we believe there are practical benefits for implementations.

## 4.2 Schönhage's and Nussbaumer's Fast Fourier Transforms

Schönhage's [Sch77] and Nussbaumer's [Nus80] FFTs craft principal roots of unity defining FFTs. For simplicity, we explain the ideas for the cases $R[x] \Big/ \left\langle x^{2^k} \pm 1 \right\rangle$.

**Cyclic Schönhage [Ber01, Section 9].** For the cyclic Schöhage for $R[x] \Big/ \left\langle x^{2^k} - 1 \right\rangle$, we choose an $l \geq \frac{k}{2} - 1$, introduce the relation $x^{2^l} \sim y$, and replace the relation with $x^{2^{l+1}} \sim -1$. Define $\mathcal{R}' \coloneqq R[x] \Big/ \left\langle x^{2^{l+1}} + 1 \right\rangle$, and rewrite the polynomial ring as a polynomial ring with indeterminate $y$ and coefficient ring $\mathcal{R}'$. Since $x^{2^{l+1}} = -1 \in \mathcal{R}'$ and $l + 2 \geq k - l$, $x^{2^{2l+2-k}}$ is a principal $2^{k-l}$-th root of unity defining a size-$(k-l)$ cyclic FFT. In summary, we have

$$\frac{R[x]}{\left\langle x^{2^k} - 1 \right\rangle} \cong \frac{R[x, y]}{\left\langle x^{2^l} - y, y^{2^{k-l}} - 1 \right\rangle} \hookrightarrow \frac{\mathcal{R}'[y]}{\left\langle y^{2^{k-l}} - 1 \right\rangle} \cong \prod_i \frac{\mathcal{R}'[y]}{\left\langle y - \boldsymbol{\omega}_{2^{k-l}}^i \right\rangle}$$

where $\boldsymbol{\omega}_{2^{k-l}} \coloneqq x^{2^{2l+2-k}}$. The optimal choice is $l = \left\lceil \frac{k}{2} \right\rceil - 1$ leading to

$$\frac{R[x]}{\left\langle x^{2^k} - 1 \right\rangle} \hookrightarrow \frac{\mathcal{R}'[y]}{\left\langle y^{2^{\left\lfloor \frac{k}{2} \right\rfloor + 1}} - 1 \right\rangle} \cong \prod_i \frac{\mathcal{R}'[y]}{\left\langle y - x^{2^{\left\lceil \frac{k}{2} \right\rceil - \left\lfloor \frac{k}{2} \right\rfloor} \cdot l} \right\rangle}$$

with $\mathcal{R}' = R[x] \Big/ \left\langle x^{2^{\left\lceil \frac{k}{2} \right\rceil}} + 1 \right\rangle$. Since multiplications by powers of $x$ in $\mathcal{R}'$ amounts to negacyclic shifts, we only need additions and subtractions for converting a polynomial multiplication in $R[x] \Big/ \left\langle x^{2^k} - 1 \right\rangle$ into $2^{k-l}$ many polynomial multiplications in $R[x] \Big/ \left\langle x^{2^{l+1}} + 1 \right\rangle$.

**Negacyclic Schönhage [Sch77].** We can also apply Schönhage to $R[x]\Big/\Big\langle x^{2^k}+1\Big\rangle$: we choose $l \geq \frac{k-1}{2}$ and proceed similarly as in the cyclic case. This leads to

$$\frac{R[x]}{\left\langle x^{2^k}+1\right\rangle} \cong \frac{R[x,y]}{\left\langle x^{2^l}-y, y^{2^{k-l}}+1\right\rangle} \hookrightarrow \frac{\mathcal{R}'[y]}{\left\langle y^{2^{k-l}}+1\right\rangle} \cong \prod_i \frac{\mathcal{R}'[y]}{\left\langle y-\boldsymbol{\omega}_{2^{k-l+1}}^{1+2i}\right\rangle}$$

where $\mathcal{R}' \coloneqq R[x]\Big/\Big\langle x^{2^{l+1}}+1\Big\rangle$ and $\boldsymbol{\omega}_{2^{k-l+1}} \coloneqq x^{2^{2l+1-k}}$. For the optimal choice $l = \left\lceil\frac{k-1}{2}\right\rceil$, we have

$$\frac{R[x]}{\left\langle x^{2^k}+1\right\rangle} \hookrightarrow \frac{\mathcal{R}'[y]}{\left\langle y^{2^{\left\lfloor\frac{k+1}{2}\right\rfloor}}+1\right\rangle} \cong \prod_i \frac{\mathcal{R}'[y]}{\left\langle y-x^{2^{\left\lceil\frac{k-1}{2}\right\rceil-\left\lfloor\frac{k-1}{2}\right\rfloor}\cdot(1+2i)}\right\rangle}.$$

**Nussbaumer [Nus80].** Nussbaumer is only applicable to the negacyclic case, but it sometimes results in smaller subproblems. Conceptually, we swap the roles of $x$ and $y$ while applying the FFT. We choose an $l \leq \frac{k}{2}$, introduce the relation $x^{2^l} \sim y$, and replace it with $x^{2^{l+1}} \sim 1$. Instead of regarding the polynomial ring as a polynomial ring with indeterminate $y$ in Schönhage, we regard it as a polynomial ring with indeterminate $x$. Define $\mathcal{R}' \coloneqq R[y]\Big/\Big\langle y^{2^{k-l}}+1\Big\rangle$. Since $y^{2^{k-l}} = -1 \in \mathcal{R}'$, $y^{2^{k-2l}}$ is a principal $2^{l+1}$-th root of unity defining a size-$2^{l+1}$ cyclic FFT. Overall, we have

$$\frac{R[x]}{\left\langle x^{2^k}+1\right\rangle} \cong \frac{R[x,y]}{\left\langle x^{2^l}-y, y^{2^{k-l}}+1\right\rangle} \hookrightarrow \frac{\mathcal{R}'[x]}{\left\langle x^{2^{l+1}}-1\right\rangle} \cong \prod_i \frac{\mathcal{R}'[x]}{\left\langle x-\boldsymbol{\omega}_{2^{l+1}}^{i}\right\rangle}$$

where $\boldsymbol{\omega}_{2^{l+1}} \coloneqq y^{2^{k-2l}}$. For the optimal choice $l = \left\lfloor\frac{k}{2}\right\rfloor$, we have

$$\frac{R[x]}{\left\langle x^{2^k}+1\right\rangle} \hookrightarrow \frac{\mathcal{R}'[x]}{\left\langle x^{2^{\left\lfloor\frac{k}{2}\right\rfloor+1}}-1\right\rangle} \cong \prod_i \frac{\mathcal{R}'[x]}{\left\langle x-y^{2^{\left\lceil\frac{k}{2}\right\rceil-\left\lfloor\frac{k}{2}\right\rfloor}\cdot i}\right\rangle}.$$

**Comparisons of Schönhage and Nussbaumer.** Table 11 summarizes the domains, images, and defining conditions of radix-2 Schönhage and Nussbaumer, and Table 12 summarizes the domains and images of radix-2 Schönhage and Nussbaumer with optimal parameters. As seen in Table 12, for the negacyclic case $R[x]\Big/\Big\langle x^{2^k}+1\Big\rangle$, Nussbaumer results in size-$\left\lceil\frac{k}{2}\right\rceil$ negacyclic convolutions and Schönhage results in size-$\left\lceil\frac{k+1}{2}\right\rceil$ negacyclic convolutions. This implies Nussbaumer is more preferable in the negacyclic case if the number of operations in $R$ is the sole optimizing target [Ber01, Section 9].

Table 11: Overview of radix-2 Schönhage and Nussbaumer.

| | Domain | Image | Condition |
|---|---|---|---|
| Cyclic Schönhage | $\frac{R[x]}{\left\langle x^{2^k}-1\right\rangle}$ | $\left(\frac{R[x]}{\left\langle x^{2^{l+1}}+1\right\rangle}\right)^{2^{k-l}}$ | $l \geq \frac{k}{2}-1$ |
| Negacyclic Schönhage | $\frac{R[x]}{\left\langle x^{2^k}+1\right\rangle}$ | $\left(\frac{R[x]}{\left\langle x^{2^{l+1}}+1\right\rangle}\right)^{2^{k-l}}$ | $l \geq \frac{k-1}{2}$ |
| Nussbaumer | $\frac{R[x]}{\left\langle x^{2^k}+1\right\rangle}$ | $\left(\frac{R[y]}{\left\langle y^{2^{k-l}}+1\right\rangle}\right)^{2^{l+1}}$ | $l \leq \frac{k}{2}$ |

Table 12: Overview of optimal radix-2 Schönhage and Nussbaumer.

| | Domain | Image |
|---|---|---|
| Cyclic Schönhage | $\frac{R[x]}{\left\langle x^{2^k}-1 \right\rangle}$ | $\left( \frac{R[x]}{\left\langle x^{2^{\left\lceil \frac{k}{2} \right\rceil + 1}} + 1 \right\rangle} \right)^{2^{\left\lfloor \frac{k}{2} \right\rfloor + 1}}$ |
| Negacyclic Schönhage | $\frac{R[x]}{\left\langle x^{2^k}+1 \right\rangle}$ | $\left( \frac{R[x]}{\left\langle x^{2^{\left\lceil \frac{k+1}{2} \right\rceil}} + 1 \right\rangle} \right)^{2^{\left\lfloor \frac{k-1}{2} \right\rfloor}}$ |
| Nussbaumer | $\frac{R[x]}{\left\langle x^{2^k}+1 \right\rangle}$ | $\left( \frac{R[y]}{\left\langle y^{2^{\left\lceil \frac{k}{2} \right\rceil}} + 1 \right\rangle} \right)^{2^{\left\lfloor \frac{k}{2} \right\rfloor + 1}}$ |

**Generalizations.** There are several directions generalizing Schönhage and Nussbaumer. For the polynomial modulus $x^{2^k} \pm 1$ in Schönhage, the idea applies to any factors of $x^{2^k} \pm 1$. In fact, the case $x^{2^k} + 1$ directly follows from $x^{2^{k+1}} - 1$. As for the polynomial modulus in Nussbaumer, we demonstrate the roles of the polynomial factors in Appendix H.2. Another direction is to replace $x$ by an odd power of $x$. Finally, for the general $n$, we refer to Appendix G.

**Real-world example(s).** [BBCT22] transformed $\mathbb{Z}_{4591}[x] / \left\langle (x^{1024}+1)(x^{512}-1) \right\rangle$ as follows. They started with Schönhage for

$$\frac{\mathbb{Z}_{4591}[x]}{\langle (x^{1024}+1)(x^{512}-1) \rangle} \cong \frac{\mathbb{Z}_{4591}[x,y]}{\langle x^{32}-y, (y^{32}+1)(y^{16}-1) \rangle} \hookrightarrow \frac{\left( \frac{\mathbb{Z}_{4591}[x]}{\langle x^{64}+1 \rangle} \right)[y]}{\langle (y^{32}+1)(y^{16}-1) \rangle}$$

and applied Nussbaumer to $\mathbb{Z}_{4591}[x] / \left\langle x^{64}+1 \right\rangle$.

## 4.3 Coefficient Ring Switching

For multiplying polynomials in $\mathbb{Z}_q[x]/\langle \boldsymbol{g} \rangle$ for $\boldsymbol{g} = x^n \pm 1$, we can always multiply in $\mathbb{Z}[x]/\langle \boldsymbol{g} \rangle$ and reduce to $\mathbb{Z}_q$ at the end. There are many ways to compute the result in $\mathbb{Z}$. For simplicity, let's assume we want to multiply two polynomials. Since the result over $\mathbb{Z}$ has coefficients with absolute values bounded by $\frac{nq^2}{4}$ for signed arithmetic, we choose a $q'$ admitting a suitable FFT over $\boldsymbol{g}$ with $\frac{q'}{2} > \frac{nq^2}{4}$ and compute in $\mathbb{Z}_{q'}[x]/\langle \boldsymbol{g} \rangle$ with signed arithmetic. For unsigned arithmetic, the condition is replaced by $q' > nq^2$. In many lattice-based cryptosystems, one of the operands has coefficients with absolute values bounded by a small constant, and $q'$ only needs to be larger than a small-multiple of $nq$. For example, one of the operands in NTRU [CDH+20] has coefficients drawn from $\{0, \pm 1\}$ and the small secret vector of polynomials in Saber [DKRV20] has coefficients drawn from $\{-3, \dots, 0, \dots, 3\}, \{-4, \dots, 0, \dots, 4\}$, and $\{-5, \dots, 0, \dots, 5\}$. Obviously, $\mathbb{Z}_q \hookrightarrow \mathbb{Z}_{q'}$ is an injection. If arithmetic defined over $q'$ is too large for efficient implementations, one can also choose coprime integers $q_i$'s as long as their product $q' \coloneqq \prod_i q_i$ fulfills the same conditions. The tuple of coprime integers is called a residue number system (RNS). Multiplying over $\mathbb{Z}_{q'}$ and $\prod_i \mathbb{Z}_{q_i}$ is used in many contexts, including lattice-based cryptography [FSS20, BBC$^+$20, ACC$^+$21, CHK$^+$21, ACC$^+$22], and also before public key cryptography [Nic71, Pol71].

## 4.4 Comparisons

We briefly compare the cost of coefficient ring injections. See Table 13 for an overview.

**Coefficient ring switching vs localization.** Localization introduces inverses of integers, commonly $2^{-k}$. In this case, we replace the coefficient ring with the $k$-bit larger one. Very often, we choose a $k$ such that the new coefficient ring still amount to the same arithmetic precision, so there is usually no additional cost in practice. As for coefficient ring switching, since the bit-size is at least 2× larger, cares must be taken while choosing the new coefficient ring.

**Coefficient ring switching vs Schönhage/Nussbaumer.** Schönhage and Nussbaumer adjoin the principal roots of unity by extending the polynomial moduli, and result in 2× number of coefficients. Coefficient ring switching introduces the principal roots by replacing the coefficient rings with much larger ones, and the polynomial moduli remain the same. To figure out which technique is more beneficial, programmers have to first figure out the efficiency of the multiplication in the coefficient rings. In Schönhage and Nussbaumer, the coefficient ring remains the same but we have doubly many elements. On the other hand, if we switch to a new coefficient ring, the bit-size of the new coefficient ring is at least 2× larger than the original one. If the cost of multiplication in the original coefficient ring is very fast compared to the new large coefficient ring, then Schönhage and Nussbaumer might be more preferable.

Table 13: Overview of the cost of coefficient ring injections.

| Techinque | Adjoined structure | Coeff. ring (bit-size) | Poly. modulus |
|---|---|---|---|
| Localization | $2^{-k}$ | $k$-bit larger | - |
| Schönhage/ Nussbaumer | $\omega_{2^k}$ | - | 2× #coeff. |
| Coeff. ring switching | $2^{-k}, \omega_{2^k}$ | $2^+\times$ larger | - |

# 5 Polynomial Moduli

This section reviews several techniques related to the polynomial modulus $\boldsymbol{g}$ of $R[x]/\langle \boldsymbol{g}(x) \rangle$. Section 5.2 reviews twisting and composed multiplication converting $R[x]/\langle \boldsymbol{g}(x) \rangle$ into a polynomial ring of the form $R[y]/\langle \boldsymbol{g}(\zeta y) \rangle$, Section 5.1 reviews embedding and evaluation at $\infty$ for choosing a polynomial $\boldsymbol{h}$ admitting the monomorphism $R[x]/\langle \boldsymbol{g}(x) \rangle \hookrightarrow R[x]/\langle \boldsymbol{h}(x) \rangle$. Section 5.3 reviews truncation computing products in $R[x]/\left\langle \prod_{i \in \mathcal{I}'} \boldsymbol{g}_i \right\rangle$ with an isomorphism derived from an isomorphism for $R[x]/\left\langle \prod_{i \in \mathcal{I}} \boldsymbol{g}_i \right\rangle$ with $\mathcal{I}' \subset \mathcal{I}$. Section 5.4 reviews incomplete transformations and striding, and Section 5.5 reviews the Toeplitz matrix-vector product for $R[x]/\langle x^n - \alpha x - \beta \rangle$ from the dual module view of algebra homomorphisms multiplying two size-$n$ polynomials in $R[x]$.

## 5.1 Embedding (Polynomial Modulus) and Evaluation at $\infty$

Let $\boldsymbol{g} \in R[x]$ be a polynomial with $\deg(\boldsymbol{g}) \leq n$. An obvious approach for multiplying polynomials in $R[x]/\langle \boldsymbol{g} \rangle$ is multiplying in $R[x]$ followed by reducing modulo $\boldsymbol{g}$. This is the embedding technique for ignoring the structure of $\boldsymbol{g}$. For $R[x]$, one further applies an identity map from $R[x]$ to $R[x]/\langle \boldsymbol{h} \rangle$ where $\boldsymbol{h}$ is a polynomial with degree larger than the product in $R[x]$. $\boldsymbol{h}$ is usually a polynomial with a very nice structure for fast transformations.

Evaluation at $\infty$ is an optimization for choosing $\boldsymbol{h}$ [Win80]. Suppose $\boldsymbol{r}$ is the product in $R[x]$, $d$ the degree, and $r_d$ the leading term of $\boldsymbol{r}$. Instead of computing $\boldsymbol{r}$, we compute $\boldsymbol{r} - r_d\boldsymbol{h}$ by embedding into $R[x]/\langle \boldsymbol{h} \rangle$ with $\deg(\boldsymbol{h}) = d$. The term $r_d\boldsymbol{h}$ is computed individually and added back. In the literature, the idea is commonly presented as allowing $\boldsymbol{h}$ to contain the polynomial $x - \infty$. Historically, evaluation at $\infty$ was first used by [KO62]. [Too63] chose small integers for evaluation, and [Win80, Page 31] replaced a point with $\infty$ for unifying Karatsuba and Toom–Cook. [Win80]'s idea was already as general as this section and applied to other choices of $\boldsymbol{h}$.

In [KO62], they computed $(a_0 + a_1x)(b_0 + b_1x)$ with $(a_0 + a_1x)(b_0 + b_1x) = a_0b_0 + ((a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1)x + a_1b_1x^2$. If we choose $\boldsymbol{h} = x^2 + x$, the polynomial $(a_0 + a_1x)(b_0 + b_1x) - a_1b_1(x^2 + x) = a_0b_0 + (a_0b_1 + a_1b_0 - a_1b_1)x$ can be computed in $R[x]/\left\langle x^2 + x \right\rangle$. Applying $R[x]/\left\langle x^2 + x \right\rangle \cong R[x]/\langle x \rangle \times R[x]/\langle x - 1 \rangle$ gives us $(a_0, a_0 + a_1)$ and $(b_0, b_0 + b_1)$. After point-multiplying and inverting, we have $a_0b_0 + ((a_0 + a_1)(b_0 + b_1) - a_0a_1)x$. Adding $a_1b_1(x^2 + x)$ derives the desired result.

It doesn't seem that people have ever chosen $\boldsymbol{h}$ with $x - \infty$ for FFT in the literature. We believe the reason is that one usually splits $\boldsymbol{h}$ into a large number of small factors for FFT, and the benefit of replacing one of them with $x - \infty$ is marginal. Nevertheless, we give the following example of multiplying $(a_0 + a_1x)(b_0 + b_1x)$ for referential purposes. We rewrite $(a_0 + a_1x)(b_0 + b_1x)$ as $(a_0b_0 + a_1b_1) + (a_0b_1 + a_1b_0)x + a_1b_1(x^2 - 1)$, compute $(a_0b_0 + a_1b_1) + (a_0b_1 + a_1b_0)x$ with the isomorphism $R[x]/\left\langle x^2 - 1 \right\rangle \cong \prod R[x]/\langle x \pm 1 \rangle$, and finally add $a_1b_1(x^2 - 1)$ to the result.

## 5.2 Twisting and Composed Multiplication

### 5.2.1 Twisting

Let $\zeta \in R$ be an invertible element. Twisting is an isomorphism from $R[x]/\langle \boldsymbol{g}(x) \rangle$ to $R[y]/\langle \boldsymbol{g}(\zeta y) \rangle$ by introducing $x \sim \zeta y$. We have the isomorphism $R[x]/\langle \boldsymbol{g}(x) \rangle \cong R[x, y]/\langle x - \zeta y, \boldsymbol{g}(\zeta y) \rangle$ and treat $R[x]/\langle x - \zeta y \rangle$ as the coefficient ring. Let $n = \deg(\boldsymbol{g})$. In order to change the basis from $(1, x, \dots, x^{n-1})$ to $(1, y, \dots, y^{n-1}) = (1, \zeta x, \dots, \zeta^{n-1}x^{n-1})$, we have to multiply the coefficients with the powers $\zeta, \dots, \zeta^{n-1}$. This usually amounts to $n - 1$ multiplications in $R$. However, if $n$ is odd and $\zeta = -1$, we do not need any multiplication for the isomorphism $R[x]/\langle x^n + 1 \rangle \cong R[x, y]/\langle x + y, y^n - 1 \rangle$. We will shortly see how this insight can be systematized in Section 5.3.

Twisting was introduced in [GS66] for computing FFTs with $R[x]/\langle x^{n_0n_1} - 1 \rangle \cong \prod_i R[x]/\left\langle x^{n_1} - \omega_{n_0}^i \right\rangle \cong \prod_i R[x]/\langle x^{n_1} - 1 \rangle$. See [DH84, Für09] for more insights on the choices of $n_0$ and $n_1$.

### 5.2.2 Composed Multiplication

We go through a specialized approach when $R = \mathbb{F}_q$. Given $\boldsymbol{f}_0, \boldsymbol{f}_1 \in \mathbb{F}_q[x]$, we defined their composed multiplication [BC87] as

$$\boldsymbol{f}_0 \odot \boldsymbol{f}_1 \coloneqq \prod_{\boldsymbol{f}_0(\alpha)=0} \prod_{\boldsymbol{f}_1(\beta)=0} (x - \alpha\beta)$$

where $\alpha, \beta$ are elements from an extension field of $\mathbb{F}_q$. Composed multiplication generalizes twisting to the polynomial modulus of the form $(x - \zeta) \odot \boldsymbol{f}(x)$. In particular, we have $\mathbb{F}_q[x]/\langle (x - \zeta) \odot \boldsymbol{f}(x) \rangle \cong \mathbb{F}_q[y]/\langle x - \zeta y, \boldsymbol{f}(y) \rangle$.

Another benefit of composed multiplication is systematically deriving transformations based on (presumably much simpler) coprime factorizations. Let $\boldsymbol{f}_0 = \prod_{i_0} \boldsymbol{f}_{0,i_0}$ and $\boldsymbol{f}_1 = \prod_{i_1} \boldsymbol{f}_{1,i_1}$ be coprime factorizations in $\mathbb{F}_q[x]$. We have $\boldsymbol{f}_0 \odot \boldsymbol{f}_1 = \prod_{i_0} \left(\boldsymbol{f}_{0,i_0} \odot \boldsymbol{f}_1\right) = \prod_{i_0,i_1} \left(\boldsymbol{f}_{0,i_0} \odot \boldsymbol{f}_{1,i_1}\right)$. A practically important example is $\boldsymbol{f}_0 = x^r - 1 = \prod_{i_0} \left(x - \omega_r^{i_0}\right) \in \mathbb{F}_q[x]$ and $\boldsymbol{f}_1 = x^{2^k} - 1$. Given a factorization $x^{2^k} - 1 = \prod_{i_1} \boldsymbol{f}_{1,i_1}$ in $\mathbb{F}_q[x]$, we have

$$x^{2^k r} - 1 = \prod_{i_0} \left(x^{2^k} - \omega_r^{2^k i_0}\right) = \prod_{i_0,i_1} \omega_r^{\deg(\boldsymbol{f}_{1,i_1})} \boldsymbol{f}_{1,i_1}(\omega_r^{-i_0} x).$$

**Real-world example(s).** For an odd number $r$ with $r|(4591-1)$, we have the size-$r$ transformation $R[x]/\langle x^r - 1\rangle \cong \prod_i R[x]/\langle x - \omega_r^i\rangle$. We extend it to a size-$2^k r$ transformation for the ease of vectorization. [HLY24] implemented the isomorphism $\mathbb{Z}_{4591}[x]\big/\big\langle x^{1632} - 1\big\rangle \cong \prod_i \mathbb{Z}_{4591}[x]\big/\big\langle x^{16} \pm \omega_{51}^i\big\rangle$, and factor $\mathbb{Z}_{4591}[x]\big/\big\langle x^{16} \pm \omega_{51}^i\big\rangle$ into polynomial rings modulo the composed multiplications of $x - \omega_{51}^i$ and factors of $x^{16} \pm 1$.

## 5.3 Truncation

Truncation is a simple and powerful idea. Let $\mathcal{I}' \subset \mathcal{I}$ be index sets and $\{\boldsymbol{g}_i\}_{i\in\mathcal{I}}$ be coprime polynomials in $R[x]$. Suppose we are given the following isomorphism

$$\eta : \begin{cases} R[x]\Big/\Big\langle \prod_{i\in\mathcal{I}} \boldsymbol{g}_i \Big\rangle & \to \quad \prod_{i\in\mathcal{I}} R[x]/\langle \boldsymbol{g}_i\rangle\ , \\ \boldsymbol{a} & \mapsto \quad (\boldsymbol{a} \bmod \boldsymbol{g}_i)_{i\in\mathcal{I}}\ . \end{cases}$$

We can naturally define an isomorphism $\eta_{\mathcal{I}'}$ as

$$\eta_{\mathcal{I}'} : \begin{cases} R[x]\Big/\Big\langle \prod_{i\in\mathcal{I}'} \boldsymbol{g}_i \Big\rangle & \to \quad \prod_{i\in\mathcal{I}'} R[x]/\langle \boldsymbol{g}_i\rangle\ , \\ \boldsymbol{a} & \mapsto \quad (\boldsymbol{a} \bmod \boldsymbol{g}_i)_{i\in\mathcal{I}'}\ . \end{cases}$$

$\eta_{\mathcal{I}'}$ is called the truncation of $\eta$ at $R[x]/\big\langle \prod_{i\in\mathcal{I}'} \boldsymbol{g}_i\big\rangle$. Truncation was introduced by [CF94, Section 7]. [Ber08] (according to [vdH04], the work [Ber08] was already online prior to [vdH04]) described the benefit in terms of complexity, and [vdH04] named the technique "truncated Fourier transform" for the FFT case. We call it truncation since it is not restricted to FFTs. We demonstrate some of its applications in this section. See Appendix H for more applications to fast transformations.

### 5.3.1 Application I: $R[x]\Big/\Big\langle x^{2^{k-1}} + 1\Big\rangle$ from $R[x]\Big/\Big\langle x^{2^k} - 1\Big\rangle$

We derive FFT for $R[x]\Big/\Big\langle x^{2^{k-1}} + 1\Big\rangle$ from the one for $R[x]\Big/\Big\langle x^{2^k} - 1\Big\rangle$. For a principal $2^k$-th root of unity $\omega_{2^k}$ realizing $R[x]\Big/\Big\langle x^{2^k} - 1\Big\rangle \cong \prod_{i=0}^{2^k-1} R[x]/\langle x - \omega_{2^k}^i\rangle$, we have $R[x]\Big/\Big\langle x^{2^{k-1}} + 1\Big\rangle \cong \prod_{i=0}^{2^{k-1}-1} R[x]/\langle x - \omega_{2^k}^{1+2i}\rangle$. We can generalize the idea to arbitrary transformation sizes. Below is a straightforward generalization of [CF94, Section 7] outlined in [Hwa22, Section 10]. Let $b = n$ and $\tilde{b} = \sum_j \tilde{b}_j 2^j$ be the 2's complement representation of $-n$ as a $\lceil \log_2 n\rceil$-bit integer. We have $b + \tilde{b} = 2^{\lceil \log_2 n\rceil}$ by definition and define a transformation for

$$R[x]\Bigg/\Bigg\langle \frac{x^{2^{\lceil \log_2 n\rceil}} - 1}{\prod_j \left(x^{2^j} + 1\right)^{\tilde{b}_j}} \Bigg\rangle .$$

This boils down to transformations for rings of the form $R[x] \Big/ \left\langle x^{2^k} \pm 1 \right\rangle$. An example is the Schönhage for $R[x] / \big\langle (x^{1024}+1)(x^{512}-1) \big\rangle$ derived from $R[x] / \big\langle x^{2048} - 1 \big\rangle$.

### 5.3.2 Application II: Rader's FFT

Let $p$ be an odd prime, $\mathcal{I} = \{0, \dots, p-1\}$, $\mathcal{I}^* = \{z \in \mathcal{I} | z \perp p\}$, and $g$ be a generator of $\mathcal{I}^*$. For a principal $p$-th root of unity, we show how Rader's FFT converts the computing task of size-$p$ cyclic FFT into a size-$\lambda(p)$ cyclic convolution in Section 3.6. In this section, we show that the isomorphism $R[x] / \big\langle \prod_{i \in \mathcal{I}^*} \left(x - \omega_p^i\right) \big\rangle \cong \prod_{i \in \mathcal{I}^*} R[x] / \big\langle x - \omega_p^i \big\rangle$ and its inverse can also be converted into size-$\lambda(p)$ cyclic convolutions. For generalization truncating a size-$n$ cyclic DFT to the roots with exponents coprime to $n$, see [Ber23, Sections 4.12.3 and 4.12.4].

**Forward transformation.** Given a polynomial $\sum_{j \in \mathbb{Z}_{\lambda(p)}} a_j x^j \in R[x] / \big\langle \prod_{i \in \mathcal{I}^*} \left(x - \omega_p^i\right) \big\rangle$ and its image $\left(\hat{a}_{i-1}\right)_{i \in \mathcal{I}^*} = \sum_{j \in \mathbb{Z}_{\lambda(p)}} a_j x^j \bmod \left(x - \omega_p^i\right)$, we have:

$$\begin{aligned} \hat{a}_{g^{\log_g i}-1} = \hat{a}_{i-1} = \sum_{j \in \mathbb{Z}_{\lambda(p)}} a_j \omega_p^{ij} = \omega_p^{-i} \sum_{j \in \mathbb{Z}_{\lambda(p)}} a_j \omega_p^{i(j+1)} = \omega_p^{-i} \sum_{j \in \mathcal{I}^*} a_{j-1} \omega_p^{ij} \\ = \quad \omega_p^{-g^{\log_g i}} \sum_{-\log_g j \in \mathbb{Z}_{\lambda(p)}} a_{g^{\log_g j}-1} \omega_p^{g^{\log_g i + \log_g j}} . \end{aligned}$$

If we multiply both sides by $\omega_p^{g^{\log_g i}}$, then we find that $\left(\omega_p^{g^k} \hat{a}_{g^k-1}\right)_{k \in \mathbb{Z}_{\lambda(p)}}$ is a size-$\lambda(p)$ cyclic convolution of $\left(a_{g^{-k}-1}\right)_{k \in \mathbb{Z}_{\lambda(p)}}$ and $\left(\omega_n^{g^k}\right)_{k \in \mathbb{Z}_{\lambda(p)}}$.

**Inverse transformation.** [Ber23, Section 4.8.2] showed that convolution by $\left(\omega_p^{g^k}\right)_{k \in \mathbb{Z}_{\lambda(p)}}$ can be inverted by convolution. By definition, convolution in the polynomial ring $R[x] / \big\langle x^{\lambda(p)} - 1 \big\rangle$ is the ring multiplication in the group algebra $R[\mathbb{Z}_{\lambda(p)}]$. Therefore, the inversion amounts to multiplying the multiplicative inverse of $\left(\omega_p^{g^k}\right)_{k \in \mathbb{Z}_{\lambda(p)}}$ in the group algebra $R[\mathbb{Z}_{\lambda(p)}]$. The inverse of $\left(\omega_p^{g^k}\right)_{k \in \mathbb{Z}_{\lambda(p)}}$ is $\frac{1}{p}\left(\omega_n^{-g^{-k}} - 1\right)_{k \in \mathbb{Z}_{\lambda(p)}}$. [Ber23] proved this by showing that the convolution of $\left(\omega_p^{g^k}\right)_{k \in \mathbb{Z}_{\lambda(p)}}$ and $\left(\omega_p^{-g^{-k}} - 1\right)_{k \in \mathbb{Z}_{\lambda(p)}}$ is $\left(\delta_{0,k} p\right)_{k \in \mathbb{Z}_{\lambda(p)}}$: For all $k \in \mathbb{Z}_{\lambda(p)}$, we find

$$\sum_{i+j=k} \omega_n^{g^i} \left(\omega_n^{-g^{-j}} - 1\right) = \sum_{i+j=k} \omega_n^{g^i\left(1-g^{-(i+j)}\right)} - \sum_{i+j=k} \omega_n^{g^i} = \delta_{0,k} p$$

as desired.

## 5.4 Incomplete Transformation and Striding

### 5.4.1 Incomplete Transformation

For a monic polynomial $\boldsymbol{g}(x^v) \in R[x]$, we call a homomorphism $f : R[x]/\langle \boldsymbol{g}(x^v) \rangle \to \mathcal{A}$ "incomplete" if $f$ starts with introducing $x^v \sim y$ and proceed as a polynomial ring in $y$ with the coefficient ring $R[x]/\langle x^v - y \rangle$. There are several benefits for an incomplete transformation: (i) the definability of fast transformation, (ii) the vectorization-friendliness of $x^v \sim y$, and (iii) the code size for implementing $f$. We give an example for (i) in this section. For the benefit of vectorization, see Section 6.2. As for (iii), we refer to [AHY22, Sections 3.2 and 3.3] for more details.

**Real-world example(s).** Let's take the polynomial ring $\mathbb{Z}_{3329}[x]/\langle x^{256}+1\rangle$ used in Kyber as an example. Since 3329 is a prime, we can only define a size-$n$ cyclic FFT for $n|3328$. This doesn't permit splitting the polynomial ring into linear factors since $x^{256}+1=\Phi_{512}$ and $512 \not| \ 3328$. What we can do is introduce $x^2 \sim y$ and split $\left(\mathbb{Z}_{3329}[x]/\langle x^2-y\rangle\right)[y]/\langle y^{128}+1\rangle$ into linear factors in $y$.

### 5.4.2 Striding

A closely related idea is striding – we regard $R[y]/\langle \boldsymbol{g}(y)\rangle$ as the coefficient ring. This is Nussbaumer (cf. Section 4.2) if we replace $x^v-y$ with an $\boldsymbol{h}(x)$, and ask $\boldsymbol{g}(y)|\Phi_{n'}(y)$ and $\boldsymbol{h}(x)|(x^{n'}-1)$ with $n'\geq 2v-1$. We also have striding Toom–Cook [Ber01, BMK$^+$22] if $\boldsymbol{h}(x)=\prod_i(x-s_i)$ for $\{s_i\}\subset\mathbb{Q}\cup\{\infty\}$.

## 5.5 Toeplitz Matrix-Vector Product

This section goes through a generic technique converting a fast computation for $R[x]$ into a fast computation for $R[x]/\langle x^n-\alpha x-\beta\rangle$. We present the mathematical background in this section and will review the architectural insights in Section 6.4.2.

### 5.5.1 Bilinear System

We review a generic technique for bilinear systems adapted from [Win80, Theorem 6].

**Theorem 1** ([Win80, Theorem 6] for $R$ commutative)**.** Let $R$ be a ring, $\mathcal{I},\mathcal{J},\mathcal{K}$ be finite index sets, and $(a_i)_{i\in\mathcal{I}},(b_j)_{j\in\mathcal{J}},(c_k)_{k\in\mathcal{K}}$ be tuples drawn from $R$. For a bilinear system

$$S_0:\forall k\in\mathcal{K},\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}r_{(i,j,k)}a_ib_j$$

with $r_{(i,j,k)}\in R$, we construct the following bilinear systems:

$$S_1:\forall j\in\mathcal{J},\sum_{i\in\mathcal{I}}\sum_{k\in\mathcal{K}}r_{(i,j,k)}a_ic_k,$$

$$S_2:\forall i\in\mathcal{I},\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}}r_{(i,j,k)}c_kb_j.$$

Then any bilinear algorithm for one of $S_0,S_1$ or $S_2$ leads to algorithms for the other two.

One can prove Theorem 1 by defining a $|\mathcal{K}|\times|\mathcal{I}|$ matrix $B_{k,i}:=\left(\sum_{j\in\mathcal{J}}r_{(i,j,k)}b_j\right)$, and write $S_0$ as $B\boldsymbol{a}$ and $S_2$ as $B^T\boldsymbol{c}$ where $\boldsymbol{a}$ and $\boldsymbol{c}$ are the column representations of $(a_i)_{i\in\mathcal{I}}$ and $(c_k)_{k\in\mathcal{K}}$. See Appendix J for details. If we choose $r_{(i,j,k)}:=[\![i+j=k]\!]$ where $[\![]\!]$ is the Iverson bracket[7] and $|\mathcal{K}|=|\mathcal{I}|+|\mathcal{J}|-1$, $S_0$ represents the coefficients of $\left(\sum_{i\in\mathcal{I}}a_ix^i\right)\left(\sum_{j\in\mathcal{J}}b_jx^j\right)$ in an obvious way. Then, $S_2$ becomes

$$S_2:\forall i\in\mathcal{I},\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}}[\![k-j=i]\!]\,c_kb_j.$$

This is called a transposed multiplication [Sho99, Section 3] or a middle product [HQZ04]. [Fid73, Theorem 4 and 5] proved that transposing an algorithm results in same numbers of constant multiplications ($ra_i$ for a constant $r$ in $R$), non-constant multiplications ($a_ib_j$), and additions/subtractions with a linear difference. For the history of transposition principle, see [BCS13, Section 4].

[7] Iverson bracket is an indicator function for the truthfulness. The image of $[\![]\!]$ is $\{0,1\}$, which can be certainly embedded into a ring.

We illustrate with the case $|\mathcal{I}| = |\mathcal{J}| = n$. $S_0 : \forall k \in \mathcal{K}, \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \llbracket i + j = k \rrbracket \, a_i b_j = \sum_{i \in \mathcal{I}, i \leq k} a_i b_{k-i}$ can be written as:

$$\begin{pmatrix} a_0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots \\ a_{n-1} & \ddots & \ddots & \ddots \\ 0 & \ddots & \ddots & \ddots \\ \vdots & \ddots & \ddots & \ddots \\ 0 & \ddots & \ddots & \ddots \end{pmatrix} \begin{pmatrix} b_0 \\ \vdots \\ b_{n-1} \end{pmatrix}.$$

And $S_2 : \forall i \in \mathcal{I}, \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} \llbracket k - j = i \rrbracket \, c_k b_j = \sum_{j \in \mathcal{J}} c_{i+j} b_j$ can be written as:

$$\begin{pmatrix} c_0 & \iddots & \iddots \\ \vdots & \iddots & \iddots \\ c_{n-1} & \cdots & c_{2n-2} \end{pmatrix} \begin{pmatrix} b_0 \\ \vdots \\ b_{n-1} \end{pmatrix}.$$

$S_2$ relates $S_0$ to polynomial multiplication modulo a polynomial.

### 5.5.2 Toeplitz Transform for $R[x]/\langle x^n - \alpha x - \beta \rangle$

Let $M$ be an $n \times n$ matrix. We call $M$ a Hankel matrix if $M_{i,j} = M_{i+1,j-1}$ for all possible $i, j$, and a Toeplitz matrix if $M_{i,j} = M_{i+1,j+1}$ for all possible $i, j$. Notice that a Hankel matrix can be converted into a Toeplitz matrix by multiplying an anti-diagonal matrix of ones and vice versa.

This section explains how to derive a fast computation for $R[x]/\langle x^n - \alpha x - \beta \rangle$ by looking at an already well-studied algebra homomorphism $f$ multiplying two size-$n$ polynomials in $R[x]$. There are four steps: (i) interpreting multiplication in $R[x]/\langle x^n - \alpha x - \beta \rangle$ as a Toeplitz matrix-vector product; (ii) interpreting the Toeplitz matrix-vector product as a composition of applying an anti-diagonal matrix of ones and a Hankel matrix–vector product; (iii) rewriting the Hankel matrix–vector product as a bilinear system of the form $S_2$; and (iv) converting the computing task into a bilinear system of the form $S_0$. Once we go through all the steps (i) – (iv), we can now convert an $f$ into an algorithm for $R[x]/\langle x^n - \alpha x - \beta \rangle$ via the module–theoretic view. Notice that steps (ii) and (iii) are sometimes described as a single step. We describe them separately for clarity.

Steps (i) – (iii) are already shown in previous paragraphs. We now explain how to interpret the multiplication in $R[x]/\langle x^n - \alpha x - \beta \rangle$ as a Toeplitz matrix-vector product with potential post-processing. We define $\mathbf{Toeplitz}_n$ as the following function mapping a $(2n-1)$-tuple drawn from $R$ to a Toeplitz matrix over $R$:

$$\mathbf{Toeplitz}_n : (z_{2n-2}, \dots, z_0) \mapsto \begin{pmatrix} z_{n-1} & \cdots & z_0 \\ \vdots & \ddots & \ddots \\ z_{2n-2} & \ddots & \ddots \end{pmatrix}.$$

Let $\boldsymbol{a} = \sum_i a_i x^i, \boldsymbol{b} = \sum b_i x^i$ be size-$n$ polynomials. We recall that computing $\sum_i c_i x^i =$

$\boldsymbol{ab}$ in $R[x]$ can be regarded as the following matrix–vector product:

$$\begin{pmatrix} c_0 \\ \vdots \\ c_{2n-2} \end{pmatrix} = \begin{pmatrix} b_0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots \\ b_{n-1} & \ddots & \ddots & \ddots \\ 0 & \ddots & \ddots & \ddots \\ \vdots & \ddots & \ddots & \ddots \\ 0 & \ddots & \ddots & \ddots \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix}.$$

Since $(c_0, \ldots, c_{n-1})$ can be computed with a Toeplitz matrix-vector product, we only need to convert reduction modulo $x^n - \alpha x - \beta$ into the manipulation of Toeplitz matrices. A standard approach for reducing modulo $x^n - \alpha x - \beta$ is multiplying $(c_n, \ldots, c_{2n-2})$ by $\alpha$ and $\beta$ and adding the results to $(c_1, \ldots, c_{n-1})$ and $(c_0, \ldots, c_{n-2})$. Based on this, $\boldsymbol{ab} \bmod (x^n - \alpha x - \beta)$ can be written as

$$(M_0 + M_1 + M_2)\,\boldsymbol{a}$$

where

$$\begin{cases} M_0 = \textbf{Toeplitz}_n(b_{n-1}, \ldots, b_0, 0, \ldots, 0), \\ M_1 = \textbf{Toeplitz}_n(0, \ldots, 0, \beta b_{n-1}, \ldots, \beta b_1), \end{cases}$$

and

$$M_2 = \alpha \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & b_{n-1} & \cdots & b_0 \\ \vdots & \ddots & \ddots & \ddots \\ 0 & \ddots & \ddots & \ddots \end{pmatrix}.$$

**A specialized approach for $\beta = 1$.** We review the case $\beta = 1$ implied by [FH07, Section 3.2]. See [HB95, FD05] for related works when $R = \mathbb{F}_2$ and Appendix I for an approach handling generic $\beta$ with some overhead. Since $\beta = 1$, $M_0 + M_1$ is the circulant matrix implementing $\boldsymbol{ab} \bmod (x^n - 1)$. Obviously, if we multiply a circulant matrix by a cyclic shift matrix (either left-multipling or right-multipling), we still have a circulant matrix. Let $P$ be the matrix moving the 0-th row of a circulant matrix to the bottom. We find that both $P(M_0 + M_1)$ and $PM_2$ are Toeplitz matrices. Therefore, $P(M_0 + M_1 + M_2)$ is a Toeplitz matrix and we can implement $(M_0 + M_1 + M_2)\,\boldsymbol{a}$ as

$$(M_0 + M_1 + M_2)\,\boldsymbol{a} = P^{-1}\left(P\left(M_0 + M_1 + M_2\right)\boldsymbol{a}\right).$$

In Section 6.2, we will justify why cyclic shift matrices are practically efficient.

**Padding.** The last instrument is padding. Suppose we have an $n \times n$ Toeplitz matrix $T = \textbf{Toeplitz}\,(z_{2n-2}, \ldots, z_0)$. For an $n' \geq n$, we can always pad $T$ to an $n' \times n'$ Toeplitz matrix $T'$ as follows:

$$T' = \textbf{Toeplitz}\left(\underbrace{0, \ldots, 0}_{n'-n}, z_{2n-2}, \ldots, z_0, \underbrace{0, \ldots, 0}_{n'-n}\right).$$

The point is that if a $n \times n$ Toeplitz matrice does not admit efficient implementations, we can pad them to slighly larger ones with efficient implementations [IKPC22, Section 3.1].

# 6 Vectorization

This section reviews the formalization of vectorization by [Hwa24].

## 6.1 Vector Instruction Sets/Extensions

Vector instruction sets and extensions are important ingredients for optimized implementations since high-dimensional polynomial rings in lattice-based cryptosystems admit high degree of parallelism. Common vector instructions sets and extensions are the Digital Signal Processing extension for Armv7-M [ARM21b, Section A1.3] (in this case, we also call it Armv7E-M), the Neon extension for Armv7-A [ARM12, Section A2.6], the Advanced SIMD instructions in Armv8-A [ARM21a, Section A1.5], the Helium extension for Armv8-M [ARM23, Sections B5 and C2.3.1], AVX2 and AVX-512 for `x86` [Int23].

A vector contains power-of-two number of bits of data. We have 32-bit general purpose registers in Armv7E-M, 128-bit vector registers in Armv7-A and Armv8-A, 256-bit vector registers in AVX2, and 512-bit vector registers in AVX-512. In a vector instruction set/extension, we have vector instructions with vector registers (or general purpose registers in Armv7E-M) as arguments. For bit-field arithmetic, we have logical or/exclusive-or/and/not and logical/arithmetic shift. For arithmetic and permutations, we have additions, subtractions, multiplications, and permutations operating on vector registers as packed 8-bit, 16-bit, 32-bit, and 64-bit data. When the context is clear, we denote $v$ as the number of elements in a vector register.

Roughly speaking, there are two categories of vector instructions:

- Vector-by-vector instructions: the vector instruction takes two vector registers as inputs and returns a vector register as the output.
- Vector-by-scalar instructions: the vector instruction takes a vector register and a scalar (a constant, a lane of a vector register [ARM21b, ARM12, ARM21a], or a general purpose register [ARM23]) and returns a vector register as the output.

## 6.2 Vectorization-Friendliness

We first review the notion "vectorization-friendliness" formally relating homomorphisms to vector-by-vector instructions [Hwa24]. Conceptually, vectorization-friendliness qualifies if a homomorphism can be mapped to a string of vector-by-vector instructions and cyclic/negacyclic shifts. Cyclic and negacyclic shfits are vectorization-friendly since we can implement them with extractions or memory operations:

- Extractions: For cyclic shift, we extract consecutive elements from a pair of vector registers and extract again with inputs swapped. The resulting pair of vector registers is now a cyclic shift of the original pair. For the negacyclic shift, we replace a vector register by its negative value in one of the extractions. This idea is applicable to Armv7/8-A since we have `ext` implementing exactly the desired operations [HLY24].
- Memory operations: We can also implement cyclic/negacyclic shift with memory operations – we perform unaligned loads, shuffle the last vector register (and negate it in the negacyclic case), and store the vectors to memory [BBCT22].

**The set `BlockDiag`.** We define `BlockDiag` as a certain set of block diagonal matrices implementing cyclic/negacyclic shifts and twisting. Formally, `BlockDiag` is defined as a set of all possible block diagonal matrices with each block a $v' \times v'$ matrix that is a diagonal matrix implementing twisting or a cyclic/nagecyclic shift matrix for all $v$-multiple $v'$.

**Vectorization-friendliness, formally [Hwa24].** Let $f$ be an algebra homomorphism and $M_f$ its matrix form. We call $f$ vectorization-friendly if

$$M_f = \prod_i \left(M_{f_i} \otimes I_v\right) S_{f_i}$$

for $S_{f_i} \in \texttt{BlockDiag}$ and some matrices $M'_{f_i}s$. Once we find such a decomposition for a vectorization-friendly $f$, we implement $M_{f_i} \otimes I_v$ with vector additions, subtractions, and multiplications, and $S_{f_i}$ with vector multiplications and cyclic/negacyclic shifts.

**Dimension requirement of vectorization-friendliness.** From the definition, we know that $f$ is vectorization-friendly only if its domain has dimension a multiple of $v$.

**Additional properties of vectorization-friendliness.** Obviously, if an algebra homomorphism is vectorization-friendly, its inverse (if exists) and module-theoretic dual are also vectorization-friendly.

## 6.3 Permutation-Friendliness

Conceptually, permutation-friendliness stands for vectorization-friendliness up to a special type of permutation – interleaving.

**The set `Interleave` [Hwa24].** Again, let $v'$ be a multiple of $v$. We define the transposition matrix $T_{v'^2}$ as the $v'^2 \times v'^2$ matrix permuting the elements as if transposing a $v' \times v'$ matrix. See Appendix K for examples. We call a matrix $M$ interleaving matrix with step $v'$ if it takes the form

$$M = \left(\pi' \otimes I_{v'}\right)\left(I_m \otimes T_{v'^2}\right)\left(\pi \otimes I_{v'}\right)$$

for a positive integer $m$ and permutation matrices $\pi, \pi'$ permuting $mv'$ elements. The set `Interleave` consists of interleaving matrices of all possible steps. Obviously, we can implement an interleaving matrix as a transposition matrix with on-the-fly permutations.

**Permutation-friendliness, formally [Hwa24].** Let $g$ be an algebra homomorphism and $M_g$ its matrix form. We call $g$ permutation-friendly if

$$M_g = \prod_i S_{g_i} M_{g_i}$$

for $S_{g_i} \in \texttt{Interleave}$ and vectorization-friendly $M_{g_i}$'s. Once we find such a decomposition of a permutation-friendly $g$, we implement the vectorization-friendly parts as described in previous section and the interleaving matrices with permutation instructions.

**Dimension requirement of permutation-friendliness.** By definition, permutation-friendliness necessitates a stronger dimension condition than vectorization-friendliness due to the existence of interleaving matrices. Interleaving matrices necessitates that a permutation-friendly homomorphism must have dimension a multiple of $v^2$.

## 6.4 Guide of Vectorization

### 6.4.1 Vectorization with Vector-By-Vector Multiplication Instructions

Generally, while computing with vector-by-vector instructions, we choose algebra homomorphisms $f$ and $g$ such that $f$ is vectorization-friendly and $g$ is permutation-friendly. Their composition $g \circ f$ then admits a suitable mapping to the target vector instruction set.

### 6.4.2 Vectorization Vector-By-Scalar Multiplication Instructions

For an $m \times n$ Toeplitz matrix $M = \textbf{Toeplitz}_{m\times n}(a_{m-1}, \dots, a_0, a'_1, \dots, a'_{n-1})$ over the ring $R$, [CCHY24] demonstrated the benefit of vector-by-scalar multiplication instructions when applying $M$ to a vector $\boldsymbol{b} = (b_0, \dots, b_{n-1})$. Since polynomial multiplications in $R[x]/\langle x^n - \alpha x - \beta\rangle$ can be rephased as Toeplitz matrix-vector products (cf. Section 5.5.2 and Appendix I), we can multiply polynomials in $R[x]/\langle x^n - \alpha x - \beta\rangle$ with vector-by-scalar multiplication instructions. Conceptually, the goal is to design transformations resulting in small-dimensional Toeplitz matrix-vector products and implement them with vector-by-scalar multiplication instructions. We outline the overall strategy as follows.

1. Choose a vectorization-friendly algebra homomorphism $f$ decomposing into small-dimensional polynomial multiplications.

2. If the resulting polynomial multiplications are small-dimensional Toeplitz matrix-vector products, then we implement them with vector-by-scalar multiplication instructions.

3. If step 2 doesn't hold (for example, when some polynomial multiplications in the image are not Toeplitz matrix-vector products), we dualize the transformation as described in Section 5.5.2.

4. Eventually, we have small-dimensional Toeplitz matrix-vector products regardless if $f$ if results in small-dimensional Toeplitz matrix-vector products.

The remaining question is the relation between small-dimensional Toeplitz matrix-vector products and vector-by-scalar multiplication instructions.

**Small-dimensional Toeplitz matrix-vector products [CCHY24].** For the small-dimensional case, [CCHY24] showed that one can implement the Toeplitz matrix-vector product efficiently with vector-by-scalar multiplication instructions. For simplicity, we demonstrate with the case $m = n = 4$ and $R = \mathbb{Z}_{2^{32}}$:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} a_0 & a'_1 & a'_2 & a'_3 \\ a_1 & a_0 & a'_1 & a'_2 \\ a_2 & a_1 & a_0 & a'_1 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

For deploying vector-by-scalar multiplications, the key is to identify the reuses of the scalar operands. Obviously, we find that each of $b_0, \dots, b_3$ is involved in four multiplications in $R$: we compute $a_0b_0$, $a_1b_0$, $a_2b_0$, $a_3b_0$ for the operand $b_0$, etc. Therefore, an obvious choice is to map each columns to a vector and apply vector-by-scalar multiplications. There are two ways for constructing the column vectors of $\textbf{Toeplitz}(a_3, \cdots, a_0, a'_1, \cdots, a'_3)$ from an array storing $a'_3, \dots, a'_1, a_0, \dots, a_3$: either loading from the addresses pointing to $a_0, a'_1, \dots, a_3$, or loading the first column and first row and combining them with special instructions. See Appendix L for illustrations. After constructing the matrix column-wise, we now identify the column vector $c$ as the sum of columns scaled by the corresponding elements in $\boldsymbol{b}$. In other words,

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = b_0 \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} + b_1 \begin{pmatrix} a'_1 \\ a_0 \\ a_1 \\ a_2 \end{pmatrix} + b_2 \begin{pmatrix} a'_2 \\ a'_1 \\ a_0 \\ a_1 \end{pmatrix} + b_3 \begin{pmatrix} a'_3 \\ a'_2 \\ a'_1 \\ a_0 \end{pmatrix}.$$

Algorithm 1 is an illustration.

**Algorithm 1** Applying a $4 \times 4$ Toeplitz matrix with vector-by-scalar multiplication instructions [CCHY24].

**Inputs: Toeplitz**$(a_3, a_2, a_1, a_0, a'_1, a'_2, a'_3)$, $(b_0, b_1, b_2, b_3)$.
**Outputs: Toeplitz**$(a_3, a_2, a_1, a_0, a'_1, a'_2, a'_3)(b_0, b_1, b_2, b_3)$.

```
t0 = $a_3||a_2||a_1||a_0$
t1 = $a_2||a_1||a_0||a'_1$
t2 = $a_1||a_0||a'_1||a'_2$
t3 = $a_0||a'_1||a'_2||a'_3$
c = mul(t0, $b_0$)
c = mla(c, t1, $b_1$)
c = mla(c, t2, $b_2$)
c = mla(c, t3, $b_3$)
```

# 7 Case Studies

We go through several case studies in this section. Section 7.1 compares Barrett and Montgomery multiplications with Dilithium implementations for modular arithmetic, and Section 7.2 compares Montgomery and Plantard multiplications with Kyber implementations. We then go through several algebraic techniques and vectorization. Section 7.3 explains how to exploit the matrix-to-vector structure with Saber as an example, Section 7.4 reviews the benefit of Toeplitz matrix-vector multiplication with NTRU as an example, and Section 7.5 details the design choices for vectorization with NTRU Prime as an example.

## 7.1 Dilithium : Barrett vs Montgomery Modular Arithmetic

This section reviews the modular arithmetic used in Dilithium [ABD+20a]. Dilithium is a lattice-based digital signature recently selected by the National Institute of Standardization and Technology (NIST) for standardization. In Dilithium, we want to multiply polynomials in $\mathbb{Z}_q[x]/\langle x^{256} + 1 \rangle$ for $q = 2^{23} - 2^{13} + 1$. Since $q$ is a prime supporting a size-$2^{13}$ cyclic FFT, we can split $x^{256} + 1$ into linear factors (notice that $x^{256} + 1 = \Phi_{512} | (x^{512} - 1)$ and $512 | (q - 1)$). The choice of FFT is already determined by the specification – one of the operands is assumed to be transformed. The remaining question is to compute the modular arithmetic efficiently. For a 32-bit value $a$, modular reduction is fairly simple. Since $q$ is fairly close to $2^{23}$, $a - \left\lfloor \frac{a}{2^{23}} \right\rceil q$ is a representative of $a \bmod {}^{\pm}q$ within an acceptable range[8].

How about modular multiplications? In Section 2.5, we compare three classes of modular multiplications – Montgomery, Barrett, and Plantard, and review the required multiplication instructions. In practice, low multiplications are fairly common, while high multiplications and long multiplications usually lack accumulative or subtractive variants. See Table 14 for a summary for combinations of precisions and architectures. For the actual instructions, see [ARM21b, Section A4.4.3], [ARM21a, Sections C3.5.14, C3.5.16, and C3.5.18], and [Ora14, Section 3.7].

### 7.1.1 Armv8-A Neon Implementations

For vectorized implementations, [BHK+22b] implemented Barrett multiplication and the subtractive variant of Montgomery multiplication with Armv8.0-A Neon. For Armv8.0-A, there are multiplication instructions `sq{, r}dmulh` computing and doubling the high products – For two values $a$ and $b$, `sqdmulh` computes $\left\lfloor \frac{2ab}{\mathtt{R}} \right\rfloor$ with saturations, and `sqrdmulh`

[8] When $-2^{31} \leq a < 2^{31}$, we have $-4186113 \leq a - \left\lfloor \frac{a}{2^{23}} \right\rceil q \leq 4194303$ by brute-force testing.

Table 14: Overview of the available forms of input-independent signed multiplication instructions in some popular instruction set architectures and extensions.

| Low multiplications | | | |
|---|---|---|---|
| | `mullo` | `mlalo` | `mlslo` |
| Armv7-M | ✓($\mathtt{R} = 2^{32}$) | ✓($\mathtt{R} = 2^{32}$) | ✓($\mathtt{R} = 2^{32}$) |
| Armv7E-M | ✓($\mathtt{R} = 2^{32}$) | ✓($\mathtt{R} = 2^{32}$) | ✓($\mathtt{R} = 2^{32}$) |
| Armv8.0-A | ✓($\mathtt{R} = 2^{8}, 2^{16}, 2^{32}$) | ✓($\mathtt{R} = 2^{8}, 2^{16}, 2^{32}$) | ✓($\mathtt{R} = 2^{8}, 2^{16}, 2^{32}$) |
| Armv8.1-A | ✓($\mathtt{R} = 2^{8}, 2^{16}, 2^{32}$) | ✓($\mathtt{R} = 2^{8}, 2^{16}, 2^{32}$) | ✓($\mathtt{R} = 2^{8}, 2^{16}, 2^{32}$) |
| AVX2 | ✓($\mathtt{R} = 2^{16}, 2^{32}$) | - | - |
| High multiplications | | | |
| | `mulhi` | `mlahi` | `mlshi` |
| Armv7-M | - | - | |
| Armv7E-M | ✓($\mathtt{R} = 2^{32}$) | - | - |
| Armv8.0-A | ✓($\mathtt{R} = 2^{16}, 2^{32}$) | - | - |
| Armv8.1-A | ✓($\mathtt{R} = 2^{16}, 2^{32}$) | ✓($\mathtt{R} = 2^{16}, 2^{32}$) | ✓($\mathtt{R} = 2^{16}, 2^{32}$) |
| AVX2 | ✓($\mathtt{R} = 2^{16}$) | - | - |
| Long multiplications | | | |
| | `mull` | `mlal` | `mlsl` |
| Armv7-M | - | - | - |
| Armv7E-M | ✓($\mathtt{R} = 2^{16}, 2^{32}$) | ✓($\mathtt{R} = 2^{16}, 2^{32}$) | - |
| Armv8.0-A | ✓($\mathtt{R} = 2^{8}, 2^{16}, 2^{32}$) | ✓($\mathtt{R} = 2^{8}, 2^{16}, 2^{32}$) | ✓($\mathtt{R} = 2^{8}, 2^{16}, 2^{32}$) |
| Armv8.1-A | ✓($\mathtt{R} = 2^{8}, 2^{16}, 2^{32}$) | ✓($\mathtt{R} = 2^{8}, 2^{16}, 2^{32}$) | ✓($\mathtt{R} = 2^{8}, 2^{16}, 2^{32}$) |
| AVX2 | ✓($\mathtt{R} = 2^{32}$) | - | - |

applies rounding $\lfloor\rceil$ instead of flooring $\lfloor\rfloor$. For Montgomery multiplication, [BHK+22b] implemented

$$\frac{1}{2}\left(\left\lfloor\frac{2ab}{\mathtt{R}}\right\rfloor - \left\lfloor\frac{2\left(abq^{-1} \bmod {}^{\pm}\mathtt{R}\right)q}{\mathtt{R}}\right\rfloor\right)$$

as shown in Algorithm 2. One can show that $\frac{1}{2}\left(\left\lfloor\frac{2ab}{\mathtt{R}}\right\rfloor - \left\lfloor\frac{2\left(abq^{-1} \bmod {}^{\pm}\mathtt{R}\right)q}{\mathtt{R}}\right\rfloor\right) = \left\lfloor\frac{ab}{\mathtt{R}}\right\rfloor - \left\lfloor\frac{\left(abq^{-1} \bmod {}^{\pm}\mathtt{R}\right)q}{\mathtt{R}}\right\rfloor$. We leave the justification to readers[9]. For Barrett multiplication, [BHK+22b] implemented

$$\left(\left(ab \bmod {}^{\pm}\mathtt{R}\right) - \left(\left\lfloor\frac{a\left\lfloor\frac{b\mathtt{R}}{q}\right\rceil_2}{\mathtt{R}}\right\rceil q \bmod {}^{\pm}\mathtt{R}\right)\right) \bmod {}^{\pm}\mathtt{R}$$

for $\lfloor\rceil_2 \coloneqq r \mapsto 2\left\lfloor\frac{r}{2}\right\rceil$ as shown in Algorithm 3. Since there is a subtractive form for low multiplications only, Barrett multiplication saves one addition/subtraction compared to Montgomery multiplication. Additionally, [HLY24, Algorithms 3 and 4] proposed the accumulative and subtractive variants computing representatives of $d \pm ab \bmod q$ and [Yan22] found their benefits for computing radix-2 Cooley–Tukey butterflies on platforms implementing barrel shift (for example, Cortex-M4). We leave the exploration of the accumulative/subtractive Barrett multiplication to the readers.

[9] Apply McEliece's observation that for a continuous, monotonically increasing function $f : \mathbb{R}' \to \mathbb{R}''$ with $\mathbb{R}', \mathbb{R}'' \subset \mathbb{R}$ and $f(x) \in \mathbb{Z} \longrightarrow x \in \mathbb{Z}$, we have $\lfloor f(x)\rfloor = \lfloor f(\lfloor x\rfloor)\rfloor$ when $f(x), f(\lfloor x\rfloor) \in \mathbb{R}'$ and $\lceil f(x)\rceil = \lceil f(\lceil x\rceil)\rceil$ when $f(x), f(\lceil x\rceil) \in \mathbb{R}'$ [GKP94, Equation 3.10].

**Algorithm 2** Single-width Montgomery multiplication [BHK+22b, Algorithm 12].

**Inputs:** Values $a, b \in \left[-\frac{\mathtt{R}}{2}, \frac{\mathtt{R}}{2}\right)$.

**Output:** $\mathtt{c} = \frac{1}{2}\left(\left\lfloor \frac{2ab}{\mathtt{R}} \right\rfloor - \left\lfloor \frac{2\left(abq^{-1} \bmod {}^{\pm}\mathtt{R}\right)q}{\mathtt{R}} \right\rfloor\right)$.

```
sqdmulh   c,  $a$,  $b$                        ▷ $\mathtt{c} = \left\lfloor \frac{2ab}{\mathtt{R}} \right\rfloor$.
mul       t,  $a$,  $bq^{-1} \bmod {}^{\pm}\mathtt{R}$       ▷ $\mathtt{t} = abq^{-1} \bmod {}^{\pm}\mathtt{R}$.
sqdmulh   t,  t,  $q$                        ▷ $\mathtt{t} = \left\lfloor \frac{2\left(abq^{-1} \bmod {}^{\pm}\mathtt{R}\right)q}{\mathtt{R}} \right\rfloor$.
shsub     c,  c,  t                        ▷ $\mathtt{c} = \frac{1}{2}\left(\left\lfloor \frac{2ab}{\mathtt{R}} \right\rfloor - \left\lfloor \frac{2\left(abq^{-1} \bmod {}^{\pm}\mathtt{R}\right)q}{\mathtt{R}} \right\rfloor\right)$.
```

**Algorithm 3** Single-width Barrett multiplication [BHK+22b, Algorithm 10].

**Inputs:** Values $a, b \in \left[-\frac{\mathtt{R}}{2}, \frac{\mathtt{R}}{2}\right)$.

**Output:** $\mathtt{lo} = ab - \left\lfloor \frac{a \left\lfloor \frac{b\mathtt{R}}{q} \right\rceil_2}{\mathtt{R}} \right\rceil q$.

```
mul       lo,  $a$,  $b$                       ▷ $\mathtt{lo} = ab$.
sqrdmulh  hi,  $a$,  $\frac{\left\lfloor \frac{b\mathtt{R}}{q} \right\rceil_2}{2}$               ▷ $\mathtt{hi} = \left\lfloor \frac{a \left\lfloor \frac{b\mathtt{R}}{q} \right\rceil_2}{\mathtt{R}} \right\rceil$.
mls       lo,  hi,  $q$                      ▷ $\mathtt{lo} = ab - \left\lfloor \frac{a \left\lfloor \frac{b\mathtt{R}}{q} \right\rceil_2}{\mathtt{R}} \right\rceil q$.
```

### 7.1.2 Armv7-M Implementations

This section reviews [HKS23]'s observation of Barrett multiplication on Cortex-M3. Cortex-M3 implements the ISA Armv7-M where `mul/mla/mls`, `{u, s}{mul, mla}l` are the only multiplication instructions. However, double-size multiplications `{u, s}{mul, mla}l` take input-dependent time [ARM10] and can only be used for computing public data. In the literature, [GOPT10] showed that input-dependent time multiplications enable straightforward Simple Power Analysis (SPA) attacks. For computing the 32-bit NTTs of secret data in Dilithium, [GKS21] implemented 32-bit Montgomery multiplication while emulating the double-size multiplications with `mul/mla/mls` as shown in Algorithm 4.

**Algorithm 4** Constant-time 32-bit Montgomery multiplication [GKS21, Listing 7]

**Inputs:** $\mathtt{al} + \mathtt{ah} \cdot \mathtt{R} = a, \mathtt{bl} + \mathtt{bh} \cdot \mathtt{R} = b$.

**Output:** $\mathtt{tmph} = \frac{ab + \left(-abq^{-1} \bmod {}^{\pm}\mathtt{R}\right)q}{\mathtt{R}}$.

```
SBSMULL tmpl, tmph, al, ah, bl, bh       ▷ $\mathtt{tmpl} + \mathtt{tmph} \cdot \mathtt{R} = ab$.
mul         ah, tmpl, $-q^{-1} \bmod {}^{\pm}\mathtt{R}$       ▷ $\mathtt{ah} = abq^{-1} \bmod {}^{\pm}\mathtt{R}$.
ubfx        al,   ah,  #0, #16
asr         ah,   ah,  #16              ▷ $\mathtt{al} + \mathtt{ah} \cdot \mathtt{R} = -abq^{-1} \bmod {}^{\pm}\mathtt{R}$.
SBSMLAL tmpl, tmph, al, ah, ql, qh       ▷ $\mathtt{tmph} = \frac{ab + \left(-abq^{-1} \bmod {}^{\pm}\mathtt{R}\right)q}{\mathtt{R}}$.
```

[HKS23] proposed using Barrett multiplication for 32-bit modular multiplications on Cortex-M3. They observed the following:

B1 While Montgomery multiplication requires two double-size/high multiplications and one low multiplication, Barrett multiplication requires one high multiplication and two low multiplications.

B2 In Barrett multiplication, the high multiplication only needs to be approximately correct.

Observation B1 saves one emulation of the double-size/high multiplication, and observation B2 enables a faster emulation with tolerable errors.

Let's consider $\llbracket\rrbracket$ the following integer approximation:

$$\forall r \in \mathbb{R}, \llbracket r \rrbracket = a_{r,h} b_h + \left\lfloor \frac{a_{r,l} b_h}{\sqrt{\mathtt{R}}} \right\rfloor + \left\lfloor \frac{a_{r,h} b_l}{\sqrt{\mathtt{R}}} \right\rfloor$$

for $a_{r,l} + a_{r,h}\sqrt{\mathtt{R}} = \frac{r\mathtt{R}}{\left\lfloor \frac{b\mathtt{R}}{q} \right\rceil}$ and $b_l + b_h\sqrt{\mathtt{R}} = \left\lfloor \frac{b\mathtt{R}}{q} \right\rceil$. For $-\frac{q}{2} \leq b < \frac{q}{2}$ and $-\frac{\mathtt{R}}{2} \leq a_{r,l} + a_{r,h}\sqrt{\mathtt{R}} < \frac{\mathtt{R}}{2}$, [HKS23] showed that $|r - \llbracket r \rrbracket| \leq 3$ and $\left|\mathrm{mod}^{\llbracket\rrbracket} \mathtt{R}\right| \leq \frac{7\mathtt{R}}{2}$, and computed

$$ab - \left\llbracket \frac{a \left\lfloor \frac{b\mathtt{R}}{q} \right\rceil}{\mathtt{R}} \right\rrbracket q$$

as a representative of $ab \bmod {}^{\pm} q$ with range bounded by

$$\frac{|a| \left|\mathrm{mod}^{\pm} q\right| + \left|\mathrm{mod}^{\llbracket\rrbracket} \mathtt{R}\right| |q|}{\mathtt{R}} \leq \frac{|a| \frac{q}{2} + \frac{7}{2}\mathtt{R}q}{\mathtt{R}} = \frac{q}{2}\left(7 + \frac{|a|}{\mathtt{R}}\right).$$

Algorithm 5 is an illustration.

**Algorithm 5** Constant-time 32-bit Barrett multiplication with approximate high multiplication [HKS23].

**Inputs:** $\mathtt{a} = a, \mathtt{b} = b$.

**Output:** $\mathtt{t3} = ab - \left\llbracket \frac{a \left\lfloor \frac{b\mathtt{R}}{q} \right\rceil}{\mathtt{R}} \right\rrbracket q$.

```
mul            t3,  a,   b                      ▷ $\mathtt{t3} = ab \bmod {}^{\pm}\mathtt{R}$.
ubfx           t0,  a,  #0, #16
asr             a,  a, #16                      ▷ $\mathtt{t0} + \mathtt{a} \cdot \mathtt{R} = a$.
smmulr_approx  t1,  a, bhi, t0, blo, t2         ▷ $\mathtt{t1} = \left\llbracket \frac{a \left\lfloor \frac{b\mathtt{R}}{q} \right\rceil}{\mathtt{R}} \right\rrbracket$.
mls            t3, t1,   $q$, t3                ▷ $\mathtt{t3} = ab - \left\llbracket \frac{a \left\lfloor \frac{b\mathtt{R}}{q} \right\rceil}{\mathtt{R}} \right\rrbracket q$.
```

## 7.2 Kyber : Montgomery vs Plantard Modular Arithmetic

In this section, we compare the applications of Montgomery and Plantard multiplications to Kyber [ABD+20b], a lattice-based key encapsulation mechanism (KEM) recently selected by NIST for standardization. In Kyber, we want to multiply polynomials in $\mathbb{Z}_{3329}[x]/\left\langle x^{256} + 1 \right\rangle$ via the FFT $\mathbb{Z}_{3329}[x]/\left\langle x^{256} + 1 \right\rangle \cong \prod_i \mathbb{Z}_{3329}[x]/\left\langle x^2 - \omega_{256}^{2i+1} \right\rangle$. We review the modular multiplications for the coefficeint ring $\mathbb{Z}_{3329}$ where all the elements are stored as 16-bit integers. We assume $\mathtt{R} = 2^{16}$ in this section. [HZZ+22] and [AMOT22] independently found the signed Plantard multiplications. [HZZ+22] identified the benefit of 16-bit modular arithmetic if there are 16×32-bit multiplication instructions, and [AMOT22] identified the benefit of 32-bit modular arithmetic using only 64-bit multiplication instructions. [HZZ+24] later implemented 16-bit Plantard multiplication following [AMOT22]'s insights when there are no 16 × 32-bit multiplication instructions.

### 7.2.1 Armv7-M Implementations

In Armv7-M, since all the registers contain 32-bit values, we can compute the 16-bit Montgomery multiplication with `mul` and `mla` in an obvious way (cf. Algorithm 6).

[HZZ+24] implemented 16-bit Plantard multiplication with [AMOT22]'s insights. For 16-bit values $a \in \left[-\frac{\mathtt{R}}{2}, -\frac{\mathtt{R}}{2}\right)$ and $b \in \left[-\frac{q}{2}, \frac{q}{2}\right)$, we compute

$$\left\lfloor \frac{\left\lfloor \frac{-abq^{-1} \bmod {}^{\pm}\mathtt{R}^2}{\mathtt{R}} \right\rfloor q + 2^{\alpha} q}{\mathtt{R}} \right\rfloor$$

as a representative of $-ab\mathtt{R}^{-2} \bmod {}^{\pm}q$. See Algorithm 7 for an illustration. If $b$ is known in prior, we skip the computation for $-bq^{-1} \bmod {}^{\pm}\mathtt{R}^2$ and cancel out the scaling $-\mathtt{R}^2 \bmod {}^{\pm}q$ by precomputing $-\left(-b\mathtt{R}^2 \bmod {}^{\pm}q\right) q^{-1} \bmod {}^{\pm}\mathtt{R}^2$.

**Algorithm 6** 16-bit Montgomery multipliation with Armv7-M [GKS21].

**Inputs:** Values $a, b \in \left[-\frac{\mathtt{R}}{2}, \frac{\mathtt{R}}{2}\right)$.
**Output:** $\mathtt{t0} = ab + \left(-abq^{-1} \bmod {}^{\pm}\mathtt{R}\right) q$.

```
mul t0, $a$, $b$                              ▷ $\mathtt{t0} = ab$.
mul t1, t0, $-q^{-1} \bmod {}^{\pm}\mathtt{R}$
sxth t1, t1, #0, #16                          ▷ $\mathtt{t1} = -abq^{-1} \bmod {}^{\pm}\mathtt{R}$.
mla t0, t1, $q$, t0                           ▷ $\mathtt{t0} = ab + \left(-abq^{-1} \bmod {}^{\pm}\mathtt{R}\right) q$.
                                              ▷ The desired result is stored in the upper half.
```

**Algorithm 7** 16-bit Plantard multiplication with Armv7-M [HZZ+24].

**Inputs:** Values $a \in \left[-\frac{\mathtt{R}}{2}, \frac{\mathtt{R}}{2}\right), -bq^{-1} \in \left[-\frac{\mathtt{R}^2}{2}, \frac{\mathtt{R}^2}{2}\right)$.
**Output:** $\mathtt{t} = \left(\left\lfloor \frac{-abq^{-1} \bmod {}^{\pm}\mathtt{R}^2}{\mathtt{R}} \right\rfloor + 2^{\alpha}\right) q$.

```
mul t, $b$, $-q^{-1} \bmod {}^{\pm}\mathtt{R}^2$        ▷ $\mathtt{t} = -bq^{-1} \bmod {}^{\pm}\mathtt{R}^2$.
mul t, t, $a$                                   ▷ $\mathtt{t} = -abq^{-1} \bmod {}^{\pm}\mathtt{R}^2$.
add t, $2^{\alpha}$, t, asr #16                       ▷ $\mathtt{t} = \left\lfloor \frac{-abq^{-1} \bmod {}^{\pm}\mathtt{R}^2}{\mathtt{R}} \right\rfloor + 2^{\alpha}$.
mul t, t, $q$                                   ▷ $\mathtt{t} = \left(\left\lfloor \frac{-abq^{-1} \bmod {}^{\pm}\mathtt{R}^2}{\mathtt{R}} \right\rfloor + 2^{\alpha}\right) q$.
                                                ▷ The desired result is stored in the upper half.
```

### 7.2.2 Armv7E-M Implementations

We briefly compare Montgomery and Plantard multiplications with the Digital Signal Processing extension in Armv7E-M where "E" stands for "extension". [ABCG20] showed that 16-bit Montgomery multiplication can be implemented with three 16-bit multiplication instructions from the extension as shown in Algorithm 8. Recently, [HZZ+22] found that the multiplication instruction `smulwb` is a nice fit for 16-bit Plantard multiplication. Algorithm 9 is an illustration. If one of the multiplicands is known in prior, we can remove one multiplication and cancel out the scaling with precomputation as shown in previous section.

### 7.2.3 Application to NewHope

NewHope [PAA+20] is a second round lattice-based KEM candidate in the NIST Post-Quantum Cryptography Standardization. In NewHope, we have to multiply polynomials in $\mathbb{Z}_{12289}[x]/\langle x^n + 1\rangle$ for $n = 512, 1024$. Since NewHope also mandates the use of the NTT $\mathbb{Z}_{12289}[x]/\langle x^n + 1\rangle \cong \prod_i \mathbb{Z}_{12289}[x]/\langle x - \omega_{2n}^{2i+1}\rangle$ and elements in $\mathbb{Z}_{12289}$ can be stored as 16-bit integers, the overall optimization strategy essentially follows from optimizing Kyber.

**Algorithm 8** 16-bit Montgomery multiplication with Armv7E-M [ABCG20].

```
Inputs: lo(a) $= a_l$, lo(b) $= b_l$.
Outputs: hi(th) $= \frac{a_l b_l + \left(-a_l b_l q^{-1} \bmod^{\pm} \mathtt{R}\right) q}{\mathtt{R}}$.
smulbb th,  a, b                          ▷ hi $= a_l b_l$.
smulbb tl, th, $-q^{-1} \bmod^{\pm} \mathtt{R}$      ▷ lo $= (a_l b_l \bmod^{\pm} \mathtt{R}) \left(-q^{-1} \bmod^{\pm} \mathtt{R}\right)$.
smlabb th, tl, $q$, th                    ▷ th $= a_l b_l + \left(-a_l b_l q^{-1} \bmod^{\pm} \mathtt{R}\right) q$.
                                          ▷ The desired result is stored in the upper half.
```

**Algorithm 9** 16-bit Plantard multiplication with Armv7E-M [HZZ+22]

```
Inputs: lo(a) $= a_l, b \in \left[-\frac{q}{2}, \frac{q}{2}\right)$.
Outputs: hi(t) $= \left\lfloor \frac{\left\lfloor \frac{-a_l b q^{-1} \bmod^{\pm} \mathtt{R}^2}{\mathtt{R}} \right\rfloor q + 2^{\alpha} q}{\mathtt{R}} \right\rfloor$.
mul    t, $b$, $-q^{-1} \bmod^{\pm} \mathtt{R}^2$     ▷ t $= -bq^{-1} \bmod^{\pm} \mathtt{R}^2$.
smulwb t, t, a                            ▷ t $= \left\lfloor \frac{a_l \left(-bq^{-1} \bmod^{\pm} \mathtt{R}^2\right)}{\mathtt{R}} \right\rfloor$.
smlabb t, t, $q$, $2^{\alpha} q$                ▷ t $= \left\lfloor \frac{-a_l b q^{-1} \bmod^{\pm} \mathtt{R}^2}{\mathtt{R}} \right\rfloor q + 2^{\alpha} q$.
                                          ▷ The desired result is stored in the upper half.
```

## 7.3 Saber : Homomorphism Caching

### 7.3.1 Homomorphism Caching

Let $f : \mathcal{A} \to \mathcal{B}$ be an algebra monomorphism, and $\boldsymbol{a}_0, \boldsymbol{a}_1, \boldsymbol{b} \in \mathcal{A}$. Suppose we want to implement $\boldsymbol{a}_0\boldsymbol{b}$ and $\boldsymbol{a}_1\boldsymbol{b}$. We can compute with $f^{-1}(f(\boldsymbol{a}_0)f(\boldsymbol{b}))$ and $f^{-1}(f(\boldsymbol{a}_1)f(\boldsymbol{b}))$ using only three applications of $f$ and two applications of $f^{-1}$. This is called homomorphism caching and FFT-caching if $f$ is an FFT. [Ber08, Section 2.9] said that this was widely known in 1992. Appendix C will show historical evidence that caching was used implicitly in [Goo71] dating back to 1971.

### 7.3.2 Application to Saber

We review the application of homomorphism caching to Saber, a third round lattice-based KEM candidate in the NIST Post-Quantum Cryptography (PQC) Standardization. In Saber [DKRV20], the most performance-critical polynomial operation is multiplying $l \times l$ matrix by an $l \times 1$ vector over the polynomial ring $\mathbb{Z}_{8192}[x]/\left\langle x^{256} + 1\right\rangle$. We review the benefit of caching algebra and module homomorphisms.

**Algebra homomorphism caching.** Let $f : \mathbb{Z}_{8192}[x]/\left\langle x^{256} + 1\right\rangle \to S$ be an algebra homomorphism, $\cdot_S$ be the multiplication in $S$, and $+_S$ be the addition in $S$. We denote $\mathcal{C}(-)$ as the cost function of a map. If we apply $f$ to all the polynomials, compute matrix–vector multiplication over $S$, and transform back to a vector over $\mathbb{Z}_{8192}[x]/\left\langle x^{256} + 1\right\rangle$, the total cost is

$$(l^2 + l)\mathcal{C}(f) + l^2\mathcal{C}(\cdot_S) + (l^2 - l)\mathcal{C}(+_S) + l\mathcal{C}(f^{-1}).$$

Optimizations for the matrix–vector multiplication over $\mathbb{Z}_{8192}[x]/\left\langle x^{256} + 1\right\rangle$ should base the comparisons on the dominating term $\mathcal{C}(f) + \mathcal{C}(\cdot_S) + \mathcal{C}(+_S)$. [KRS19] chose $f$ as Toom–Cook but didn't exploit the homomorphic property. [MKV20] exploited the homomorphic property for Toom–Cook, and [CHK+21] chose $f$ as an FFT. The FFT-type approaches for Saber remain the fastest [CHK+21, ACC+22, BHK+22b].

**Module homomorphism caching.** In the previous paragraph, we have seen the importance of caching algebra homomorphisms. [BHK+22b] introduced "asymmetric multiplication" which falls into module homomorphism caching. For a polynomial $\boldsymbol{a} \in \mathbb{Z}_{8192}[x]/\left\langle x^{256}+1\right\rangle$ and an algebra homomorphism $f : \mathbb{Z}_{8192}[x]/\left\langle x^{256}+1\right\rangle \to S$, we first regard $f(\boldsymbol{a})$ as a module homomorphism mapping $f(\boldsymbol{b})$ to $f(\boldsymbol{a})f(\boldsymbol{b})$ for $\boldsymbol{b} \in \mathbb{Z}_{8192}[x]/\left\langle x^{256}+1\right\rangle$. If $f(\boldsymbol{a})$ amounts to polynomial multiplications modulo $x^v - \zeta$, we can turn $f(\boldsymbol{a})$ into a special kind of module homomorphism – Toeplitz matrix-vector product (cf. Section 5.5). In practice, the Toeplitz matrix conversion of $f(\boldsymbol{a})$ is cached. This is called **asymmetric multiplication** [BHK+22b, Section 4.2].

## 7.4 NTRU : Toeplitz Matrix-Vector Product

Section 5.5 discusses how to turn arbitrary algebra homomorphisms multiplying size-$n$ polynomials in $R[x]$ into a Toeplitz matrix-vector product for multiplying in $R[x]/\langle x^n - \alpha x - \beta\rangle$. In Section 6.4.2, we discuss the benefit of computing Toeplitz matrix-vector products with vector-by-scalar multiplications. We review the Toeplitz matrix-vector product approach for multiplying polynomials in $\mathbb{Z}_{2048}[x]/\left\langle x^{677}-1\right\rangle$ for the NTRU parameter set `ntruhps2048677` [CDH+20] in the third round NIST Post-Quantum Cryptography Standardization.

### 7.4.1 Armv7E-M Implementation

[IKPC22] applied Toeplitz matrix-vector product with Karatsuba and Toom–Cook. They first considered the following sequence of Karatsuba and Toom–Cook multiplying two size-720 polynomials:

$$\textbf{TC}\text{-}4 \to \textbf{K}\text{-}3 \to \textbf{K}\text{-}3 \to \textbf{K}\text{-}2$$

where **K**-2 is the usual Karatsuba in Section 3.7 and **K**-3 is the subtractive variant of 3-way Karatsuba[10]. They then took the dual of Toom–Cook, Karatsuba, and their inverses, and formed Toeplitz matrix-vector products as shown in Section 5.5. [IKPC22] identified that one no longer needs to reduce modulo a polynomial since it is merged with the polynomial multiplication itself (cf. Section 5.5.2).

### 7.4.2 Armv8-A Implementation

Shortly after, [CCHY24] explored the vectorization of Toeplitz matrix-vector products with Armv8-A. They started with the following sequence of Karatsuba and Toom–Cook multiplying two size-720 polynomials:

$$\textbf{TC}\text{-}5 \to \textbf{TC}\text{-}3 \to \textbf{TC}\text{-}3 \to \textbf{K}\text{-}2$$

and took the dual of all the homomorphisms. They showed that small-dimensional power-of-two Toeplitz matrix-vector product can be implemented efficiently for the following reasons: (i) one can construct Toeplitz matrices efficiently from its first row and column (cf. Section 6.4.2) and (ii) the existence of vector-by-scalar multiplication instructions implement the outer-product-based matrix-vector multiplication while avoiding permutations and reducing register pressure significantly [CCHY24]. See [CCHY24, Section 5.1] for more details on memory optimizations while inverting Karatsuba and Toom–Cook.

[10] In principle, we compute all possible $a_i b_i$ and $(a_i - a_j)(b_i - b_j)$ for $i \neq j$ so arbitrary $a_i b_j$ can be derived by only additions and subtractions, see [WP06, Section 3.2] for details.

## 7.5 NTRU Prime : Vectorized FFTs

In this section, we go through a detailed analysis of vectorized polynomial multipliers in NTRU Prime [BBC+20], a third round lattice-based KEM alternate candidate in the NIST Post-Quantum Cryptography Standardization. Our central objective is to answer the following question:

How FFT-, vectorization-, and permutation-friendly the coefficient ring is?

For simplicity, we focus on the polynomial ring $\mathbb{Z}_{4591}[x]/\left\langle x^{761}-x-1\right\rangle$ used in the parameter sets `ntrulpr761` and `sntrup761`. We first discuss a generic approach using Schönhage and Nussbaumer for maintaining the friendliness while exploiting no algebraic properties of the polynomial ring. Schönhage and Nussbaumer usually adjoin algebraic structures for friendliness with expenses. We then systematically analyze how to exploit the algebraic structure endowed in $\mathbb{Z}_{4591}$, showing that $\mathbb{Z}_{4591}$ actually admits FFT, vectorization, and permutation-friendly transformations. Observe that $4591-1=2\cdot 3^3\cdot 5\cdot 17$ and $4591^2-1=2^5\cdot 3^3\cdot 5\cdot 7\cdot 17\cdot 41$, we summarize the following findings from the works [HLY24, Hwa24].

- We qualify $\mathbb{Z}_{4591}$ as an FFT-friendly prime by considering the application of Good–Thomas and Rader's FFTs based on the factorization of $4591-1$ and Bruun's FFT based on the factorization of $4591^2-1$ [HLY24].
- We qualify $\mathbb{Z}_{4591}$ as a vectorization-friendly prime since the product $2^5\cdot 3^3\cdot 5\cdot 17=73440$ $(73440=16(4591-1)|(4591^2-1))$ allows a wide range of FFTs resulting small-dimensional power-of-two polynomial multiplications [HLY24].
- We qualify $\mathbb{Z}_{4591}$ as a permutation-friendly prime since 3, 5, and 17 are Fermat primes, and truncating Fermat-prime-size Rader's FFTs gives power-of-two transformations [Hwa24].

We review two AVX2-optimized implementations in this section: (i) [BBCT22]'s approach with truncated Schönhage and Nussbaumer FFTs, and (ii) [Hwa24]'s approach with truncated Rader, Good–Thomas, and Bruun FFTs.

### 7.5.1 A Generic Approach with Truncated Schönhage and Nussbaumer FFTs

Let's recall the AVX2-optimized polynomial multiplication for `ntrulpr761/sntrup761` from [BBCT22]. For multiplying polynomials in $\mathbb{Z}_{4591}[x]/\left\langle x^{761}-x-1\right\rangle$, [BBCT22] computed the product in $\mathbb{Z}_{4591}[x]/\left\langle (x^{512}-1)(x^{1024}+1)\right\rangle$ as follows. They first applied Schönhage replacing $x^{32}-y$ with $x^{64}+1$:

$$\frac{\mathbb{Z}_{4591}[x]}{\left\langle (x^{512}-1)(x^{1024}+1)\right\rangle} \overset{\eta_0}{\cong} \frac{\frac{\mathbb{Z}_{4591}[x]}{\left\langle x^{32}-y\right\rangle}[y]}{\left\langle (y^{16}-1)(y^{32}+1)\right\rangle} \overset{\eta_1}{\hookrightarrow} \frac{\frac{\mathbb{Z}_{4591}[x]}{\left\langle x^{64}+1\right\rangle}[y]}{\left\langle (y^{16}-1)(y^{32}+1)\right\rangle}.$$

Since $x^2$ is a principal 64-th root of unity in $\mathbb{Z}_{4591}[x]/\left\langle x^{64}+1\right\rangle$, we have $(y^{16}-1)(y^{32}+1)=\prod_{i\not\equiv 2 \pmod 4}(y-x^{2i})$ over $\mathbb{Z}_{4591}[x]/\left\langle x^{64}+1\right\rangle$. We find Schönhage's FFT vectorization-friendly since $64=4\cdot 16$. After splitting the polynomial ring in $y$, the remaining problem is multiplying in $\mathbb{Z}_{4591}[x]/\left\langle x^{64}+1\right\rangle$. [BBCT22] interleaved the polynomials with no leftovers and applied Nussbaumer as follows:

$$\frac{\mathbb{Z}_{4591}[x]}{\left\langle x^{64}+1\right\rangle} \overset{\eta_2}{\cong} \frac{\frac{\mathbb{Z}_{4591}[z]}{\left\langle z^{8}+1\right\rangle}[x]}{\left\langle x^{8}-z\right\rangle} \overset{\eta_3}{\hookrightarrow} \frac{\frac{\mathbb{Z}_{4591}[z]}{\left\langle z^{8}+1\right\rangle}[x]}{\left\langle x^{16}-1\right\rangle}.$$

Since $z$ is a principal 16-th root of unity in $\mathbb{Z}_{4591}[z]/\left\langle z^{8}+1\right\rangle$, we can factor $x^{16}-1$ into $\prod_j(x-z^j)$ over $\mathbb{Z}_{4591}[z]/\left\langle z^{8}+1\right\rangle$. In summary, we are left with $\frac{1536\cdot 2\cdot 2}{8}=768$ polynomial multiplications in $\mathbb{Z}_{4591}[z]/\left\langle z^{8}+1\right\rangle$. For multiplying polynomials in $\mathbb{Z}_{4591}[z]/\left\langle z^{8}+1\right\rangle$ for details.

### 7.5.2 A Specialized Approach with Truncated Rader, Good–Thomas, and Bruun FFTs

We briefly review the friendliness measures found by [HLY24, Hwa24]. The state-of-the-art AVX2 implementation [Hwa24] computed the products in $\mathbb{Z}_{4591}[x]/\left\langle\Phi_{17}(x^{96})\right\rangle$. [Hwa24] first applied truncated Rader's FFT to the isomorphism:

$$\frac{\mathbb{Z}_{4591}[x]}{\langle\Phi_{17}(x^{96})\rangle} \cong \prod_{i\neq 0} \frac{\mathbb{Z}_{4591}[x]}{\langle x^{96}-\omega_{17}^{i}\rangle}$$

and twisted each of $\mathbb{Z}_{4591}[x]/\left\langle x^{96}-\omega_{17}^{i}\right\rangle$ into $\mathbb{Z}_{4591}[x]/\left\langle x^{96}-1\right\rangle$. They then applied Good–Thomas FFT implementing the isomorphism:

$$\frac{\mathbb{Z}_{4591}[x]}{\langle x^{96}-1\rangle} \cong \prod_{j} \frac{\mathbb{Z}_{4591}[x]}{\left\langle x^{16}-\omega_{6}^{j}\right\rangle}$$

and twisted into $\mathbb{Z}_{4591}[x]/\left\langle x^{16}\pm 1\right\rangle$. Since each vector registers in AVX2 contains sixteen 16-bit values, all of the above are vectorization-friendly. The remaining problems are 48 polynomial multiplications in $\mathbb{Z}_{4591}[x]/\left\langle x^{16}-1\right\rangle$ and 48 in $\mathbb{Z}_{4591}[x]/\left\langle x^{16}+1\right\rangle$. Since 48 is a multiple of 16, we can interleave the polynomials with no leftovers. This implies permutation-friendliness. For multiplying polynomials in $\mathbb{Z}_{4591}[x]/\left\langle x^{16}\pm 1\right\rangle$, see [Hwa24] for details.

## 7.6 FrodoKEM

FrodoKEM [NAB+20] is a third round lattice-based KEM candidate in NIST Post-Quantum Cryptography Standardization. Different from other lattice-based cryptosystems, FrodoKEM computes a matrix-matrix product over $\mathbb{Z}_{2^k}$ for $k = 15, 16$. Since arithmetic modulo $2^k$ is supported by most of the computing devices, optimizing the number of arithmetic in $\mathbb{Z}_{2^k}$ directly translates into numerical improvements. Therefore, the rich literature on matrix-matrix products applies to optimizing FrodoKEM. See [BBC+23] for more information.

# 8 Overview of Advances

We give overviews of the advances of polynomial multiplications used in lattice-based cryptosystem implementations with emphases on modular arithmetic, algebraic techniques, and vectorization. Table 15 gives an overview of existing works for Dilithium, Kyber, and Saber, and Table 16 gives an overview of existing works for NTRU and NTRU Prime.

## 8.1 Modular Arithmetic

We first give an overview of modular arithmetic. See Table 17 for a summary of existing works on 8-bit AVR, Armv7-M, Armv7E-M, Armv8.0-A, MVE, and AVX2.

### 8.1.1 Vector architecture implementations

[Sei18] was the first work proposing signed Montgomery multiplication. They applied the idea to the vectorized 16-bit NTT used in the Ring-LWE scheme NewHope. Their idea nicely captured the availability of 16-bit multiplication instructions in AVX2, and it was applied to Kyber [ABD+20b] and NTRU Prime [BBC+20]. The subtractive variant was also implemented by [SKS+21] in Armv8-A. The "unsigned Barrett multiplication" was implemented in [Sho].

Table 15: Target architectures/extensions of existing works for Dilithium, Kyber, and Saber.

| | Dilithium | Kyber | Saber |
|---|---|---|---|
| [BKS19] | - | Armv7E-M | - |
| [KRS19] | - | - | Armv7E-M |
| [ABD+20a] | AVX2 | - | - |
| [ABD+20b] | - | AVX2 | - |
| [DKRV20] | - | - | AVX2 |
| [ABCG20] | - | - Armv7E-M | - |
| [MKV20] | - | - | Armv7E-M, AVX2 |
| [IKPC20] | - | - | Armv7E-M |
| [CHK+21] | - | - | Armv7-M, AVX2 |
| [GKS21] | Armv7-M | - | - |
| [SKS+21] | - | Armv8-A | - |
| [NG21] | - | Armv8-A | Armv8-A |
| [BHK+22b] | Armv8-A | Armv8-A | Armv8-A |
| [AHKS22] | Armv7-M | Armv7E-M | - |
| [HZZ+22] | - | Armv7E-M | - |
| [AMOT22] | - | - | RISC-V |
| [HKS23] | Armv7-M | - | - |

[BHK+22b] independently[11] found the signed Barrett multiplication, the correspondence between Montgomery and Barrett multiplication, and their variants and implemented them with Armv8-A. [BHK+22a] later demonstrated that if one increases the precision of the arithmetic, then we have the canonical representations of the products for some special moduli, and implemented the idea with M-profile vector extension (MVE).

### 8.1.2 Microcontroller Implementations

[BKS19] implemented Barrett reduction and the subtractive variant of Montgomery multiplication with the SIMD instruction `smul{b, t}{b, t}` in Armv7E-M. [ABCG20] switched to the accumulative variant of Montgomery multiplication and absorbed the addition by replacing a `smul{b, t}{b, t}` with `smla{b, t}{b, t}` [ABCG20, Algorithm 11]. The signed Barrett reduction was later improved by [ACC+21] with instructions `smlaw{b, t}`[12], but it was not reported (we found this while carefully examining the assembly programs). In [ACC+21], they also proposed the uses of `s{mul, mla}l` in Armv7-M for 32-bit Montgomery multiplication and `smmulr` in Armv7E-M for 32-bit Barrett reduction. The 32-bit Montgomery multiplication was independently proposed by [GKS21].

The improvement of signed Barrett reduction with Armv7E-M was later reported in [AHKS22]. [HZZ+22] and [AMOT22] independently found the signed versions of Plantard multiplication. [HZZ+22] applied the idea to 16-bit modular arithmetic with Armv7E-M instructions `s{mul, mla}w{b, t}` while [AMOT22] applied the idea to 32-bit modular arithmetic using $64 = 64 \times 64$ arithmetic on K210 (64-bit) [AMOT22, Section V-B]. Shortly after, [HZZ+24] applied signed Plantard arithmetic to Armv7-M with essentially the same idea from [AMOT22]. Recently, [HKS23] proposed the uses of Barrett

[11] [BHK+22b] cited the ePrint version of [SKS+21]. The subtractive variant of Montgomery multiplication was shown in the published version but not the ePrint one. We informed the authors of [BHK+22b] for this miscontribution.

[12] `smlaw{b, t}` multiplies a 32-bit value by a certain half of a 32-bit value, accumulates the result to the accumulator, and returns the most significant 32-bit value.

Table 16: Target architectures/extensions of existing works for NTRU and NTRU Prime.

| | NTRU | NTRU Prime |
|---|---|---|
| [KRS19] | Armv7E-M | - |
| [BBC$^+$20] | AVX2 | - |
| [CDH$^+$20] | - | AVX2 |
| [ACC$^+$21] | - | Armv7-M |
| [CHK$^+$21] | Armv7-M, AVX2 | - |
| [NG21] | Armv8-A | - |
| [IKPC22] | Armv7E-M | - |
| [AHY22] | Armv7-M | Armv7-M |
| [BBCT22] | - | AVX2 |
| [CCHY24] | Armv8-A | - |
| [HLY24] | - | Armv8-A |
| [Hwa24] | - | Armv8-A, AVX2 |

Table 17: Summary of existing works of modular multiplications relevant to our target architectures and extensions.

| | Barrett | Montgomery | Plantard |
|---|---|---|---|
| [Sho] | ✓ | - | - |
| [Sei18] | AVX2 | AVX2 | - |
| [BKS19] | Armv7E-M | Armv7E-M | - |
| [ABCG20] | - | Armv7E-M | - |
| [ACC$^+$21] | Armv7E-M | Armv7-M | - |
| [GKS21] | - | Armv7-M | - |
| [SKS$^+$21] | Armv8.0-A | Armv8.0-A | - |
| [BHK$^+$22b] | Armv8.0-A | Armv8.0-A | - |
| [AHKS22] | Armv7E-M | - | - |
| [BHK$^+$22a] | MVE | - | - |
| [HZZ$^+$22] | - | - | Armv7E-M |
| [AMOT22] | - | - | ✓ |
| [HKS23] | Armv7-M, 8-bit AVR | - | - |

multiplication when long/high multiplication instructions are slow, unusable, or unavailable and implemented the ideas with Armv7-M and 8-bit AVR.

## 8.2 Algebraic Techniques

In Dilithium and Kyber, most optimizations are about modular arithmetic, memory footprint, and instructions scheduling, so we exclude them unless specified otherwise in this section.

### 8.2.1 Vector Architecture Implementations

We first give an overview of AVX2-optimized implementations. For the big-by-small polynomial multiplication, [BBC$^+$20] implemented 16-bit Good–Thomas FFT with permutations instantiated as logical operations for `ntrulpr761/sntrup761` and applied radix-2 FFT to the power-of-two dimension. [CHK$^+$21] applied 16-bit size-256 negacyclic FFT to

Saber and size-1024, size-1536, and size-1728 cyclic FFTs to NTRU. For the big-by-big polynomial multiplication, [MKV20, CDH$^+$20] applied Toom–Cook and Karatsuba to NTRU and Saber. For NTRU Prime, [BBCT22] implemented truncated Schönhage's and Nussbaumer's FFTs (cf. Section 7.5.2), and [Hwa24] applied truncated Rader's, Good–Thomas, and Bruun's FFTs following [HLY24]'s Armv8-A work.

For the Armv8-A Neon implementations, [NG21] implemented 16-bit size-256 negacyclic FFT for Saber, and 3- and 4-way Toom–Cook for NTRU. Shortly after, [BHK$^+$22b] demonstrated 32-bit negacyclic FFT is more performant for Saber [13]. [CCHY24] deployed 5-way Toom–Cook to NTRU and showed that Toeplitz transformation with Toom–Cook was more favorable due to the presence of vector-by-scalar multiplication instructions on Armv8-A, and [HLY24] applied Rader's, Good–Thomas, and Bruun's FFTs. Finally, [Hwa24] applied truncated Rader's FFT, Good–Thomas FFT, and Toeplitz matrix-vector products to small-dimensional cyclic/negacyclic convolutions.

### 8.2.2 Microcontroller Implementations

[KRS19] applied Toom–Cook and Karatsuba to NTRU and Saber. [MKV20] later cached the homomorphisms in the case of Saber and [IKPC20] applied the Toeplitz matrix-vector product to Saber with Toom–Cook and Karatsuba as the underlying homomorphisms. [ACC$^+$21] proposed three implementations for NTRU Prime parameter sets `ntrulpr761`/`sntrup761`: (i) a Good–Thomas FFT computing the big-by-small polynomial multiplication with 32-bit arithmetic over $\mathbb{Z}$, (ii) an FFT using radix-2, radix-3, and radix-5 butterflies with 16-bit arithmetic over $\mathbb{Z}_{4591}$, and (iii) an FFT using radix-3 and Rader's radix-17 butterflies with 16-bit arithmetic over $\mathbb{Z}_{4591}$. The big-by-small polynomial multiplication came from [BBC$^+$20] and was later adapted to NTRU and Saber [CHK$^+$21]. [IKPC22] extended [IKPC20]'s work to NTRU and [AHY22] improved [ACC$^+$21, CHK$^+$21]'s NTRU and NTRU Prime implementations by proposing vector-radix butterflies for speed [AHY22, Section 4.1] and vectorization-friendly Good–Thomas for code size [AHY22, Section 3.3].

## 9 Directions for Future Works

We point out several possible future works as follows.

**Non-uniform property of localization in Toom–Cook.** In Section 4.1, we explain that localization does not need to be uniform among subproblems and illustrate the idea with Toom–Cook. In practice, one usually applies Toom–Cook recursively. Since the required localization for subproblems is not uniform, applying more aggressive divide-and-conquer strategies for some subproblems is possible. **We would like to know the practical impact of this observation of Toom–Cook and its Toeplitz version for NTRU and Saber.**

**Schönhage and Nussbaumer for NTRU and Saber.** In lattice-based cryptosystem implementations, Schönhage and Nussbaumer FFTs were only applied to NTRU Prime where the coefficient ring is $\mathbb{Z}_q$ for an odd $q$. **We would like to know the practical impact of Schönhage's FFT, Nussbaumer's FFT, and their Toeplitz versions for NTRU and Saber where $q$ is a power of two.**

[13] This doesn't say that we should do the same thing for AVX2-optimized implementation since there are no native 32-bit multiplication instructions in AVX2.

**Barrett multiplication for finite fields.** The finite field versions of Montgomery multiplication [KA98] and Barrett reduction [Dhe03] were known in the literature. Appendix A extends the correspondence between Montgomery multiplication and Barrett multiplication [BHK$^+$22b] to principal ideal domains. For a finite field $\mathbb{F}_p$, since $\mathbb{F}_p[x]$ is a principal ideal domain, the correspondence implies the finite field version of Barrett multiplication. The Barrett reduction found by [Dhe03] is then a special case in this regard. **We would like to know the practical impact of Barrett multiplication for finite fields.**

**Toeplitz matrix-vector product for NTRU Prime.** Section 5.5.2 explains that polynomial multiplication modulo $x^n - \alpha x - \beta$ can be turned into a Toeplitz matrix-vector product. **We would like to know the performance of the Toeplitz approach for NTRU Prime.**

**Falcon.** Falcon [PFH$^+$20] is a hash-and-sign digital signature scheme that is recently selected by NIST for standardization.

1. In the key generation, we are given a polynomial modulus $\Phi = \Phi_{\{512,1024\}}(x)$, two polynomials $f, g \in \mathbb{Z}[x]/\langle\Phi\rangle$, and an integer $q = 12289$, and are asked to solve for polynomials $F$ and $G$ satisfying

   $$gF - fG = q \mod \Phi.$$

   We briefly review the `TowerSolver` by [PP19] as follows. Since $\Phi$ is a power-of-two cyclotomic polynomial, we have the tower of rings

   $$\mathbb{Z} \cong \mathbb{Z}[x]/\langle\Phi_2(x)\rangle \subset \mathbb{Z}[x]/\langle\Phi_4(x)\rangle \subset \cdots \subset \mathbb{Z}[x]/\langle\Phi\rangle$$

   and the tower of fields

   $$\mathbb{Q} \cong \mathbb{Q}[x]/\langle\Phi_2(x)\rangle \subset \mathbb{Q}[x]/\langle\Phi_4(x)\rangle \subset \cdots \subset \mathbb{Q}[x]/\langle\Phi\rangle .$$

   For simplicity, we denote $\mathcal{O}_k = \mathbb{Z}[x]/\langle\Phi_{2^k}(x)\rangle$ and $\mathcal{K}_k = \mathbb{Q}[x]/\langle\Phi_{2^k}(x)\rangle$. Suppose we start with the ring $\mathcal{O}_k$ for a positive integer $k \geq 2$. `TowerSolver` recursively solves for a solution in $\mathcal{O}_{k-1}$, lifts the solution back to $\mathcal{O}_k$ with the norm associated to the field extension $\mathcal{K}_k \backslash \mathcal{K}_{k-1}$, and reduces the coefficients with Babai's reduction and FFT in $\mathbb{C}[x]/\langle\Phi_{2^k}(x)\rangle$.

   (a) During each recursion, the bit size of the coefficient is essentially doubled and the polynomial ring dimension over $\mathbb{Z}$ is halved. Since the bit size quickly exceeds the word size on modern computing devices, the coefficient ring operations can only be implemented with multi-precision arithmetic that is not reviewed in this paper. The state-of-the-art implemenation applied CRT over several 31-bit NTT-friendly prime moduli. **We would like to know if there are better choices for the CRT moduli.**

   (b) As for Babai's reduction and FFT in $\mathbb{C}[x]/\langle\Phi_{2^k}(x)\rangle$, the state-of-the-art implementation [Por23] applied fixed-point arithmetic with 64-bit precision. The author provided a portable C implementation and an AVX2-optimized implementation. **We would like to know the performance of fixed-point FFT on more platforms.**

2. In the signature generation, we also need FFT over $\mathbb{C}[x]/\langle\Phi_{2^k}(x)\rangle$ for the fast Fourier orthogonalization sampler [DP16]. But we need a much higher precision for sampling compared to key generation due to the security of the underlying

hardness assumption. The state-of-the-art implementation [Por19] used floating-point arithmetic and emulated the computation when there is no floating-point units. **We would like to know if one can improve the emulated floating-point arithmetic.**

3. In the signature verification, we only need polynomial arithmetic in $\mathbb{Z}_{12289}[x]/\langle\Phi\rangle$, whose optimizing strategies basically follow from literatures reviewed in Section 7.2.

## References

[AB74] Ramesh C. Agarwal and Charles S. Burrus. Fast convolution using Fermat number transforms with applications to digital filtering. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 22(2):87–97, 1974. URL: https://ieeexplore.ieee.org/abstract/document/1162555.

[ABCG20] Erdem Alkim, Yusuf Alper Bilgin, Murat Cenk, and François Gérard. Cortex-M4 optimizations for {R, M} LWE schemes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):336–357, 2020. URL: https://tches.iacr.org/index.php/TCHES/article/view/8593.

[ABD$^+$20a] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS–Dilithium. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2020. URL: https://pq-crystals.org/dilithium/.

[ABD$^+$20b] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS–Kyber. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2020. URL: https://pq-crystals.org/kyber/.

[ACC$^+$21] Erdem Alkim, Dean Yun-Li Cheng, Chi-Ming Marvin Chung, Hülya Evkan, Leo Wei-Lun Huang, Vincent Hwang, Ching-Lin Trista Li, Ruben Niederhagen, Cheng-Jhih Shih, Julian Wälde, and Bo-Yin Yang. Polynomial Multiplication in NTRU Prime Comparison of Optimization Strategies on Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):217–238, 2021. URL: https://tches.iacr.org/index.php/TCHES/article/view/8733.

[ACC$^+$22] Amin Abdulrahman, Jiun-Peng Chen, Yu-Jia Chen, Vincent Hwang, Matthias J. Kannwischer, and Bo-Yin Yang. Multi-moduli NTTs for Saber on Cortex-M3 and Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):127–151, 2022. URL: https://tches.iacr.org/index.php/TCHES/article/view/9292.

[AHKS22] Amin Abdulrahman, Vincent Hwang, Matthias J. Kannwischer, and Dann Sprenkels. Faster Kyber and Dilithium on the Cortex-M4. In *Applied Cryptography and Network Security: 20th International Conference, ACNS 2022, Rome, Italy, June 20–23, 2022, Proceedings*, pages 853–871, 2022. URL: https://link.springer.com/chapter/10.1007/978-3-031-09234-3_42.

[AHY22] Erdem Alkim, Vincent Hwang, and Bo-Yin Yang. Multi-Parameter Support with NTTs for NTRU and NTRU Prime on Cortex-M4. *IACR Transactions* *on Cryptographic Hardware and Embedded Systems*, 2022(4):349–371, 2022. URL: https://tches.iacr.org/index.php/TCHES/article/view/9823.

[AMOT22] Daichi Aoki, Kazuhiko Minematsu, Toshihiko Okamura, and Tsuyoshi Takagi. Efficient Word Size Modular Multiplication over Signed Integers. In *2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)*, pages 94–101. IEEE, 2022. URL: https://ieeexplore.ieee.org/document/9974215.

[ARM10] ARM. *Cortex-M3 Technical Reference Manual*, 2010. URL: https://developer.arm.com/documentation/ddi0337/h.

[ARM12] ARM. *ARM Architecture Reference Manual*, 2012. URL: https://developer.arm.com/documentation/ddi0406/cb.

[ARM21a] ARM. *Arm Architecture Reference Manual, Armv8, for Armv8-A architecture profile*, 2021. URL: https://developer.arm.com/documentation/ddi0487/gb/?lang=en.

[ARM21b] ARM. *Armv7-M Architecture Refernce Manual*, 2021. URL: https://developer.arm.com/documentation/ddi0403/ed.

[ARM23] ARM. *Armv8-M Architecture Reference Manual*, 2023. URL: https://developer.arm.com/documentation/ddi0553/latest.

[Bar86] Paul Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *CRYPTO 1986*, LNCS, pages 311–323. SV, 1986. URL: https://link.springer.com/chapter/10.1007/3-540-47721-7_24.

[BBC$^+$20] Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, Chitchanok Chuengsatiansup, Tanja Lange, Adrian Marotzke, Bo-Yuan Peng, Nicola Tuveri, Christine van Vredendaal, and Bo-Yin Yang. NTRU Prime. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2020. URL: https://ntruprime.cr.yp.to/.

[BBC$^+$23] Joppe W. Bos, Olivier Bronchain, Frank Custers, Joost Renes, Denise Verbakel, and Christine van Vredendaal. Enabling FrodoKEM on Embedded Devices. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):74–96, 2023. URL: https://tches.iacr.org/index.php/TCHES/article/view/10957.

[BBCT22] Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, and Nicola Tuveri. OpenSSLNTRU: Faster post-quantum TLS key exchange. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 845–862, 2022. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/bernstein.

[BC87] J. V. Brawley and L. Carlitz. Irreducibles and the composed product for polynomials over a finite field. *Discrete Mathematics*, 65(2):115–139, 1987. URL: https://www.sciencedirect.com/science/article/pii/0012365X8790135X.

[BCS13] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. Mcbits: Fast constant-time code-based cryptography. In *Cryptographic Hardware and Embedded Systems-CHES 2013: 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings 15*, pages 250–272. Springer,

2013. URL: https://link.springer.com/chapter/10.1007/978-3-642-40349-1_15#:~:text=Abstract,a%20single%20Ivy%20Bridge%20core.

[Ber01] Daniel J. Bernstein. Multidigit multiplication for mathematicians, 2001. URL: https://cr.yp.to/papers.html#m3.

[Ber05] Daniel J. Bernstein. Cache-timing attacks on aes, 2005. URL: https://cr.yp.to/antiforgery/cachetiming-20041111.pdf.

[Ber07] Daniel J. Bernstein. The tangent FFT. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes: 17th International Symposium, AAECC-17*, pages 291–300, 2007. URL: https://link.springer.com/chapter/10.1007/978-3-540-77224-8_34.

[Ber08] Daniel J. Bernstein. Fast multiplication and its applications. *Algorithmic number theory*, 44:325–384, 2008. URL: https://cr.yp.to/papers.html#multapps.

[Ber23] Daniel J. Bernstein. Fast norm computation in smooth-degree abelian number fields. *Research in Number Theory*, 9(4):82, 2023. URL: https://link.springer.com/article/10.1007/s40993-022-00402-0.

[BGM93] Ian F. Blake, Shuhong Gao, and Ronald C. Mullin. Explicit Factorization of $x^{2^k}+1$ over $\mathbb{F}_p$ with Prime $p \equiv 3 \bmod 4$. *Applicable Algebra in Engineering, Communication and Computing*, 4(2):89–94, 1993. URL: https://link.springer.com/article/10.1007/BF01386832.

[BHK$^+$22a] Hanno Becker, Vincent Hwang, Matthias J. Kannwischer, Lorenz Panny, and Bo-Yin Yang. Efficient Multiplication of Somewhat Small Integers using Number–Theoretic Transforms. In *International Workshop on Security*, pages 3–23. Springer, 2022. URL: https://link.springer.com/chapter/10.1007/978-3-031-15255-9_1.

[BHK$^+$22b] Hanno Becker, Vincent Hwang, Matthias J. Kannwischer, Bo-Yin Yang, and Shang-Yi Yang. Neon NTT: Faster Dilithium, Kyber, and Saber on Cortex-A72 and Apple M1. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):221–244, 2022. URL: https://tches.iacr.org/index.php/TCHES/article/view/9295.

[BKS19] Leon Botros, Matthias J. Kannwischer, and Peter Schwabe. Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4. In *Progress in Cryptology - AFRICACRYPT 2019*, volume 11627 of *Lecture Notes in Computer Science*, pages 209–228. Springer, 2019. URL: https://doi.org/10.1007/978-3-030-23696-0_11.

[BMGVdO15] F.E. Brochero Martínez, C. R. Giraldo Vergaraand, and L. Batista de Oliveira. Explicit factorization of $x^n - 1 \in \mathbb{F}_q[x]$. *Designs, Codes and Cryptography*, 77:277–286, 2015. URL: https://link.springer.com/article/10.1007/s10623-014-0005-y.

[BMK$^+$22] Hanno Becker, Jose Maria Bermudo Mera, Angshuman Karmakar, Joseph Yiu, and Ingrid Verbauwhedeg. Polynomial multiplication on embedded vector architectures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):482–505, 2022. URL: https://tches.iacr.org/index.php/TCHES/article/view/9305.

[Bou89] Nicolas Bourbaki. *Algebra I*. Springer, 1989.

[Bra84] Ronald N. Bracewell. The Fast Hartley Transform. *Proceedings of the IEEE*, 72(8):1010–1018, 1984. URL: https://ieeexplore.ieee.org/document/1457236.

[Bru78] Georg Bruun. z-transform DFT Filters and FFT's. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):56–63, 1978. URL: https://ieeexplore.ieee.org/document/1163036.

[CCHY24] Han-Ting Chen, Yi-Hua Chung, Vincent Hwang, and Bo-Yin Yang. Algorithmic Views of Vectorized Polynomial Multipliers – NTRU. In Anupam Chattopadhyay, Shivam Bhasin, Stjepan Picek, and Chester Rebeiro, editors, *Progress in Cryptology – INDOCRYPT 2023*, pages 177–196. Springer Nature Switzerland, 2024. URL: https://link.springer.com/chapter/10.1007/978-3-031-56235-8_9.

[CDH$^+$20] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, Tsunekazu Saito, Takashi Yamakawa, and Keita Xagawa. NTRU. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2020. URL: https://ntru.org/.

[CF94] Richard Crandall and Barry Fagin. Discrete Weighted Transforms and Large-integer Arithmetic. *Mathematics of computation*, 62(205):305–324, 1994. URL: https://www.ams.org/journals/mcom/1994-62-205/S0025-5718-1994-1185244-1/?active=current.

[CHK$^+$21] Chi-Ming Marvin Chung, Vincent Hwang, Matthias J. Kannwischer, Gregor Seiler, Cheng-Jhih Shih, and Bo-Yin Yang. NTT Multiplication for NTT-unfriendly Rings New Speed Records for Saber and NTRU on Cortex-M4 and AVX2. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):159–188, 2021. URL: https://tches.iacr.org/index.php/TCHES/article/view/8791.

[CK91] David G. Cantor and Erich Kaltofen. On Fast Multiplication of Polynomials over Arbitrary Algebras. *Acta Informatica*, 28(7):693–701, 1991. URL: https://link.springer.com/article/10.1007/BF01178683.

[CT65] 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. URL: https://www.ams.org/journals/mcom/1965-19-090/S0025-5718-1965-0178586-1/.

[DH84] Pierre Duhamel and Henk Hollmann. 'Split Radix' FFT Algorithm. *Electronics letters*, 20(1):14–16, 1984. URL: https://digital-library.theiet.org/content/journals/10.1049/el_19840012.

[Dhe03] Jean-François Dhem. Efficient Modular Reduction Algorithm in $\mathbb{F}_q[x]$ and Its Application to "Left to Right" Modular Multiplication in $\mathbb{F}_2[x]$. In *Cryptographic Hardware and Embedded Systems-CHES 2003: 5th International Workshop, Cologne, Germany, September 8–10, 2003. Proceedings 5*, pages 203–213. Springer, 2003. URL: https://link.springer.com/chapter/10.1007/978-3-540-45238-6_17.

[DKRV20] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. SABER. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2020. URL: https://www.esat.kuleuven.be/cosic/pqcrypto/saber/.

[DP16] Léo Ducas and Thomas Prest. Fast Fourier Orthogonalization. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, pages 191–198, 2016. URL: https://link.springer.com/chapter/10.1007/978-3-031-15777-6_7.

[DV78a] Eric Dubois and Anastasios N. Venetsanopoulos. A New Algorithm for the Radix-3 FFT. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(3):222–225, 1978. URL: https://ieeexplore.ieee.org/document/1163084.

[DV78b] Eric Dubois and Anastasios N. Venetsanopoulos. The discrete Fourier transform over finite rings with application to fast convolution. *IEEE Computer Architecture Letters*, 27(07):586–593, 1978. URL: https://ieeexplore.ieee.org/document/1675158.

[DV90] Pierre Duhamel and Martin Vetterli. Fast Fourier transforms: a tutorial review and a state of the art. *Signal processing*, 19(4):259–299, 1990. URL: https://www.sciencedirect.com/science/article/pii/016516849090158U.

[FD05] Haining Fan and Yiqi Dai. Fast Bit-Parallel $GF(2^n)$ Multiplier for All Trinomials. *IEEE Transactions on Computers*, 54(4):485–490, 2005. URL: https://ieeexplore.ieee.org/document/1401867.

[FH07] 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. URL: https://ieeexplore.ieee.org/document/4042682.

[Fid73] Charles M. Fiduccia. *On the Algebraic Complexity of Matrix Multiplication*. PhD thesis, Brown University, 1973. URL: https://cr.yp.to/bib/entries.html#1973/fiduccia-matrix.

[Flo72] Robert W Floyd. Permuting Information in Idealized Two-Level Storage. In *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, pages 105–109. Springer, 1972. URL: https://link.springer.com/chapter/10.1007/978-1-4684-2001-2_10.

[FSS20] Tim Fritzmann, Georg Sigl, and Johanna Sepúlveda. RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):239–280, 2020. URL: https://tches.iacr.org/index.php/TCHES/article/view/8683.

[Für09] Martin Fürer. Faster Integer Multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009. URL: https://doi.org/10.1137/070711761.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: a Foundation for Computer Science*. Addison-Wesley, second edition, 1994.

[GKS21] Denisa O. C. Greconici, Matthias J. Kannwischer, and Daan Sprenkels. Compact Dilithium Implementations on Cortex-M3 and Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):1–24, 2021. URL: https://tches.iacr.org/index.php/TCHES/article/view/8725.

[Goo58] I. J. Good. The Interaction Algorithm and Practical Fourier Analysis. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):361–372, 1958. URL: https://www.jstor.org/stable/2983896.

[Goo71] I. J. Good. The relationship between two fast Fourier transforms. *IEEE Transactions on Computers*, 100(3):310–317, 1971. URL: https://ieeexplore.ieee.org/document/1671829.

[GOPT10] Johann Großschädl, Elisabeth Oswald, Dan Page, and Michael Tunstall. Side-Channel Analysis of Cryptographic Software via Early-Terminating Multiplications. In *Information, Security and Cryptology–ICISC 2009: 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers 12*, pages 176–192. Springer, 2010. URL: https://link.springer.com/chapter/10.1007/978-3-642-14423-3_13.

[GS66] W. M. Gentleman and G. Sande. Fast Fourier Transforms: For Fun and Profit. In *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference*, AFIPS '66 (Fall), pages 563–578. Association for Computing Machinery, 1966. URL: https://doi.org/10.1145/1464291.1464352.

[Har42] Ralph VL Hartley. A More Symmetrical Fourier Analysis Applied to Transmission Problems. *Proceedings of the IRE*, 30(3):144–150, 1942. URL: https://ieeexplore.ieee.org/document/1694454.

[Has22] Chenar Abdulla Hassan. Radix-3 NTT-Based Polynomial Multiplication for Lattice-Based Cryptography. Master's thesis, Middle East Technical University, 2022. URL: https://open.metu.edu.tr/handle/11511/97928.

[HB95] M.A. Hasan and V.K. Bhargava. Architecture for a low complexity rate-adaptive Reed-Solomon encoder. *IEEE Transactions on Computers*, 44(7):938–942, 1995. URL: https://ieeexplore.ieee.org/document/392853.

[HKS23] Vincent Hwang, YoungBeom Kim, and Seog Chung Seo. Barrett Multiplication for Dilithium on Embedded Devices. Cryptology ePrint Archive, Paper 2023/1955, 2023. URL: https://eprint.iacr.org/2023/1955.

[HLY24] Vincent Hwang, Chi-Ting Liu, and Bo-Yin Yang. Algorithmic Views of Vectorized Polynomial Multipliers – NTRU Prime. In *International Conference on Applied Cryptography and Network Security*, pages 24–46. Springer, 2024. URL: https://link.springer.com/chapter/10.1007/978-3-031-54773-7_2.

[HMCS77] David B. Harris, James H. McClellan, David S. K. Chan, and Hans W. Schuessler. Vector Radix Fast Fourier Transform. In *ICASSP'77. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 548–551, 1977. URL: https://ieeexplore.ieee.org/document/1170349.

[HQZ04] Guillaume Hanrot, Michel Quercia, and Paul Zimmermann. The Middle Product Algorithm I. *Applicable Algebra in Engineering, Communication and Computing*, 14(6):415–438, 2004. URL: https://link.springer.com/article/10.1007/s00200-003-0144-2.

[HVDH22] David Harvey and Joris Van Der Hoeven. Polynomial Multiplication over Finite Fields in time $O(nlogn)$. *Journal of the ACM*, 69(2):1–40, 2022. URL: https://dl.acm.org/doi/full/10.1145/3505584.

[Hwa22] Vincent Hwang. Case Studies on Implementing Number–Theoretic Transforms with Armv7-M, Armv7E-M, and Armv8-A. Master's thesis, National Taiwan University, 2022. URL: https://github.com/vincentvbh/NTTs_with_Armv7-M_Armv7E-M_Armv8-A.

[Hwa24] Vincent Hwang. Pushing the Limit of Vectorized Polynomial Multiplication for NTRU Prime, 2024. To appear at ACISP 2024, currently available at https://eprint.iacr.org/2023/604.

[HZZ$^+$22] Junhao Huang, Jipeng Zhang, Haosong Zhao, Zhe Liu, Ray CC Cheung, Çetin Kaya Koç, and Donglong Chen. Improved Plantard Arithmetic for Lattice-based Cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):614–636, 2022. URL: https://tches.iacr.org/index.php/TCHES/article/view/9833.

[HZZ$^+$24] Junhao Huang, Haosong Zhao, Jipeng Zhang, Wangchen Dai, Lu Zhou, Ray CC Cheung, Cetin Kaya Koc, and Donglong Chen. Yet another Improvement of Plantard Arithmetic for Faster Kyber on Low-end 32-bit IoT Devices. *IEEE Transactions on Information Forensics and Security*, 2024. URL: https://ieeexplore.ieee.org/document/10453274.

[IKPC20] Írem Keskinkurt Paksoy and Murat Cenk. TMVP-based Multiplication for Polynomial Quotient Rings and Application to Saber on ARM Cortex-M4. Cryptology ePrint Archive, Paper 2020/1302, 2020. URL: https://eprint.iacr.org/2020/1302.

[IKPC22] Írem Keskinkurt Paksoy and Murat Cenk. Faster NTRU on ARM Cortex-M4 with TMVP-based multiplication. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 69(10):4083–4092, 2022. URL: https://ieeexplore.ieee.org/document/9835023.

[Int23] Intel. *Intel Architecture Instruction Set Extensions Programming Reference*, 2023. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-avx-512-instructions.html.

[Jac12a] Nathan Jacobson. *Basic Algebra I.* Courier Corporation, 2012.

[Jac12b] Nathan Jacobson. *Basic Algebra II.* Courier Corporation, 2012.

[JF07] Steven G. Johnson and Matteo Frigo. A Modified Split-Radix FFT With Fewer Arithmetic Operations. *IEEE Transactions on Signal Processing*, 55(1):111–119, 2007. URL: https://ieeexplore.ieee.org/document/4034175.

[KA98] Cetin Kaya Koc and Tolga Acar. Montgomery Multiplication in GF($2^k$). *Designs, Codes and Cryptography*, 14:57–69, 1998. URL: https://link.springer.com/article/10.1023/A:1008208521515.

[KAK96] Cetin Kaya Koc, Tolga Acar, and Burton S. Kaliski. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996. URL: https://ieeexplore.ieee.org/document/502403.

[KO62] A. Karatsuba and Yu. Ofman. Multiplication of many-digital numbers by automatic computers. In *Doklady Akademii Nauk*, volume 145(2), pages 293–294, 1962. URL: http://cr.yp.to/bib/1963/karatsuba.html.

[KRS19] 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. URL: https://link.springer.com/chapter/10.1007/978-3-030-21568-2_14.

[LS19] Vadim Lyubashevsky and Gregor Seiler. NTTRU: Truly Fast NTRU Using NTT. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):180–201, 2019. URL: https://tches.iacr.org/index.php/TCHES/article/view/8293.

[LVB07] T. Lundy and James Van Buskirk. A new matrix approach to real FFTs and convolutions of length $2^k$. *Computing*, 80:23–45, 2007. URL: https://link.springer.com/article/10.1007/s00607-007-0222-6.

[LZ22] Zhichuang Liang and Yunlei Zhao. Number Theoretic Transform and Its Applications in Lattice-based Cryptosystems: A Survey. *arXiv preprint arXiv:2211.13546*, 2022. URL: https://arxiv.org/abs/2211.13546.

[Mey96] Helmut Meyn. Factorization of the Cyclotomic Polynomial $x^{2^n} + 1$ over Finite Fields. *Finite Fields and Their Applications*, 2(4):439–442, 1996. URL: https://www.sciencedirect.com/science/article/pii/S107157979690026X.

[MKV20] 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*, 2020(2):222–244, 2020. URL: https://tches.iacr.org/index.php/TCHES/article/view/8550.

[Mon85] Peter L. Montgomery. Modular Multiplication Without Trial Division. *Mathematics of computation*, 44(170):519–521, 1985. URL: https://www.ams.org/journals/mcom/1985-44-170/S0025-5718-1985-0777282-X/?active=current.

[Mur96] Hideo Murakami. Real-valued fast discrete Fourier transform and cyclic convolution algorithms of highly composite even length. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, volume 3, pages 1311–1314, 1996. URL: https://ieeexplore.ieee.org/document/543667.

[MV83a] Jean-Bernard Martens and Marc C. Vanwormhoudt. Convolution Using a Conjugate Symmetry Property for Number Theoretic Transforms Over Rings of Regular Integers. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 31(5):1121–1125, 1983. URL: https://ieeexplore.ieee.org/document/1164198.

[MV83b] Jean-Bernard Martens and Marc C. Vanwormhoudt. Convolutions of Long Integer Sequences by Means of Number Theoretic Transforms Over Residue Class Polynomial Rings. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 31(5):1125–1134, 1983. URL: https://ieeexplore.ieee.org/abstract/document/1164201.

[NAB$^+$20] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebil. FrodoKEM. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2020. URL: https://frodokem.org/.

[NG21] Duc Tri Nguyen and Kris Gaj. Fast NEON-Based Multiplication for Lattice-Based NIST Post-quantum Cryptography Finalists. In *Post-Quantum Cryptography: 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20–22, 2021, Proceedings*, pages 234–254, 2021. URL: https://link.springer.com/chapter/10.1007/978-3-030-81293-5_13.

[Nic71] Peter J. Nicholson. Algebraic Theory of Finite Fourier Transforms. *Journal of Computer and System Sciences*, 5(5):524–547, 1971. URL: https://www.sciencedirect.com/science/article/pii/S0022000071800144.

[NIS] NIST, the US National Institute of Standards and Technology. Post-Quantum Cryptography Standardization Project. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography.

[Nus80] Henri J. Nussbaumer. Fast Polynomial Transform Algorithms for Digital Convolution. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(2):205–215, 1980. URL: https://ieeexplore.ieee.org/document/1163372.

[Nus82] Henri J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms*. Springer Berlin, Heidelberg, 2nd edition, 1982. URL: https://doi.org/10.1007/978-3-642-81897-4.

[Ora14] Oracle. *x86 Assembly Language Reference Manual*, 2014. URL: https://docs.oracle.com/cd/E36784_01/html/E36859/enmzx.html#scrolltoc.

[PAA$^+$20] Thomas Pöppelmann, Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Peter Schwabe, Douglas Stebila, Martin R. Albrecht, Emmanuela Orsini, Valery Osheter, Kenneth G. Paterson, Guy Peer, and Nigel P. Smart. NewHope. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2020. URL: https://newhopecrypto.org/.

[PFH$^+$20] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2020. URL: https://falcon-sign.info/.

[Pla21] Thomas Plantard. Efficient word size modular arithmetic. *IEEE Transactions on Emerging Topics in Computing*, 9(3):1506–1518, 2021. URL: https://ieeexplore.ieee.org/document/9416314.

[Pol71] John M. Pollard. The Fast Fourier Transform in a Finite Field. *Mathematics of computation*, 25(114):365–374, 1971. URL: https://www.ams.org/journals/mcom/1971-25-114/S0025-5718-1971-0301966-0/?active=current.

[Por19] Thomas Pornin. New Efficient, Constant-Time Implementations of Falcon, 2019. URL: https://eprint.iacr.org/2019/893.

[Por23] Thomas Pornin. Improved Key Pair Generation for Falcon, BAT and Hawk, 2023. URL: https://eprint.iacr.org/2023/290.

[PP19] Thomas Pornin and Thomas Prest. More Efficient Algorithms for the NTRU Key Generation Using the Field Norm. In *IACR International Workshop on Public Key Cryptography*, pages 504–533. Springer, 2019. URL: https://link.springer.com/chapter/10.1007/978-3-030-17259-6_17.

[Rad68] Charles M. Rader. Discrete Fourier Transforms When the Number of Data Samples Is Prime. *Proceedings of the IEEE*, 56(6):1107–1108, 1968. URL: https://ieeexplore.ieee.org/document/1448407.

[Sch77] Arnold Schönhage. Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Informatica*, 7(4):395–398, 1977. URL: https://link.springer.com/article/10.1007/bf00289470.

[Sei18] Gregor Seiler. Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography. Cryptology ePrint Archive, Report 2018/039, 2018. URL: https://eprint.iacr.org/2018/039.

[Sho] Victor Shoup. NTL: a Library for Doing Number Theory. URL: http://www.shoup.net/ntl/.

[Sho99] Victor Shoup. Efficient Computation of Minimal Polynomials in Algebraic Extensions of Finite Fields. In *Proceedings of the 1999 international symposium on Symbolic and algebraic computation*, pages 53–58, 1999. URL: https://dl.acm.org/doi/10.1145/309831.309859.

[SKS+21] Pakize Sanal, Emrah Karagoz, Hwajeong Seo, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. Kyber on ARM64: Compact Implementations of Kyber on 64-bit ARM Cortex-A Processors. In *Security and Privacy in Communication Networks: 17th EAI International Conference, SecureComm 2021, Virtual Event, September 6–9, 2021, Proceedings, Part II*, pages 424–440. Springer, 2021. URL: https://link.springer.com/chapter/10.1007/978-3-030-90022-9_23.

[SS71] Arnold Schönhage and Volker Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971. URL: https://link.springer.com/article/10.1007/BF02242355.

[Sto66] Thomas G. Stockham, Jr. High-Speed Convolution and Correlation. In *Proceedings of the April 26-28, 1966, Spring joint computer conference*, pages 229–233, 1966. URL: https://dl.acm.org/doi/10.1145/1464182.1464209.

[Tho63] Llewellyn Hilleth Thomas. Using a computer to solve problems in physics. *Applications of digital computers*, pages 44–45, 1963.

[Too63] Andrei L. Toom. The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers. *Soviet Mathematics Doklady*, 3:714–716, 1963. URL: http://toomandre.com/my-articles/engmat/MULT-E.PDF.

[TW13] Aleksandr Tuxanidy and Qiang Wang. Composed products and factors of cyclotomic polynomials over finite fields. *Designs, codes and cryptography*, 69(2):203–231, 2013. URL: https://link.springer.com/article/10.1007/s10623-012-9647-9.

[vdH04] Joris van der Hoeven. The truncated Fourier transform and applications. In *Proceedings of the 2004 international symposium on Symbolic and algebraic computation*, pages 290–296, 2004. URL: https://dl.acm.org/doi/10.1145/1005285.1005327.

[Wan23] William Wang, 2023. Personal communication.

[War12] Henry S. Warren. *Hacker's Delight*. Addison-Wesley, 2012.

[Win78] Shmuel Winograd. On Computing the Discrete Fourier Transform. *Mathematics of computation*, 32(141):175–199, 1978. URL: https://www.ams.org/journals/mcom/1978-32-141/S0025-5718-1978-0468306-4/?active=current.

[Win80] Shmuel Winograd. *Arithmetic Complexity of Computations*, volume 33. Society for Industrial and Applied Mathematics, 1980. URL: https://epubs.siam.org/doi/10.1137/1.9781611970364.

[WP06] André Weimerskirch and Christof Paar. Generalizations of the Karatsuba Algorithm for Efficient Implementations. Cryptology ePrint Archive, Paper 2006/224, 2006. URL: https://eprint.iacr.org/2006/224.

[WY21] Yansheng Wu and Qin Yue. Further factorization of $x^n - 1$ over a finite field (II). *Discrete Mathematics, Algorithms and Applications*, 13(06):2150070, 2021. URL: https://www.worldscientific.com/doi/10.1142/S1793830921500701.

[WYF18] Yansheng Wu, Qin Yue, and Shuqin Fan. Further factorization of $x^n - 1$ over a finite field. *Finite Fields and Their Applications*, 54:197–215, 2018. URL: https://www.sciencedirect.com/science/article/pii/S1071579718300996.

[Yan22] Bo-Yin Yang, 2022. Personal communication.

[Yan23] Bo-Yin Yang, 2023. Personal communication.

[YJX24] Yanze Yang, Yiran Jia, and Guangwu Xu. On Modular Algorithms and Butterfly Operations in Number Theoretic Transform. *arXiv preprint arXiv:2402.00675*, 2024. URL: https://arxiv.org/abs/2402.00675.

# A Modular Arithmetic for Principal Ideal Domains

Let $R$ be a principal ideal domain, $e_0, e_1 \in R$ be elements with $\gcd(e_0, e_1) = 1$. We assume implicitly that $R/\langle e_0 \rangle$ , $R/\langle e_1 \rangle \subset R$ by first fixing the representatives for each equivalence classes.

**Montgomery multiplication.** We define Montgomery multiplication as:

$$\frac{ab + \left(ab\left(-e_0^{-1} \bmod \langle e_1\rangle\right) \bmod \langle e_1\rangle\right) e_0}{e_1} \equiv abe_1^{-1} \bmod \langle e_0\rangle .$$

If $b$ is a constant, we compute the following instead:

$$\frac{a\left(be_1 \bmod \langle e_0\rangle\right) + \left(a\left(be_1 \bmod \langle e_0\rangle\right)\left(-e_0^{-1} \bmod \langle e_1\rangle\right) \bmod \langle e_1\rangle\right) e_0}{e_1} \equiv ab \bmod \langle e_0\rangle .$$

We prove the equivalence as follows.

*Proof.* Let $\texttt{term} = ab + \left(ab\left(-e_0^{-1} \bmod \langle e_1\rangle\right) \bmod \langle e_1\rangle\right) e_0$ be an abbreviation. By definition, we have

$$\texttt{term} \bmod \langle e_1\rangle = \left(ab + \left(ab\left(-e_0^{-1} \bmod \langle e_1\rangle\right) \bmod \langle e_1\rangle\right) e_0\right) \bmod \langle e_1\rangle \ = 0.$$

Therefore, $\texttt{term}$ is a multiple of $e_1$ and $\frac{\texttt{term}}{e_1} \in R$. It remains to show that $\frac{\texttt{term}}{e_1} \equiv abe_1^{-1} \bmod \langle e_0\rangle$. This boils down to the fact that $\texttt{term} \equiv ab \bmod \langle e_0\rangle$ and $e_0 \perp e_1$. □

**Barrett multiplication.** Suppose we are given an ideal $\langle e\rangle$ and a quotient ring $R/\langle e\rangle$ with a choice function implementing the inclusion $R/\langle e\rangle \subset R$. For an $a \in R$, we define $\left\llbracket \frac{a}{e} \right\rrbracket$ as the element in $R$ satisfying:

$$e\left\llbracket \frac{a}{e} \right\rrbracket = a - a \bmod \langle e\rangle .$$

For elements $a, b \in R$, Barrett multiplication computes the following

$$ab - \left\llbracket \frac{a\left\llbracket \frac{be_1}{e_0} \right\rrbracket}{e_1} \right\rrbracket e_0.$$

**Correspondence between Montgomery and Barrett multiplication.** We claim the following equation:

$$ab - \left\llbracket \frac{a\left\llbracket \frac{be_1}{e_0} \right\rrbracket}{e_1} \right\rrbracket e_0 = \frac{a\left(be_1 \bmod \langle e_0\rangle\right) + \left(a\left(be_1 \bmod \langle e_0\rangle\right)(-e_0^{-1}) \bmod \langle e_1\rangle\right) e_0}{e_1}.$$

*Proof.* We first find the following:

$$\left\llbracket \frac{be_1}{e_0} \right\rrbracket \bmod \langle e_1\rangle = \left(be_1 \bmod \langle e_0\rangle\right)(-e_0^{-1}) \bmod \langle e_1\rangle .$$

Then, we have:

$$\begin{aligned}
& ab - \left\llbracket \frac{a\left\llbracket \frac{be_1}{e_0} \right\rrbracket}{e_1} \right\rrbracket e_0 \\
= \quad & \frac{abe_1 - a\left\llbracket \frac{be_1}{e_0} \right\rrbracket e_0 + \left(a\left\llbracket \frac{be_1}{e_0} \right\rrbracket \bmod \langle e_1\rangle\right) e_0}{e_1} \\
= \quad & \frac{abe_1 - a\left\llbracket \frac{be_1}{e_0} \right\rrbracket e_0 + \left(a\left(be_1 \bmod \langle e_0\rangle\right)(-e_0^{-1}) \bmod \langle e_1\rangle\right) e_0}{e_1} \\
= \quad & \frac{abe_1 - a\left(be_1 - \left(be_1 \bmod \langle e_0\rangle\right)\right) + \left(a\left(be_1 \bmod \langle e_0\rangle\right)(-e_0^{-1}) \bmod \langle e_1\rangle\right) e_0}{e_1} \\
= \quad & \frac{a\left(be_1 \bmod \langle e_0\rangle\right) + \left(a\left(be_1 \bmod \langle e_0\rangle\right)(-e_0^{-1}) \bmod \langle e_1\rangle\right) e_0}{e_1}.
\end{aligned}$$

□

[KAK96, KA98] demonstrated the benefit of unsigned Montgomery multiplication for multi-precision arithmetic. They computed Montgomery multiplication with $\langle e_0 \rangle = 2^{ab}\mathbb{Z}$ and arithmetic modulo $2^a$. We leave the principal-ideal-domain view of the multi-precision case $\left\langle e_0^k \right\rangle$ and its relation to Barrett multiplication as future work.

# B Roots Defining Discrete Fourier Transforms

The goal of this section is to verify the following theorem qualifying the definability of DFTs.

**Theorem 2.** For a ring $R$, an element $\zeta \in R$, and a positive integer $n$, we have

$$\Phi_n(\zeta) = 0 \longrightarrow \left( \forall j = 1, \ldots, n-1, \sum_{0 \le i < n} \zeta^{ij} = 0 \right).$$

**Lemma 1.** For a positive integer $n$ and a proper divisor $j$ of $n$, $\Phi_n(x)$ is a divisor of $\sum_{0 \le i < \frac{n}{j}} x^{ij}$.

*Proof.*

$$\sum_{0 \le i < \frac{n}{j}} x^{ij} = \frac{x^n - 1}{x^j - 1} = \frac{\prod_{d|n} \Phi_d(x)}{\prod_{d|j} \Phi_d(x)} = \Phi_n(x) \cdot \prod_{d|n, d \nmid j, d<n} \Phi_d(x).$$

Therefore, $\Phi_n(x)$ is a divisor of $\sum_{0 \le i < \frac{n}{j}} x^{ij}$.

□

**Lemma 2.** For a ring $R$, an element $\zeta \in R$, a positive integer $n$, and a proper divisor $j$ of $n$, $\Phi_n(\zeta)$ is a divisor of $\sum_{0 \le i < \frac{n}{j} - 1} \zeta^{ij}$.

*Proof of Theorem 2.* For proving

$$\Phi_n(\zeta) = 0 \longrightarrow \left( \forall j = 1, \ldots, n-1, \sum_{0 \le i < n} \zeta^{ij} = 0 \right),$$

we distinguish between two cases: (i) $j|n$ and (ii) $j \nmid n$.

(i) $j|n$: $\sum_{0 \le i < n} \zeta^{ij} = \frac{n}{j} \sum_{0 \le i < \frac{n}{j}} \zeta^{ij} = \frac{n}{j} \cdot \Phi_n(\zeta) \cdot \prod_{d|n, d \nmid j, d<n} \Phi_d(\zeta) = 0.$

(ii) $j \nmid n, d = \gcd(j, n)$: $\sum_{0 \le i < n} \zeta^{ij} = \frac{n}{d} \sum_{0 \le i < \frac{n}{d}} \zeta^{ij} = \frac{n}{d} \sum_{0 \le i < \frac{n}{d}} \zeta^{id} = 0.$

□

**Theorem 3.** For a non-commutative ring $R$, Theorem 2 holds when $\zeta$ belongs to the center of $R$ where the center is the subset consisting of elements commuting to all elements in $R$.

**Remark 1.** For a ring $R$, a positive integer $n$, and an element $\zeta \in R$, $\zeta$ is a principal $n$-th root of unity defining a size-$n$ cyclic DFT over the coefficient ring $R/\langle \Phi_n(\zeta) \rangle$.

**Corollary 1.** For a positive integer $n$, 2 is a principal $n$-th root of unity defining a size-$n$ cyclic DFT over $\mathbb{Z}_{\Phi_n(2)}$. This is the well-known Fermat number transform [SS71, AB74].

# C Algebraic View of Good–Thomas FFT

This section presents an algebraic view of the Good–Thomas FFT [Goo58, Goo71]. Let $n_0, \dots, n_{d-1}$ be coprime integers and $n = \prod_j n_j$. We have $R[\mathbb{Z}_n] \cong \bigotimes_j R[\mathbb{Z}_{n_j}]$ as algebras, or equivalently, $R[x]/\langle x^n - 1\rangle \cong R[x_0, \dots, x_{d-1}]/\left\langle x_0^{n_0} - 1, \dots, x_{d-1}^{n_{d-1}} - 1\right\rangle$ as polynomial rings. We leave the proof as an exercise. For a $d$-dimensional polynomial $\boldsymbol{a} = \left(a_{i_0,\dots,i_{d-1}}\right)_{i_0,\dots,i_{d-1}} \in \bigotimes_j R[\mathbb{Z}_{n_j}]$, we define $\boldsymbol{a}_{i_0,\dots,i_{h-1}}$ as the $(d-h)$-dimensional tuple $\left(a_{i_0,\dots,i_{d-1}}\right)_{i_h,\dots,i_{d-1}}$ for $h > 0$ and $\boldsymbol{a}$ otherwise. Multiplying two elements $\boldsymbol{a}, \boldsymbol{b}$ is regarded as the following multi-dimensional cyclic convolution $\cdot_h$ defined recursively:

$$\boldsymbol{a}_{i_0,\dots,i_{h-1}} \cdot_h \boldsymbol{b}_{i_0,\dots,i_{h-1}} = \begin{cases} a_{i_0,\dots,i_{d-1}} b_{i_0,\dots,i_{d-1}} & \text{if } h = d,\\ \sum_k \left(\sum_{k_a+k_b=k} \boldsymbol{a}_{i_0,\dots,i_{h-1},k_a} \cdot_{h+1} \boldsymbol{b}_{i_0,\dots,i_{h-1},k_b}\right) x_h^k & \text{otherwise.} \end{cases}$$

Our goal is to implement $\cdot_0$, the multiplication in $R[x_0, \dots, x_{d-1}]/\left\langle x_0^{n_0} - 1, \dots, x_{d-1}^{n_{d-1}} - 1\right\rangle$.

We now apply homomorphism caching as follows. Let $f : R[x_{d-1}]/\left\langle x_{d-1}^{n_{d-1}} - 1\right\rangle \cong R_{d-1}$ be a homomorphism. We naturally have $R[x_0, \dots, x_{d-1}]/\left\langle x_0^{n_0} - 1, \dots, x_{d-1}^{n_{d-1}} - 1\right\rangle \cong R[x_0, \dots, x_{d-2}]/\left\langle x_0^{n_0} - 1, \dots, x_{d-2}^{n_{d-2}} - 1\right\rangle \otimes R_{d-1}$. This contextualizes $\cdot_{d-2}$ as

$$\boldsymbol{a}_{i_0,\dots,i_{d-3}} \cdot_{d-2} \boldsymbol{b}_{i_0,\dots,i_{d-3}} = \sum_k f^{-1}\left(\sum_{k_a+k_b=k} f(\boldsymbol{a}_{i_0,\dots,i_{d-3},k_a}) \cdot_{R_{d-1}} f(\boldsymbol{b}_{i_0,\dots,i_{d-3},k_b})\right) x_{d-2}^k.$$

We compute and cache $f(\boldsymbol{a}_{i_0,\dots,i_{d-3},k_b})$ and $f(\boldsymbol{a}_{i_0,\dots,i_{d-3},k_b})$, and use them for all the $k$'s. Similarly, the idea applies to all other dimensions. As a side note, arbitrary additive group isomorphism $\mathbb{Z}_n \cong \prod_j \mathbb{Z}_{n_j}$ suffices and there are $\phi(n)$ of them where $\phi$ is the Euler's totient function. In Appendix D, we survey the vector–radix transform [HMCS77] for optimizing the multi-dimensional transformation directly.

The notion of homomorphism caching is the actual reason making the multi-dimensional cyclic convolution fast. Historically, Good–Thomas FFT was first presented in [Goo58][14] as a correspondence between a DFT defined on $R[x]/\langle x^n - 1\rangle$ and a tensor product of the DFTs defined on $R[x_j]/\left\langle x_j^{n_j} - 1\right\rangle$. This was cited as a motivation of Cooley–Tukey FFT in [CT65]. [GS66, Sto66] pointed out the use of Cooley–Tukey FFT for cyclic convolutions. [Goo71] explained the differences between Good–Thomas FFT and Cooley–Tukey FFT, and acknowledged the application of multi-dimensional transform to multi-dimensional cyclic convolution. Based on this, we believe that homomorphism caching was already used in [Goo71].

# D Vector-Radix Transform

In Section 3.4, we know that one-dimensional size-$n$ cyclic convolution can be turned into a multi-dimensional cyclic convolution of dimensionals based on a coprime factorization of $n$. If we apply isomorphism for each dimension and cache the results, then we save the cost of transformation significantly. This section explains how one can save more by directly optimizing a multi-dimensional transform $\otimes_j f_j$ with vector-radix transformation [HMCS77].

Frequently, $f_j$ is a composition of one-dimensional isomorphisms shown in Section 3. Let's write $f_j = f_{j,0} \circ \cdots \circ f_{j,h-1}$. A crucial property while tensoring two compositions $f_{0,0} \circ f_{0,1}$ and $f_{1,0} \circ f_{1,1}$ is that $(f_{0,0} \circ f_{0,1}) \otimes (f_{1,0} \circ f_{1,1}) = (f_{0,0} \otimes f_{1,0}) \circ (f_{0,1} \otimes f_{1,1})$. Usually, $f_j$ can be characterized as a composition of multiplicative steps and additive steps. During the multiplicative steps, we only multiply coefficients by some constants.

[14] In the literature, people commonly attribute the idea to [Goo58, Tho63]. However, we are unable to locate the work [Tho63], and only find the publication information. If someone finds a copy, we would like to see how general the idea was in [Tho63].

For the additive steps, we perform additions and subtractions. One observation is that multiplicative steps are faster if we apply their composition directly. Suppose we have two multiplicative steps represented as matrix multiplications by $\begin{pmatrix} 1 & 0 \\ 0 & \zeta_0 \end{pmatrix} \otimes I_2$ and $I_2 \otimes \begin{pmatrix} 1 & 0 \\ 0 & \zeta_1 \end{pmatrix}$. Since $\left( \begin{pmatrix} 1 & 0 \\ 0 & \zeta_0 \end{pmatrix} \otimes I_2 \right) \left( I_2 \otimes \begin{pmatrix} 1 & 0 \\ 0 & \zeta_1 \end{pmatrix} \right) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \zeta_1 & 0 & 0 \\ 0 & 0 & \zeta_0 & 0 \\ 0 & 0 & 0 & \zeta_0\zeta_1 \end{pmatrix}$, we only need three multiplications on the right-hand side. If we compute with the left-hand side, then we need four multiplications. The high-dimensional generalization and $f_j$'s as series of compositions are obivous. See [AHY22] for applications.

# E Generalization of Rader's FFT

If $n$ is an odd prime power, we define $\mathcal{I}^* := \{z \in \mathbb{Z}_n | z \perp n\}$ and find a $g$ satisfying $\left\{g^k | k \in \mathbb{Z}_{\lambda(n)}\right\} = \mathcal{I}^*$ where $\lambda$ is Carmichael's lambda function.[15]. We can now split the DFT map $(a_j)_{j \in \mathcal{I}} \mapsto (\hat{a}_i)_{i \in \mathcal{I}}$ into $i \in \mathcal{I}^*$ and $i \in \mathcal{I} - \mathcal{I}^*$. For $i \in \mathcal{I}^*$, we move $\sum_{j \in \mathcal{I} - \mathcal{I}^*} a_j \omega_n^{ij}$ to the left-hand side and find

$$\hat{a}_{g^{\log_g i}} - \sum_{j \in \mathcal{I} - \mathcal{I}^*} a_j \omega_n^{ij} = \sum_{j \in \mathcal{I}^*} a_j \omega_n^{ij} = \sum_{-\log_g j \in \mathbb{Z}_{\lambda(n)}} a_{g^{\log_g j}} \omega_n^{g^{\log_g i + \log_g j}}.$$

Obviously, collecting the right-hand side forms a system of equations implementing a size-$\lambda(n)$ cyclic convolution [Win78, Section IV]. See [Ber23, Sections 4.12.3 and 4.12.4] for further generalization exploiting multiplicative subgroups in $\mathcal{I}^*$ for arbitrary $n$.

# F A Formal Treatment of Localization

For a ring $R$ and a multiplicative set $S \subset R$, localization is a standard technique introducing inverses of $S$. In this paper, we are interested in the case when $R$ is a polynomial ring over $\mathbb{Z}_{2^k}$ and $S$ is the set $\{1, 2, 4, \ldots\}$.

**Multiplicative set.** For a subset $S$ of a ring, we call it multiplicative set if it is closed under the ring multiplication. For example, $\left\{1, z, z^2, \ldots\right\} \subset \mathbb{Z}$ is a multiplicative set for any $z \in \mathbb{Z}$.

**Localization of a ring.** For a ring $R$ and a multiplicative set $S \subset R$, localization formally introduces divisions by elements in $S$. Consider the set $R \times S$ and the following equivalence relation

$$\forall (r_1, s_1), (r_2, s_2) \in R \times S, (r_1, s_1) \sim (r_2, s_2) \longleftrightarrow \exists s \in S, s s_2 r_1 = s s_1 r_2.$$

The localization of $R$ at $S$ is defined as the quotient ring $S^{-1}R \coloneqq R \times S/\sim$. The most common example is the set of rational numbers $\mathbb{Q}$ – we define $\mathbb{Q}$ as the localization of $\mathbb{Z}$ at $\mathbb{Z} - \{0\}$.

**Inverting a monomorphism.** Let $\mathcal{A}$ and $\mathcal{B}$ be rings and $\eta : \mathcal{A} \to \mathcal{B}$ be a ring monomorphism. For an integer $z \in \mathbb{Z} - \{0\}$, suppose we find a map $\psi_z : \eta(\mathcal{A}) \to \mathcal{A}$ such that

$$\forall \boldsymbol{a} \in \mathcal{A}, (\psi_z \circ \eta)(\boldsymbol{a}) = z\boldsymbol{a}.$$

[15] There is always such a $g$ since $n$ is an odd prime.

We define a homomorphism $\xi : \mathcal{Z}^{-1}\eta(\mathcal{A}) \to \mathcal{Z}^{-1}\mathcal{A}$ as

$$\forall z^{-k}\eta(\boldsymbol{a}) \in \mathcal{Z}^{-1}\eta(\mathcal{A}), \xi\left(z^{-k}\eta(\boldsymbol{a})\right) := z^{-1-k}\psi_z(\eta(\boldsymbol{a})).$$

If we restrict the image of $\xi$ to $\eta(\mathcal{A})$, we find $\xi|_{\eta(\mathcal{A})} := \left(\eta(\boldsymbol{a}) \mapsto z^{-1}\psi_z(\eta(\boldsymbol{a}))\right) = \eta^{-1}$. In summary, to invert $\eta$ while given $\psi_z$ with $z \in \mathbb{Z} - \{0\}$ non-invertible in $\mathcal{A}$, it suffices to define $\xi : \mathcal{Z}^{-1}\eta(\mathcal{A}) \to \mathcal{Z}^{-1}\mathcal{A}$ and apply $\xi|_{\eta(\mathcal{A})}$.

Notice that applying $\xi$ assumes an already existing approach for multiplying $z^{-1}$. An alternative way is to find $\psi_{z_0}$ and $\psi_{z_1}$ with $z_0 \perp z_1$ and integers $e_0, e_1$ satisfying $e_0 z_0 + e_1 z_1 = 1$, and define $\eta^{-1}$ as

$$\eta^{-1} := r \mapsto e_0\psi_{z_0}(r) + e_1\psi_{z_1}(r).$$

Since $e_0$ and $e_1$ are integers, $\eta^{-1}$ can be implemented entirely with arithmetic in $R$. [CK91] used localization and Schönhage's [Sch77] radix-2 and radix-3 FFTs for multiplying polynomials over arbitrary unital (possibly-noncommutative) rings.

# G Generalizations of Schönhage and Nussbaumer

Let $\boldsymbol{g}(x^{n_1}) \in R[x]$ be a degree-$n_0 n_1$ monic polynomial. Schönhage and Nussbaumer exploit the structure of $\boldsymbol{g}(x^{n_1})$ by introducing $x^{n_1} \sim y$ (so $R[x]/\langle \boldsymbol{g}(x^{n_1})\rangle \cong R[x,y]/\langle x^{n_1} - y, \boldsymbol{g}(y)\rangle$ ). Schönhage splits the structure into small structures by adjoining the defining condition. On the other hand, Nussbaumer adjoins a structure for splitting and uses $\boldsymbol{g}(x^{n_1})$ as the defining condition. We start by replacing $x^{n_1} - y$ with $\boldsymbol{h}(x)$ satisfying $\deg(\boldsymbol{h}) \geq 2n_1 - 1$.

**Schönhage.** Schönhage identifies an $n$ satisfying $\boldsymbol{g}(y)|(y^n - 1)$ and $\boldsymbol{h}(x)|\Phi_n(x)$, and treats $R[x]/\langle \boldsymbol{h}(x)\rangle$ as the coefficient ring. We then split as follows:

$$\frac{R[x]}{\langle \boldsymbol{g}(x^{n_1})\rangle} \cong \frac{R[x,y]}{\langle x^{n_1} - y, \boldsymbol{g}(y)\rangle} \hookrightarrow \frac{R[x,y]}{\langle \boldsymbol{h}(x), \boldsymbol{g}(y)\rangle} \cong \frac{\left(\frac{R[x]}{\langle \boldsymbol{h}(x)\rangle}\right)[y]}{\langle \boldsymbol{g}(y)\rangle} \cong \prod_i \frac{\left(\frac{R[x]}{\langle \boldsymbol{h}(x)\rangle}\right)[y]}{\langle y - x^i\rangle}.$$

Since $\boldsymbol{h}(x)|\Phi_n(x)$, $\Phi_n(x) = 0 \in R[x]/\langle \boldsymbol{h}(x)\rangle$ and $x$ is a principal $n$-th root of unity defining a size-$n$ cyclic FFT. Furthermore, since $\boldsymbol{g}(y)|\,(y^n - 1)$, we apply truncation and obtain an FFT with indeterminate $y$, coefficient ring $R[x]/\langle \boldsymbol{h}(x)\rangle$, and polynomial modulus $\boldsymbol{g}(y)$.

**Nussbaumer.** Nussbaumer identifies an $n$ satisfying $\boldsymbol{g}(y)|\Phi_n(x)$ and $\boldsymbol{h}(x)|(x^n - 1)$, and splits as follows:

$$\frac{R[x]}{\langle \boldsymbol{g}(x^{n_1})\rangle} \cong \frac{R[x,y]}{\langle x^{n_1} - y, \boldsymbol{g}(y)\rangle} \hookrightarrow \frac{R[x,y]}{\langle \boldsymbol{h}(x), \boldsymbol{g}(y)\rangle} \cong \frac{\left(\frac{R[y]}{\langle \boldsymbol{g}(y)\rangle}\right)[x]}{\langle \boldsymbol{h}(x)\rangle} \cong \prod_i \frac{\left(\frac{R[y]}{\langle \boldsymbol{g}(y)\rangle}\right)[x]}{\langle x - y^i\rangle}.$$

See [MV83a, MV83b, Ber01] for more discussions generalizing the notion of principal roots of unity to automorphisms defining FFTs.

# H Applications of Truncation

## H.1 $R[x]/\langle x^r + 1\rangle$ from $R[x]/\langle x^{2r} - 1\rangle$ for $r \perp 2$

Our second application is to systematically generalize the isomorphism $R[x]/\langle x^r + 1\rangle \cong R[x,y]/\langle x + y, y^r - 1\rangle$ for an odd $r$. Let $\psi : \mathbb{Z}_{2r} \cong \mathbb{Z}_2 \times \mathbb{Z}_r$ be the additive group isomorphism $1 \mapsto (1,1)$. Recall that $\psi$ induces an algebra isomorphism $\psi' : R[x]/\left\langle x^{2r} - 1\right\rangle \cong$

$R[z]/\langle z^2-1\rangle \otimes R[y]/\langle y^r-1\rangle$ (cf. Section C). From $R[z]/\langle z^2-1\rangle \cong \prod R[z]/\langle z\pm 1\rangle$, we have

$$\frac{R[x]}{\langle x^r+1\rangle} \cong \psi'^{-1}\left(\frac{R[z]}{\langle z+1\rangle} \otimes \frac{R[y]}{\langle y^r-1\rangle}\right) \cong \frac{R[x,y]}{\langle x+y, y^r-1\rangle}.$$

Similarly, whenever we are working on a polynomial ring with modulus a factor of $x^{q_0q_1}-1$ for $q_0 \perp q_1$, we can always look for transformations for $R[z]/\langle z^{q_0}-1\rangle \otimes R[y]/\langle y^{q_1}-1\rangle$ and pull them back to the desired domain (in our example, we exploit $R[z]/\langle z^2-1\rangle \cong \prod R[z]/\langle z\pm 1\rangle$). Examples in the literature are the CRT negacyclic/tricyclic transform in [HVDH22, Sections 3.5 and 3.6].

One should notice that we could derive optimizations by exploiting some properties of a factor of $x^{q_0q_1}-1$ and bring the resulting computation back to the isomorphism $R[x]/\langle x^{q_0q_1}-1\rangle \cong R[z]/\langle z^{q_0}-1\rangle \otimes R[y]/\langle y^{q_1}-1\rangle$. The optimization comes from splitting $\Phi_{3\cdot 2^k} = \left(x^{2^k}-\omega_6\right)\left(x^{2^k}-\omega_6^5\right)$ exploiting the identity $\omega_6+\omega_6^5=1$ [LS19] and $\Phi_3=(x-\omega_3)(x-\omega_3^2)$ exploiting the identity $\omega_3+\omega_3^2=-1$ [DV78a, Has22, AHY22]. We leave them as exercises for the readers.

### H.2 Nussbaumer from Schönhage

As we know, for arbitrary $\boldsymbol{g}$ a factor of $x^{2^k}-1$, we can derive the corresponding Schönhage's FFT via truncating the Schönhage's FFT for $R[x]\Big/\Big\langle x^{2^k}-1\Big\rangle$. We now show how to exploit the same idea for Nussbaumer's FFT systematically. An example is to derive the Nussbaumer for $R[x]/\langle x^{1536}+1\rangle$ from the Schönhage for $R[x]/\langle (x^{1024}+1)(x^{512}-1)\rangle$. Given polynomials $\boldsymbol{g}(z)|(z^{n'}-1)$ and $\boldsymbol{h}(z)|\Phi_{n'}(z)$ with $\deg(\boldsymbol{g})=2n_0$ and $\deg(\boldsymbol{h})=2n_1$, we have a size-$2n_0n_1$ transformation via Schönhage as follows

$$\frac{R[x]}{\langle \boldsymbol{g}(x^{n_1})\rangle} \cong \frac{R[x,y]}{\langle x^{n_1}-y, \boldsymbol{g}(y)\rangle} \hookrightarrow \frac{R[x,y]}{\langle \boldsymbol{h}(x), \boldsymbol{g}(y)\rangle} \cong \frac{\left(\frac{R[x]}{\langle \boldsymbol{h}(x)\rangle}\right)[y]}{\langle \boldsymbol{g}(y)\rangle}.$$

We exchange $x$ and $y$ in $\left(R[x]/\langle \boldsymbol{h}(x)\rangle\right)[y]/\langle \boldsymbol{g}(y)\rangle$, and invert the derivation of Nussbaumer. This gives us the following transformation for Nussbaumer

$$\frac{R[x]}{\langle \boldsymbol{h}(x^{n_0})\rangle} \cong \frac{R[x,y]}{\langle x^{n_0}-y, \boldsymbol{h}(y)\rangle} \hookrightarrow \frac{R[x,y]}{\langle \boldsymbol{g}(x), \boldsymbol{h}(y)\rangle} \cong \frac{\left(\frac{R[y]}{\langle \boldsymbol{h}(y)\rangle}\right)[x]}{\langle \boldsymbol{g}(x)\rangle}.$$

# I Interprating Multiplications in $R[x]/\langle x^n-\alpha x-\beta\rangle$ as TMVPs

We outline [Yan23]'s ideas interprating multiplications in $R[x]/\langle x^n-\alpha x-\beta\rangle$ as Toeplitz matrix-vector products for generic $\beta$ as follows. For reducing modulo $x^n-\alpha x$, if we additionally add the element $\sum_{j=0}^{n-2}\alpha b_{n-1-j}a_j$ to $c_0$, then the resulting computation is compatible with a Toeplitz matrix-vector product. This gives us the following transformation matrix mapping $\boldsymbol{a}$ to $\boldsymbol{ab}\in R[x]/\langle x^n-\alpha x-\beta\rangle$:

$$M_0+M_1+M_2'-\begin{pmatrix}\alpha b_{n-1} & \cdots & \alpha b_1 & 0\\ 0 & \cdots & 0 & 0\\ \vdots & \ddots & \vdots & \vdots\\ 0 & \cdots & 0 & 0\end{pmatrix}$$

where $M_0$ and $M_1$ are the same as previous paragraph and

$$M_2' = \mathbf{Toeplitz}_n(0,\ldots,0,\alpha b_{n-1},\ldots,\alpha b_1,0).$$

Since $M_0 + M_1 + M_2'$ is the Toeplitz matrix

$$\mathbf{Toeplitz}_n(b_{n-1}, \ldots, b_1, b_0 + \alpha b_{n-1}, \beta b_{n-1} + \alpha b_{n-2}, \ldots, \beta b_2 + \alpha b_1, \beta b_1),$$

$\boldsymbol{ab} \in R[x]/\langle x^n - \alpha x - \beta \rangle$ can be written as a Toeplitz matrix-vector product with post-processing as follows:

$$\boldsymbol{ab} \bmod (x^n - \alpha x - \beta) = (M_0 + M_1 + M_2')\, \boldsymbol{a} - \begin{pmatrix} \alpha b_{n-1} & \cdots & \alpha b_1 & 0 \\ 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \end{pmatrix} \boldsymbol{a}.$$

## J A Formal Treatment of Bilinear Systems

Let $\mathcal{A}, \mathcal{B}, \mathcal{C}$ be modules over the ring $R$. We call a map $\eta : \mathcal{A} \times \mathcal{B} \to \mathcal{C}$ a bilinear map if

- $\forall \boldsymbol{a} \in \mathcal{A}, \eta(\boldsymbol{a}, -) : \mathcal{B} \to \mathcal{C}$ is a module homomorphism.
- $\forall \boldsymbol{b} \in \mathcal{B}, \eta(-, \boldsymbol{b}) : \mathcal{A} \to \mathcal{C}$ is a module homomorphism.

Suppose we have maps $\psi : \mathcal{A}^* \to \mathcal{A}', \kappa : \mathcal{B} \to \mathcal{B}', \iota : \mathcal{C}' \to \mathcal{C}^*$, and a bilinear map $\xi : \mathcal{C}' \times \mathcal{B}' \to \mathcal{A}'$ satisfying

$$\forall \boldsymbol{b} \in \mathcal{B}, \xi\left(-, \kappa\left(\boldsymbol{b}\right)\right) = \psi \circ \eta\left(-, \boldsymbol{b}\right)^* \circ \iota.$$

If $\eta\left(-, \boldsymbol{b}\right) = f_{\boldsymbol{b}} \circ g_{\boldsymbol{b}}$ for some $f_{\boldsymbol{b}}$ and $g_{\boldsymbol{b}}$, we have the corresponding factorization for $\xi\left(-, \kappa\left(\boldsymbol{b}\right)\right)$:

$$\forall \boldsymbol{b} \in \mathcal{B}, \xi\left(-, \kappa\left(\boldsymbol{b}\right)\right) = \psi \circ g_{\boldsymbol{b}}^* \circ f_{\boldsymbol{b}}^* \circ \iota.$$

In Section 5.5, we present the ideas with bilinear systems. We now rephrase the core idea of [Win80] as follows: Let's assume $\mathcal{A}' = \mathcal{A}, \mathcal{B}' = \mathcal{B}$, $\mathcal{C}' = \mathcal{C}$, $\psi = \boldsymbol{a}^* \mapsto \boldsymbol{a}$, $\kappa = \mathrm{id}_{\mathcal{B}}$, and $\iota = \boldsymbol{c} \mapsto \boldsymbol{c}^*$. For finite index sets $\mathcal{I}, \mathcal{J}, \mathcal{K}$ and $(r_{(i,j,k)})_{(i,j,k) \in \mathcal{I} \times \mathcal{J} \times \mathcal{K}}$, define $\boldsymbol{a} = (a_i)_{i \in \mathcal{I}} \in \mathcal{A}, \boldsymbol{b} = (b_j)_{j \in \mathcal{J}} \in \mathcal{B}, \boldsymbol{c} = (c_k)_{k \in \mathcal{K}} \in \mathcal{C}$. Then, we write

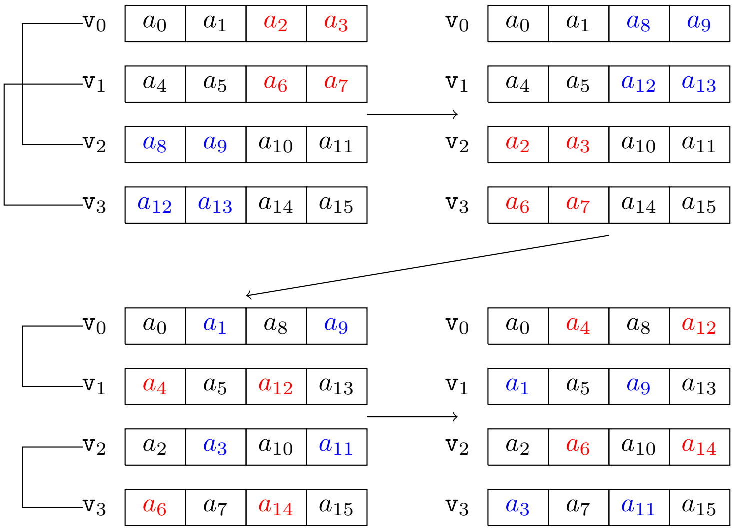
$$\begin{cases} \left( \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} r_{(i,j,k)} a_i b_j \right)_{k \in \mathcal{K}} & = \eta(-, \boldsymbol{b})(\boldsymbol{a}) \\ \left( \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} r_{(i,j,k)} c_k b_j \right)_{i \in \mathcal{I}} & = \xi(-, \boldsymbol{b})(\boldsymbol{c}) \end{cases}$$

and find $\xi(-, \boldsymbol{b}) = (\psi \circ \eta(-, \boldsymbol{b})^* \circ \iota)$.

## K Implementing Transposition Matrices

We give a conceptual review of transposing a matrix with vector instructions. The idea was introduced by [Flo72] under the context of permuting with pages and more recently by [War12, Section 7.3] for permuting bit matrices. It was also used in [NG21, BBCT22, BHK+22b, CCHY24, Hwa24, HLY24] for vectorized polynomial multiplications.

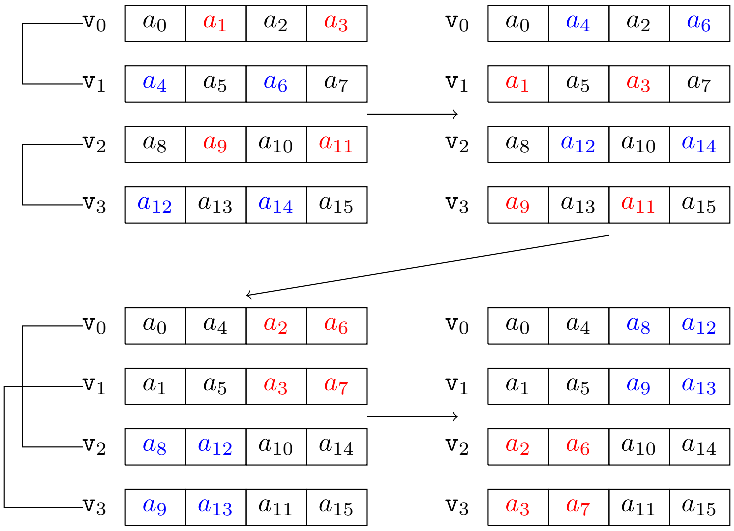
For simplicity, we illustrate how to transpose a 4 × 4 matrix. Suppose the matrix is represented in the row-major fashion by four vectors $\mathtt{v}_0, \ldots, \mathtt{v}_3$ with each holding four elements. There are two steps: (i) transpose as if we are transposing a 2 × 2 matrix with each entries 2 × 2 matrix, and (ii) transpose each of the 2 × 2 matrices. We implement step (i) by permuting the pairs $(\mathtt{v}_0, \mathtt{v}_2)$ and $(\mathtt{v}_1, \mathtt{v}_3)$, and for step (ii), we permute the pairs



$\mathtt{v}_0$ $a_0$ $a_1$ $a_2$ $a_3$
$\mathtt{v}_1$ $a_4$ $a_5$ $a_6$ $a_7$
$\mathtt{v}_2$ $a_8$ $a_9$ $a_{10}$ $a_{11}$
$\mathtt{v}_3$ $a_{12}$ $a_{13}$ $a_{14}$ $a_{15}$
$\mathtt{v}_0$ $a_0$ $a_1$ $a_8$ $a_9$
$\mathtt{v}_1$ $a_4$ $a_5$ $a_{12}$ $a_{13}$
$\mathtt{v}_2$ $a_2$ $a_3$ $a_{10}$ $a_{11}$
$\mathtt{v}_3$ $a_6$ $a_7$ $a_{14}$ $a_{15}$
$\mathtt{v}_0$ $a_0$ $a_1$ $a_8$ $a_9$
$\mathtt{v}_1$ $a_4$ $a_5$ $a_{12}$ $a_{13}$
$\mathtt{v}_2$ $a_2$ $a_3$ $a_{10}$ $a_{11}$
$\mathtt{v}_3$ $a_6$ $a_7$ $a_{14}$ $a_{15}$
$\mathtt{v}_0$ $a_0$ $a_4$ $a_8$ $a_{12}$
$\mathtt{v}_1$ $a_1$ $a_5$ $a_9$ $a_{13}$
$\mathtt{v}_2$ $a_2$ $a_6$ $a_{10}$ $a_{14}$
$\mathtt{v}_3$ $a_3$ $a_7$ $a_{11}$ $a_{15}$

Figure 2: Top-down transposition of the $4 \times 4$ matrix with rows $(a_0, \dots, a_3)$, $(a_4, \dots, a_7)$, $(a_8, \dots, a_{11})$, and $(a_{12}, \dots, a_{15})$.

$(\mathtt{v}_0, \mathtt{v}_1)$ and $(\mathtt{v}_2, \mathtt{v}_3)$. See Figure 2 for an illustration. Obviously, the idea generalizes to transposing arbitrary $2^k \times 2^k$ matrices – we transpose as if we are transposing a $2 \times 2$ matrix with entries $2^{k-1} \times 2^{k-1}$ matrices, and transpose the $2^{k-1} \times 2^{k-1}$ matrices recursively. Since we start from the root level of the recursion tree, we call the approach top-down transposition. Notice that we can swap the order of (i) and (ii). We call the resulting approach bottum-up transposition (cf. Figure 3).



$\mathtt{v}_0$ $a_0$ $a_1$ $a_2$ $a_3$
$\mathtt{v}_1$ $a_4$ $a_5$ $a_6$ $a_7$
$\mathtt{v}_2$ $a_8$ $a_9$ $a_{10}$ $a_{11}$
$\mathtt{v}_3$ $a_{12}$ $a_{13}$ $a_{14}$ $a_{15}$
$\mathtt{v}_0$ $a_0$ $a_4$ $a_2$ $a_6$
$\mathtt{v}_1$ $a_1$ $a_5$ $a_3$ $a_7$
$\mathtt{v}_2$ $a_8$ $a_{12}$ $a_{10}$ $a_{14}$
$\mathtt{v}_3$ $a_9$ $a_{13}$ $a_{11}$ $a_{15}$
$\mathtt{v}_0$ $a_0$ $a_4$ $a_2$ $a_6$
$\mathtt{v}_1$ $a_1$ $a_5$ $a_3$ $a_7$
$\mathtt{v}_2$ $a_8$ $a_{12}$ $a_{10}$ $a_{14}$
$\mathtt{v}_3$ $a_9$ $a_{13}$ $a_{11}$ $a_{15}$
$\mathtt{v}_0$ $a_0$ $a_4$ $a_8$ $a_{12}$
$\mathtt{v}_1$ $a_1$ $a_5$ $a_9$ $a_{13}$
$\mathtt{v}_2$ $a_2$ $a_6$ $a_{10}$ $a_{14}$
$\mathtt{v}_3$ $a_3$ $a_7$ $a_{11}$ $a_{15}$

Figure 3: Bottum-up transposition of the $4 \times 4$ matrix with rows $(a_0, \dots, a_3)$, $(a_4, \dots, a_7)$, $(a_8, \dots, a_{11})$, and $(a_{12}, \dots, a_{15})$.

# L Constructing the Column Representation of a Toeplitz Matrix

Algorithm 10 constructs the columns with only memory loads, and Algorithm 11 replaces some memory instructions with permutation instructions.

**Algorithm 10** Constructing the columns of a Toeplitz matrix from its array form with memory loads [CCHY24].

**Inputs:** Array `M[8] = {0,` $a'_3$`,` $a'_2$`,` $a'_1$`,` $a_0$`,` $a_1$`,` $a_2$`,` $a_3$`}`.
**Outputs:** Vector registers `t0` $= (a_0, a_1, a_2, a_3)$, `t1` $= (a'_1, a_0, a_1, a_2)$, `t2` $= (a'_2, a'_1, a_0, a_1)$, and `t3` $= (a'_3, a'_2, a'_1, a_0)$.

```
t0 = M[4-7]
t1 = M[3-6]
t2 = M[2-5]
t3 = M[1-4]
                                        ▷ Memory load.
```

**Algorithm 11** Constructing the columns of a Toeplitz matrix from its array form with memory loads and permutations [Hwa24].

**Inputs:** Array `M[8] = {0,` $a'_3$`,` $a'_2$`,` $a'_1$`,` $a_0$`,` $a_1$`,` $a_2$`,` $a_3$`}`.
**Outputs:** Vector registers `t0` $= (a_0, a_1, a_2, a_3)$, `t1` $= (a'_1, a_0, a_1, a_2)$, `t2` $= (a'_2, a'_1, a_0, a_1)$, and `t3` $= (a'_3, a'_2, a'_1, a_0)$.

```
t0 = M[4-7]
t3 = M[0-3]
                                        ▷ Memory load.
t1 = ext(t3, t0, 3)
t2 = ext(t3, t0, 2)
t3 = ext(t3, t0, 1)
```