Technical Report: Even Faster Polynomial Multiplication for NTRU Prime with AVX2

Vincent Hwang

Max Planck Institute for Security and Privacy, Bochum, Germany
vincentvbh7@gmail.com

Abstract. This paper implements a vectorization–friendly polynomial multiplication for the NTRU Prime parameter sets `ntrulpr761/sntrup761` with AVX2 based on the recently released work [Chen, Chung, Hwang, Liu, and Yang, Cryptology ePrint Archive, 2023/541]. Compared to the state-of-the-art optimized implementation by [Bernstein, Brumley, Chen, and Tuveri, USENIX Security 2022], our big-by-big polynomial multiplication is 1.77×, 1.9×, and 1.92× faster on Haswell, Skylake, and Comet Lake.

Keywords: NTRU Prime · AVX2 · Good–Thomas FFT · Rader's FFT

1 Introduction

OpenSSH 9.0 currently uses the hybrid `sntrup761x25519-sha512` key exchange by default[1]. This paper demonstrates the applicability of [CCH+23]'s ideas on polynomial multiplication for the NTRU Prime parameter sets `ntrulpr761/sntrup761` with AVX2. Our target is the polynomial multiplication in $\mathbb{Z}_{4591}[x]/\langle x^{761} - x - 1\rangle$ used by `ntrulpr761/sntrup761`. We refer to [BBC+20] for the specification of NTRU Prime. For `ntrulpr761/sntrup761`, maintaining the vectorization–friendliness while working over $\mathbb{Z}_{4591}$ was challenging. While computing the product of two polynomials, if one of the polynomials has coefficients within a small range, we call the computing task a big-by-small polynomial multiplication. Otherwise, we call it a big-by-big polynomial multiplication. In NTRU Prime, all the polynomial multiplications in the reference implementation are big by small. Nevertheless, big-by-big polynomial multiplications are used for improving the key generation of `sntrup` [BY19, BBCT22] and can replace big-by-small polynomial multiplications if the performance is improved.

[BBCT22]'s big-by-big polynomial multiplication on Haswell with AVX2 is roughly 1.5 times slower than their big-by-small one, while it was already known that on an ARM Cortex-M4 implementing Armv7E-M with limited SIMD support, big-by-big polynomial multiplication is faster than big-by-small polynomial multiplication [ACC+21, AHY22]. The reason is that when the SIMD support is raised from 2 halfwords (Armv7E-M) to 16 (AVX2), [BBCT22] applied Schönhage [Sch77] and Nussbaumer [Nus80] crafting radix-2 roots of unity. Since Schönhage and Nussbaumer usually double the number of coefficients, this eventually leads to many base multiplications (small-degree polynomial multiplications).

[CCH+23] explored various vectorization ideas for NTRU and NTRU Prime on an ARM Cortex-A72 with Neon. We are interested in their fast Fourier transformations (FFTs) for `ntrulpr761/sntrup761`. To ensure vectorization–friendliness, they first introduced the equivalence $x^{16} \sim y$. They then applied a 3-dimensional Good–Thomas

[1] See "New features" in https://marc.info/?l=openssh-unix-dev&m=164939371201404&w=2.

FFT [Goo58] based on the coprime factorization $\frac{1632}{16} = 17 \cdot 3 \cdot 2$. Radix-3 and radix-2 cyclic FFTs are obvious. For the Radix-17 cyclic FFT, they applied Rader's FFT [Rad68] to convert the computation into a size-16 cyclic convolution. The remaining problems are multiplications in the product ring $\prod_i R[x]/\langle x^{16} \pm \omega_{51}^i \rangle$ for $i = 0, \dots, 101$. [CCH+23]'s `Good-Rader-Bruun` applied Cooley–Tukey FFT [CT65] to 48 size-16 problems of the form $R[x]/\langle x^{16} - \omega_{51}^i \rangle$, Bruun's FFT [Bru78, BC87, BGM93] to 48 size-16 problems of the form $R[x]/\langle x^{16} + \omega_{51}^i \rangle$, and schoolbook multiplication to the remaining size-16 problems. We propose an implementation similar to [CCH+23]'s `Good-Rader-Bruun` but discard Bruun's FFT due to the relatively expensive polynomial reduction with AVX2, which lacks long multiplications and incurs a long dependency chain while interleaving and deinterleaving. Our big-by-big polynomial multiplication is 1.77 times faster than [BBCT22]'s on Haswell with AVX2.

Code. Our source code can be found at `https://github.com/vincentvbh/NTRU_Prime_polymul_AVX2` under CC0 license.

2 Preliminaries

2.1 AVX2 Modular Multiplication and Reduction

We recall the Montgomery multiplication [Mon85] and Barrett reduction [Bar86] from [Sei18]. `vpmullw` multiplies corresponding signed 16-bit values and places the lower 16-bit values to the destination register. `vpmulhw` places the upper 16-bit values to the destination instead. `vpmulhrsw` effectively computes $\left\lfloor \frac{ab}{2^{15}} \right\rceil$ from the signed 16-bit values a and b. For signed 16-bit values a and b, Montgomery multiplication [Mon85, Sei18] computes a representative of $ab2^{-16} \bmod {}^{\pm}q$ with

$$\left\lfloor \frac{ab - \left(abq' \bmod {}^{\pm}2^{16}\right) q}{2^{16}} \right\rfloor \equiv ab2^{-16} \pmod q$$

where $q' = q^{-1} \bmod {}^{\pm}2^{16}$ is precomputed. Algorithm 1 is an illustration. If b is known in prior, we replace $(b, bq' \bmod {}^{\pm}2^{16})$ with $\left(b2^{16} \bmod {}^{\pm}q, \left(b2^{16} \bmod {}^{\pm}q\right) q' \bmod {}^{\pm}2^{16}\right)$ to save one multiplication and mitigate the scaling by 2^{-16}. Algorithm 2 is an illustration.

Barrett reduction [Bar86, Sei18] reduces a value a by computing

$$a - \left\lfloor \frac{a \left\lfloor \frac{2^{15}}{q} \right\rceil}{2^{15}} \right\rceil q \equiv a \pmod q.$$

Algorithm 3 is an illustration. In the case of $q = 4591$, one can show (by brute-force testing) that for $a \in [-32768, 32767]$, the results lies in $[-2881, 2881]$.

Algorithm 1 Montgomery multiplication [Sei18].

Inputs: $\mathtt{a} = a, \mathtt{b} = b$.
Constants: $q = 4591$, $q' = q^{-1} \bmod {}^{\pm}2^{16} = 15631$.
Output: $\mathtt{c} = c = \left\lfloor \frac{ab - \left(abq' \bmod {}^{\pm}2^{16}\right) q}{2^{16}} \right\rfloor \equiv ab2^{-16} \bmod {}^{\pm}q$.

```
vpmullw  b, q', lo
vpmullw lo, a, lo
vpmulhw  b, a, hi
vpmulhw lo, q, lo
vpsubw  lo, hi, c
```

Algorithm 2 Montgomery multiplication with precomputation [Sei18].

Inputs: $\texttt{a} = a$.

Constants: $q = 4591$, $\texttt{b} = b2^{16} \bmod {}^{\pm}q$, $\texttt{b'} = \left(b2^{16} \bmod {}^{\pm}q\right) q^{-1} \bmod {}^{\pm}2^{16}$.

Output: $\texttt{c} = c = \left\lfloor \frac{a\left(b2^{16} \bmod {}^{\pm}q\right) - \left(a\left(\left(b2^{16} \bmod {}^{\pm}q\right)q^{-1} \bmod {}^{\pm}2^{16}\right)\right)q}{2^{16}} \right\rceil \equiv ab \bmod {}^{\pm}q$.

```
vpmullw b', a, lo
vpmulhw  b, a, hi
vpmulhw lo, q, lo
vpsubw  lo, hi, c
```

Algorithm 3 Barrett reduction [Sei18].

Input: $\texttt{a} = a$.

Constants: $q = 4591$, $\bar{q} = \left\lfloor \frac{2^{15}}{q} \right\rceil = 7$.

Output: $\texttt{a} = a' = a - \left\lfloor \frac{a\bar{q}}{2^{15}} \right\rceil q$, $-2881 \leq a' \leq 2881$.

```
vpmulhrsw  a, q̄, hi
vpmullw   hi, q, hi
vpsubw    hi, a, a
```

2.2 Chinese Remainder Theorem

In this paper, all the rings are commutative and unital. Let R be a ring. For elements $e_0, e_1 \in R$, we call them orthogonal if $e_0 e_1 = 0$. An element $e \in R$ is called idempotent if $e^2 = e$. For orthogonal idempotent elements e_0 and e_1 in R satisfying $e_0 + e_1 = 1$, we have the ring isomorphism $R \cong R/(1-e_0)R \times R/(1-e_1)R$. This easily generalizes to finitely many orthogonal idempotent elements $(e_0, \ldots, e_{d-1})$ with $\sum_i e_i = 1$ realizing $R \cong \prod_i R/(1-e_i)R$. Explicitly, we have the isomorphism $\Phi : R \to \prod_i \frac{R}{(1-e_i)R}$ mapping a to the n-tuple $(a \bmod (1-e_i)R)$ with the inverse $\Psi : (\hat{a}_i) \mapsto \sum_i \hat{a}_i e_i$ [Bou89].

We are interested in two cases: $R[x] \Big/ \left\langle \prod_{i_0,\ldots,i_{h-1}} \boldsymbol{g}_{i_0,\ldots,i_{h-1}} \right\rangle$ for coprime polynomials $\boldsymbol{g}_{i_0,\ldots,i_{h-1}}$'s in $R[x]$ and $\mathbb{Z}_{q_0 \cdots q_{d-1}}$ for coprime integers $q_0, \ldots, q_{d-1}$.

2.3 Cooley–Tukey FFT

Let $n = \prod_j n_j$, and i_j run over $0, \ldots, n_j - 1$ for each j. The Cooley–Tukey FFT [CT65] computes with the following isomorphsisms:

$$\frac{R[x]}{\left\langle \prod_{i_0,\ldots,i_{h-1}} \boldsymbol{g}_{i_0,\ldots,i_{h-1}} \right\rangle} \cong \prod_{i_0} \frac{R[x]}{\left\langle \prod_{i_1,\ldots,i_{h-1}} \boldsymbol{g}_{i_0,\ldots,i_{h-1}} \right\rangle} \cong \cdots \cong \prod_{i_0,\ldots,i_{h-1}} \frac{R[x]}{\left\langle \boldsymbol{g}_{i_0,\ldots,i_{h-1}} \right\rangle}$$

by choosing $\boldsymbol{g}_{i_0,\ldots,i_{h-1}} = x - \zeta \omega_n^{\sum_l i_l \prod_{j<l} n_j}$ where ω_n is a principal n-th root of unity[2]. The Cooley–Tukey FFT is invertible if we can "invert" n. Since $\prod_{i_0,\ldots,i_{h-1}} \boldsymbol{g}_{i_0,\ldots,i_{h-1}} = x^n - \zeta^n$, we now can multiply polynomials in $R[x]/\langle x^n - \zeta^n \rangle$ via $\prod_{i_0,\ldots,i_{h-1}} R[x] \Big/ \left\langle \boldsymbol{g}_{i_0,\ldots,i_{h-1}} \right\rangle$.

2.4 Good–Thomas FFT

Let $n = \prod_j q_j$ for coprime integers $q_0, \ldots, q_{d-1}$. There are two ways for stating Good–Thomas FFT [Goo58]: (i) as an isomorphism from a group algebra to a tensor product

[2] $\forall j = 1, \ldots, n-1, \sum_i \omega_n^{ij} = 0$.

of associative algebras; and (ii) as a correspondence between one-dimensional FFT and multi-dimensional FFT. (ii) was stated in [Goo58]. (i) is a more general statement in the modern algebra language and is apparent from [Goo58].

Recall that we have a group isomorphism $\mathbb{Z}_n \cong \prod_j \mathbb{Z}_{q_j}$. This implies an isomorphism between the group algebras $R[\mathbb{Z}_n]$ and $R\left[\prod_j \mathbb{Z}_{q_j}\right]$. Notice that $R\left[\prod_j \mathbb{Z}_{q_j}\right]$ is isomorphic to the tensor product $\bigotimes_j R\left[\mathbb{Z}_{q_j}\right]$. Suppose n is invertible in R, and there is a principal n-th root of unity $\omega_n \in R$ realizing the isomorphism $R[x]/\langle x^n - 1\rangle \cong \prod_i R[x]/\left\langle x - \omega_n^i \right\rangle$. By definition, we also have a principal n_j-th root of unity ω_{n_j} for each j. We choose $\omega_{n_j} := \omega_n^{e_j}$ so $\prod_j \omega_{n_j} = \omega_n^{\sum_j e_j} = \omega_n$. This allows us to relate the tensor product $\bigotimes_j \left(R[x_j]/\langle x_j^{n_j} - 1\rangle \cong \prod_{i_j} R[x_j]\big/\left\langle x_j - \omega_{n_j}^{i_j} \right\rangle \right)$ to $R[x]/\langle x^n - 1\rangle \cong \prod_i R[x]/\left\langle x - \omega_n^i \right\rangle$ via the relation $x \sim \prod_j x_j$. Figure 1 is an illustration.

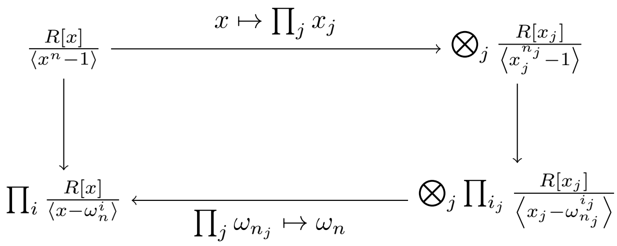

Figure 1: Commutative diagram of Good–Thomas FFT. Notice that $x \mapsto \prod_j x_j$ itself is already an FFT improving the overall asymptotic behavior.

Vectorization–friendly Good–Thomas first introduces $x^v \sim y$ for $R[x]/\langle x^{nv} - 1\rangle$ and operates as a polynomial ring modulo $y^n - 1$ [FP07, AHY22, CCH+23].

2.5 Rader's FFT

Let p be prime. Rader's FFT [Rad68] computes the map $R[x]/\langle x^p - 1\rangle \cong \prod_i R[x]/\left\langle x - \omega_p^i \right\rangle$ with a size-$(p-1)$ cyclic convolution. Since p is a prime, there is a g with $\{1, \dots, p-1\} = \left\{g^1, \dots, g^{p-1}\right\}$. This allows us to introduce two equivalences for $(\hat{a}_j) = \sum_{i=0}^{p-1} a_i \omega_p^{ij}$: (i) $(1, 2, \dots, p-1) \cong (g, g^2, \dots, \dots, g^{p-1})$ and (ii) $(1, 2, \dots, p-1) \cong (g^{-1}, g^{-2}, \dots, g^{-(p-1)})$. If we map $j \mapsto g^j$ and $i \mapsto g^{-i}$, we have $\left(\hat{a}_{g^j} - a_0\right)_{j \in \mathcal{J}} = \left(\sum_{i=1}^{p-1} a_{g^{-i}} \omega_p^{g^{j-i}}\right)_{j \in \mathcal{J}}$ where $\mathcal{J} = \{1, \dots, p-1\}$. Obviously, the right-hand side is the size-$(p-1)$ cyclic convolution of $\left(a_{g^{-i-1}}\right)_{i=0,\dots,p-2}$ and $\left(\omega_p^{g^i}\right)_{i=0,\dots,p-2}$.

2.6 Karatsuba

Karatsuba [KO62] computes the product $(a_0 + a_1 x)(b_0 + b_1 x)$ by evaluating at the point set $\{0, 1, \infty\}$. We compute $(a_0 + a_1 x)(b_0 + b_1 x) = a_0 b_0 + (a_0 b_1 + a_1 b_0)x + a_1 b_1 x^2$ with three multiplications $a_0 b_0$, $a_1 b_1$, and $(a_0 + a_1)(b_0 + b_1)$ by observing $a_0 b_1 + a_1 b_0 = (a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1$.

3 Implementation

This section goes through the implementation and is largely based on various ideas presented in [CCH+23]. For simplicity, we assume $R = \mathbb{F}_{4591}$.

3.1 Chosen Transformation

Let $(e_0, e_1, e_2) = (18, 34, 51)$ be the unique orthogonal idempotent elements satisfying $\forall a \in \mathbb{Z}_{102}, a \equiv (a \bmod 17)e_0 + (a \bmod 3)e_1 + (a \bmod 2)e_2 \pmod{102}$.

Conceptionally, we first apply the 3-dimensional Good–Thomas $R[x]/\langle x^{1632} - 1\rangle \cong \bar{R}[u, w, z]/\langle u^{17} - 1, w^3 - 1, z^2 - 1\rangle$ where $\bar{R} \coloneqq R[x]/\langle x^{16} - uwz\rangle$. We then apply the 3-dimensional FFT $\mathsf{NTT}_{\bar{R}_0:\omega_{17}} \otimes \mathsf{NTT}_{\bar{R}_1:\omega_3} \otimes \mathsf{NTT}_{\bar{R}_2:\omega_2}$ where $(\omega_{17}, \omega_3, \omega_2) = (\omega_{102}^{e_0}, \omega_{102}^{e_1}, \omega_{102}^{e_2})$, $\bar{R}_0 = \bar{R}[u]/\langle u^{17} - 1\rangle$, $\bar{R}_1 = \bar{R}[w]/\langle w^3 - 1\rangle$, and $\bar{R}_2 = \bar{R}[z]/\langle z^2 - 1\rangle$. For $\mathsf{NTT}_{\bar{R}_0:\omega_{17}}$, we apply Rader's FFT converting the computation into size-16 cyclic convolution. $\mathsf{NTT}_{\bar{R}_1:\omega_3}$ and $\mathsf{NTT}_{\bar{R}_2:\omega_2}$ are straightforward. The remaining problem is to multiply polynomials in $\prod_{i_0,i_1,i_2} R[x]/\langle x^{16} - \omega_{102}^{i_0e_0+i_1e_1+i_2e_2}\rangle$.

We denote η_0 the permutation map induced by the relation $x^{16} \sim uwz$, $\eta_1 = \mathsf{NTT}_{\bar{R}_0:\omega_{17}}$, $\eta_2 = \mathsf{NTT}_{\bar{R}_1:\omega_3} \otimes \mathsf{NTT}_{\bar{R}_2:\omega_2}$, and $\eta_3 = \mathrm{id}_{1632}$. The following is the chain of isomorphisms implemented.

$$
\begin{aligned}
\frac{R[x]}{\langle x^{1632} - 1\rangle} &\overset{\eta_0}{\cong} \frac{R[x, u, w, z]}{\langle x^{16} - uwz, u^{17} - 1, w^3 - 1, z^2 - 1\rangle} \\
&\overset{\eta_1 \otimes \mathrm{id}_3 \otimes \mathrm{id}_2}{\cong} \prod_{i_0} \frac{\bar{R}[u, w, z]}{\langle u - \omega_{17}^{i_0}, w^3 - 1, z^2 - 1\rangle} \\
&\overset{\mathrm{id}_{17} \otimes \eta_2}{\cong} \prod_{i_0,i_1,i_2} \frac{\bar{R}[u, w, z]}{\langle u - \omega_{17}^{i_0}, w - \omega_3^{i_1}, z - \omega_2^{i_2}\rangle} \\
&\overset{\eta_3}{\cong} \prod_{i_0,i_1,i_2} \frac{R[x]}{\langle x^{16} - \omega_{102}^{i_0e_0+i_1e_1+i_2e_2}\rangle}.
\end{aligned}
$$

In practice, we apply $(\eta_1 \otimes \mathrm{id}_3 \otimes \mathrm{id}_2) \circ \eta_0$ at the same time and omit η_3.

3.2 Small-Dimensional Polynomial Multiplications

The remaining problems are multiplying small-degree polynomials. In this work, our main problems are $R[x]/\langle x^{16} - 1\rangle$ and $R[x]/\langle x^{16} \pm \omega_{102}^{i_0e_0+i_1e_1}\rangle$. For $R[x]/\langle x^{16} - 1\rangle$, we split it into

$$
\frac{R[x]}{\langle x^{16} - 1\rangle} \cong \frac{R[x]}{\langle x - 1\rangle} \times \frac{R[x]}{\langle x + 1\rangle} \times \frac{R[x]}{\langle x^2 + 1\rangle} \times \frac{R[x]}{\langle x^4 + 1\rangle} \times \frac{R[x]}{\langle x^8 + 1\rangle}.
$$

For $R[x]/\langle x^{16} - \omega_{102}^{i_0e_0+i_1e_1}\rangle$, we split it into

$$
\frac{R[x]}{\langle x^{16} - \omega_{102}^{i_0e_0+i_1e_1}\rangle} \cong \frac{R[x]}{\left\langle x^8 - \omega_{51}^{\frac{i_0e_0+i_1e_1}{2}}\right\rangle} \times \frac{R[x]}{\left\langle x^8 + \omega_{51}^{\frac{i_0e_0+i_1e_1}{2}}\right\rangle}.
$$

Finally, we apply two layers of Karatsuba for $R[x]/\langle x^{16} + \omega_{102}^{i_0e_0+i_1e_1}\rangle$, and one layer of Karatsuba for $R[x]/\langle x^8 + 1\rangle$ and $R[x]\Big/\left\langle x^8 \pm \omega_{51}^{\frac{i_0e_0+i_1e_1}{2}}\right\rangle$.

4 Results

4.1 Benchmarking Environment

We benchmark on Intel(R) Core(TM) i7-4770K (Haswell) processor with the frequency 3.5 GHz, Intel(R) Xeon(R) CPU E3-1275 v5 (Skylake) with the frequency 3.6 GHz, and Intel(R) Core(TM) i7-10700K (Comet Lake) with the frequency 800 MHz. We compile with GCC 10.4.0 on Haswell, GCC 11.3.0 on Skylake, and GCC 10.2.1 on Comet Lake. TurboBoost and hyperthreading are disabled.

4.2 Polynomial Multiplication

We provide the performance in cycle counts of two functions `mulcore` and `polymul`. `mulcore` derives the products in $\mathbb{Z}_{4591}[x]$ with potential scaling by a predefined constant, and `polymul` additionally reduces the result to $\mathbb{Z}_{4591}[x]/\langle x^{761} - x - 1\rangle$ and mitigates the potential scaling. Compared to [BBCT22], our `mulcore` is 1.69×, 1.83×, and 1.88× faster on Haswell, Skylake, and Comet Lake. For `polymul`, our implementation is 1.77×, 1.9×, and 1.92× faster on Haswell, Skylake, and Comet Lake, respectively. We additionally vectorize reduction modulo $x^{761} - x - 1$ and obtain some improvements.

Table 1: Cycles of big-by-big polynomial multiplications for `ntrulprs761/sntrup761` on Haswell, Skylake, and Comet Lake with AVX2.

	[BBCT22]*	This work
Haswell		
`mulcore` $(\mathbb{Z}_{4591}[x])$	23 460	13 892
`polymul` $\left(\frac{\mathbb{Z}_{4591}[x]}{\langle x^{761}-x-1\rangle}\right)$	25 356	14 312
Skylake		
`mulcore` $(\mathbb{Z}_{4591}[x])$	21 402	11 682
`polymul` $\left(\frac{\mathbb{Z}_{4591}[x]}{\langle x^{761}-x-1\rangle}\right)$	23 306	12 242
Comet Lake		
`mulcore` $(\mathbb{Z}_{4591}[x])$	16 154	8 570
`polymul` $\left(\frac{\mathbb{Z}_{4591}[x]}{\langle x^{761}-x-1\rangle}\right)$	16 852	8 776

* Our own benchmarks.

5 References

[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. https://tches.iacr.org/index.php/TCHES/article/view/8733. 1

[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. 1, 4

[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. 2

[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 [NIS], 2020. https://ntruprime.cr.yp.to/. 1

[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. 1, 2, 6

[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. 2

[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. 2

[Bou89] Nicolas Bourbaki. *Algebra I*. Springer, 1989. 3

[Bru78] Georg Bruun. z-transform DFT Filters and FFT's. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):56–63, 1978. 2

[BY19] Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):340–398, 2019. https://tches.iacr.org/index.php/TCHES/article/view/8298. 1

[CCH+23] Han-Ting Chen, Yi-Hua Chung, Vincent Hwang, Chi-Ting Liu, and Bo-Yin Yang. Algorithmic Views of Vectorized Polynomial Multipliers for NTRU and NTRU Prime (Long Paper). Cryptology ePrint Archive, Paper 2023/541, 2023. https://eprint.iacr.org/2023/541. 1, 2, 4

[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. 2, 3

[FP07] Franz Franchetti and Markus Puschel. SIMD Vectorization of Non-Two-Power Sized FFTs. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 2, 2007. 4

[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. 2, 3, 4

[KO62] Anatolii Alekseevich Karatsuba and Yu P Ofman. Multiplication of many-digital numbers by automatic computers. In *Doklady Akademii Nauk*, volume 145(2), pages 293–294, 1962. 4

[Mon85] Peter L. Montgomery. Modular Multiplication Without Trial Division. *Mathematics of computation*, 44(170):519–521, 1985. 2

[NIS] NIST, the US National Institute of Standards and Technology. Post-quantum cryptography standardization project. https://csrc.nist.gov/Projects/post-quantum-cryptography. 6

[Nus80] Henri Nussbaumer. Fast Polynomial Transform Algorithms for Digital Convolution. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(2):205–215, 1980. 1

[Rad68] Charles M. Rader. Discrete fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, 56(6):1107–1108, 1968. 2, 4

[Sch77] Arnold Schönhage. Schnelle multiplikation von polynomen über körpern der charakteristik 2. *Acta Informatica*, 7(4):395–398, 1977. 1

[Sei18] Gregor Seiler. Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography. 2018. https://eprint.iacr.org/2018/039. 2, 3