Preprocess-then-NTT Technique and Its Applications to Kyber and NewHope

Shuai Zhou[1,2,3], Haiyang Xue[1,2,3], Daode Zhang[1,2,3,(✉)], Kunpeng Wang[1,2,3], Xianhui Lu[1,2,3], Bao Li[1,2,3], and Jingnan He[1,2,3]

1 School of Cyber Security, University of Chinese Academy of Sciences.
2 Data Assurances and communications Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China.
3 State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China.
{zhoushuai,zhangdaode }@iie.ac.cn

Abstract. The Number Theoretic Transform (NTT) provides efficient algorithm for multiplying large degree polynomials. It is commonly used in cryptographic schemes that are based on the hardness of the Ring Learning With Errors problem (RLWE), which is a popular basis for post-quantum key exchange, encryption and digital signature.

To apply NTT, modulus q should satisfy that $q \equiv 1 \mod 2n$, RLWE-based schemes have to choose an oversized modulus, which leads to excessive bandwidth. In this work, we present "Preprocess-then-NTT (Pt-NTT)" technique which weakens the limitation of modulus q, i.e., we only require $q \equiv 1 \mod n$ or $q \equiv 1 \mod n/2$. Based on this technique, we provide new parameter settings for Kyber and NewHope (two NIST candidates). In these new schemes, we can reduce public key size and ciphertext size at a cost of very little efficiency loss .

Keywords: NTT, Preprocess-then-NTT, Kyber, NewHope, Ring Learning With Errors, Module Learning With Errors

1 Introduction

Fast Fourier Transform (FFT) algorithms can be applied to the efficient nega-cyclic convolution of two integer sequences of length n. When the coefficients of sequence (or polynomial) are specialized to come from a finite field, the FFT is called the Number Theoretic Transform (NTT) [11] and can be used to compute polynomial multiplication efficiently over this specific finite field. For example, when polynomials come from $\mathcal{R}_q = \mathbb{Z}_q[x]/x^n + 1$, the product corresponds to a nega-cyclic convolution of the coefficient sequences. Note that $\mathbb{Z}_q$ denotes the quotient ring $\mathbb{Z}/q\mathbb{Z}$ of the rational integers ring $\mathbb{Z}$, and $n = 2^{n'-1}$ such that x^n+1 is the $2^{n'}$-th cyclotomic polynomial. In this setting, the NTT is usually computed with a special type of FFT algorithm that can be efficiently implemented when q is a prime satisfying that $q \equiv 1 \mod 2n$ [17], which in turn means that the underlying finite field contains $2n$-th roots of unity.

1.1 RLWE-based cryptography

Lattice-based cryptography has emerged as a promising candidate for public-key cryptography that is still secure after the likely advent of quantum computers. The first lattice-based encryption scheme was proposed by Ajtai and Dwork [1]. This scheme was later simplified and improved by Regev [19]. And a major achievement of Regev was the introduction of Learning With Errors problem (LWE), which was relatively simple to use in cryptographic constructions.

The LWE assumption is asymptotically at least as hard as some standard worst-case lattice problems [19]. Based on the LWE problem, Lyubashevsky et al. [18] proposed a variant of LWE over polynomial rings and showed that the variant enjoyed a worst-case hardness guarantee. The variant was defined as Ring-Learning With Errors problem (RLWE). The polynomial rings in RLWE assumption are usually defined as $\mathcal{R}_q = \mathbb{Z}_q[x]/x^n+1$ as mentioned above. RLWE-based schemes have been proposed for public-key encryption [18], digital signatures [16] and key exchange [8]. In order to compute the multiplication of polynomials efficiently, most RLWE-based schemes invoke the NTT technique [17] which requires that q is a prime and satisfies $q \equiv 1 \mod 2n$. As a result, these schemes have higher efficiency than that without applying the NTT algorithm.

RLWE-based schemes also have some drawbacks. Stebila et al. [20] reported the performance of standalone post-quantum cryptographic operations of key exchange protocols (passively secure key encapsulation mechanisms, KEMs), as well as standard cryptographic operations for comparison. As is shown in Table 1 [20], the key exchange scheme based on RLWE assumption (NewHope) has a significant increase in running time, while its bandwidth is too large comparing with ECDH nistp256. Reducing the bandwidth (even if only a few tens of bytes) makes sense in RLWE-based post-quantum cryptographic schemes, especially in some special scenario. For example, in the wireless sensor nodes, power is very crucial factor and most of the power is due to the RF transceiver module. Thus, decreasing the bandwidth by reducing the sizes of keys and ciphertexts will be hugely beneficial in the scenario.

Table 1: Performance of standalone cryptographic operations, showing mean runtime in milliseconds of standalone cryptographic operations, communication sizes (public key messages) in bytes, and claimed security level in bits.

Scheme	Alice0 (ms)	Bob (ms)	Alice1 (ms)	Communication(bytes) $A \to B$	$B \to A$	Claimed security classical	quantum
RSA 3072-bit	–	0.09	4.49	$387/0^*$	384	128	-
ECDH nistp256	0.37	0.70	0.33	32	32	128	-
NewHope	0.11	0.16	0.03	1824	2048	229	206

1.2 Our contribution

Because modulus q is required to satisfy that $2n|q-1$ in the NTT, RLWE-based schemes have to choose an oversized modulus, which leads to excessive bandwidth. To reduce the bandwidth of RLWE-based schemes in the case of using NTT, we present a method to preprocess the polynomials first by using a divide-and-conquer strategy, and then apply the NTT, which is called Preprocess-then-NTT (PtNTT). According to the times of preprocess, our PtNTT algorithm can be classified as 1-round PtNTT (1PtNTT) and 2-round PtNTT (2PtNTT).

Polynomial multiplication over a finite field is one of fundamental operations required in cryptographic schemes based on the RLWE problem, and NTT is commonly used in the RLWE-based schemes. So, our PtNTT can be applied to a large portion of RLWE-based schemes to reduce the value of modulus q, which will decrease the bandwidth.

1.2.1∼ 1PtNTT and its application to Kyber.

In 1PtNTT, we first divide the polynomial f with n coefficients into two new low-dimensional polynomials of degree $n/2$ according to the parity of index, and then apply the NTT to the two low-dimensional polynomials respectively. So, our 1PtNTT algorithm only requires that $q \equiv 1 \mod n$ instead of $q \equiv 1 \mod 2n$ in the NTT, i.e., weakens the limitation of modulus q. There exist some advantages and disadvantages of 1PtNTT compared with the NTT algorithm:

- `Advantages.` Our 1PtNTT algorithm weakens the limitation of modulus q, i.e., we require that $q \equiv 1 \mod n$ instead of $q \equiv 1 \mod 2n$ in the NTT.
- `Disadvantages.` Although our 1PtNTT algorithm is very efficient, it is still slightly less efficient than the NTT. The computational cost of 1PtNTT to compute the product of two polynomials of degree n is about 1.17 times that of the NTT algorithm.

Its application to Kyber. According to the three parameter sets for Kyber [1] [6] which are called Kyber512-CCA-KEM, Kyber768-CCA-KEM, Kyber1024-CCA-KEM, we will give a series of new parameter settings. Because the modulus q in our schemes is smaller, we call our scheme small-Kyber, i.e., small-Kyber512-CCA-KEM, small-Kyber768-CCA-KEM, and small-Kyber1024-CCA-KEM. The modulus $q = 3329$ and parameters for small-Kyber1024 has been proposed in an early work[22]. Comparing with the original Kyber schemes, there are some advantages and disadvantages of small-Kyber:

- `Advantages.` Because the mudulus q is smaller in small-Kyber, we can reduce both public key size and ciphertext size of schemes. More precisely,
  - In small-Kyber512-CCA-KEM, the public key size and the ciphertext size are 64 and 64 bytes respectively fewer than that of Kyber512-CCA-KEM;

[1] Kyber was constructed under the Module Learning With Errors (MLWE) assumption, which is a module version of the RLWE assumption.

  - In small-Kyber768-CCA-KEM, the public key size and the ciphertext size are 96 and 32 bytes respectively fewer than that of Kyber768-CCA-KEM;
  - In small-Kyber1024-CCA-KEM, the public key size and the ciphertext size are 128 and 128 bytes respectively fewer than that of Kyber1024-CCA-KEM.

  Note that small-Kyber has a similar decryption error probability and a same security level compared with Kyber. Please see more details in Tables 4, 5.
- `Disadvantages.` Although small-Kyber schemes have achieved high efficiency, they are slightly slower than the original Kyber schemes. For a worst case of three parameter sets, the cycle counts of "Key Generation", "Encapsulation" and "Decapsulation" in small-Kyber1024-CCA-KEM are 1.3296, 1.2856 and 1.4138 times that in Kyber1024-CCA-KEM. However, the purpose of decreasing the bandwidth is more meaningful than improving the efficiency of Kyber schemes. In small-Kyber1024-CCA-KEM, the running time of "Key Generation", "Encapsulation" and "Decapsulation" are 0.118 ms, 0.149 ms and 0.188 ms. While in Kyber1024-CCA-KEM, they are 0.089 ms, 0.116 ms and 0.133 ms. Note that, all results are obtained on a 3.3 GHz CPU. Please see more details in Table 6.

In short, we can use fewer bytes to store public keys and ciphertexts to reduce the bandwidth of schemes at a cost of very little loss of efficiency.

1.2.2∼ 2PtNTT and its application to NewHope.

In the NewHope [2], there are two parameter settings for n, i.e., $n = 512$ and $n = 1024$ respectively. Even though 1PtNTT technique can weaken the limitation of modulus q and requires that $q \equiv 1 \bmod n$, there is still no suitable prime modulus to satisfy the weakened requirement for two values of n, 512 and 1024, simultaneously. That's to say, we can not apply 1PtNTT technique to NewHope schemes directly. So we propose 2PtNTT technique to address this problem.

In 2PtNTT, we first divide the polynomial f with n coefficients into two new low-dimensional polynomials of degree $n/2$ according to the parity of index, and then divide each polynomial of degree $n/2$ into two polynomials of degree $n/4$ according to the parity of index. After that, we apply the NTT to the four polynomials of degree $n/4$ respectively. So, our 2PtNTT algorithm only requires that $q \equiv 1 \mod n/2$ instead of $q \equiv 1 \mod 2n$ in the NTT, i.e., further weakens the limitation of modulus q. There exist some advantages and disadvantages of 2PtNTT compared with the NTT algorithm:

- `Advantages.` Our 2PtNTT algorithm weakens the limitation of modulus q, i.e., we require that $q \equiv 1 \mod n/2$ instead of $q \equiv 1 \mod 2n$ in the NTT.
- `Disadvantages.` Although our 2PtNTT algorithm is also efficient, it is still slightly less efficient than the NTT. The computational cost of 2PtNTT to compute the product of two polynomials of degree n is 1.25 times that of the NTT algorithm.

Its application to NEWHOPE. According to the two parameter settings for NEWHOPE [2] which are called NEWHOPE512, NEWHOPE1024, we will give two new parameter settings. Because the modulus q in our schemes is smaller, we call our scheme small-NEWHOPE, i.e., small-NEWHOPE and small-NEWHOPE1024. Comparing with the original NEWHOPE schemes, there are some advantages and disadvantages of small-NEWHOPE:

- `Advantages.` Because the modulus q is smaller in small-NEWHOPE, we can reduce both public key size and ciphertext size of schemes. More precisely,
  - In small-NEWHOPE512-CPA-KEM, the public key size and the ciphertext size are64 and 64 bytes respectively fewer than that of NEWHOPE512-CPA-KEM;
  - In small-NEWHOPE512-CCA-KEM, the public key size and the ciphertext size are64 and 64 bytes respectively fewer than that of NEWHOPE512-CCA-KEM;
  - In small-NEWHOPE1024-CPA-KEM, the public key size and the ciphertext size are128 and 128 bytes respectively fewer than that of NEWHOPE-1024-CPA-KEM.
  - In small-NEWHOPE1024-CCA-KEM, the public key size and the ciphertext size are 128 and 128 bytes respectively fewer than that of NEWHOPE-1024-CCA-KEM.

  Note that small-NEWHOPE has a similar decryption error probability and a same security level compared with NEWHOPE. Please see more details in Tables 7, 8.
- `Disadvantages.` Although our small-NEWHOPE schemes have achieved high efficiency, they are slightly slower than the original NEWHOPE schemes. For a worst case of two parameter settings, the cycle counts of "Key Generation", "Encapsulation" and "Decapsulation" in small-NEWHOPE1024-CCA-KEM are 1.4747, 1.6202 and 2.3130 times that in NEWHOPE1024-CCA-KEM. However, the purpose of decreasing the bandwidth is more meaningful than improving the efficiency of NEWHOPE schemes. In small-NEWHOPE1024-CCA-KEM, the running time of "Key Generation", "Encapsulation" and "Decapsulation" are 0.082 ms, 0.133 ms and 0.049 ms. While in NEWHOPE1024-CCA-KEM, they are 0.056 ms, 0.082 ms and 0.021 ms. Note that, all results are obtained on a 3.3 GHz CPU. Please see more details in Table 9.

In brief, we can use fewer bytes to store public keys and ciphertexts to reduce the bandwidth of schemes at a cost of very little loss of efficiency.

1.3 Our technique

Because 2PtNTT is similar to 1PtNTT, in this subsection, we will only introduce one of our main techniques, 1PtNTT. Our 1PtNTT algorithm only requires that the modulus q satisfies that $q-1$ can be divided by n, i.e., $n|(q-1)$. However, in this case, if $2n \nmid (q-1)$, we can not exploit the negative wrapped convolution[17] and this is why we need to preprocess the polynomials. Similar to the process

of computing polynomial multiplication by using NTT, our 1PtNTT technique contains 2 phases: $\mathsf{1PtNTT}$ and $\mathsf{1PtNTT}^{-1}$.

In 1PtNTT, we first divide the polynomial $f(x) \in \mathbb{Z}_q[x]/(x^n+1)$ with n coefficients into two low-dimension polynomials of degree $n/2$ according to the parity of index, $f_{even}(y) \in \mathbb{Z}_q[y]/(y^{n/2}+1)$ and $f_{odd}(y) \in \mathbb{Z}_q[y]/(y^{n/2}+1)$, where f_{even} contains all the even-indexed coefficients of f and f_{odd} contains all the odd-indexed coefficients of f, and $y = x^2$. It is easy to see that $f(x) = f_{even}(x^2) + x \cdot f_{odd}(x^2)$. As a result, we can apply the NTT to the two low-degree polynomials. So we define $\mathsf{1PtNTT}(f) = (\mathsf{NTT}(f_{even}), \mathsf{NTT}(f_{odd}))$. In order to recover f from its 1PtNTT transformed representation $\widehat{f} = (\widehat{f}_{even}, \widehat{f}_{odd})$, we define $\mathsf{1PtNTT}^{-1}(\ \widehat{f}\ ) = (\mathsf{NTT}^{-1}(\widehat{f}_{even}), \mathsf{NTT}^{-1}(\widehat{f}_{odd}))$. It is very obvious that the following equation $\mathsf{1PtNTT}^{-1}(\mathsf{1PtNTT}(f)) = (f_{even}, f_{odd}) = f$ holds.

As we all know, the NTT provides an efficient algorithm for multiplying large degree polynomials. Here comes a question how can we use 1PtNTT to compute the product of two polynomials f and g? Let y denote x^2 and let $p(x) \in \mathcal{R}_q$ denote the product of $f(x)$ and $g(x)$, then

$$p_{even}(y) = f_{even}(y) \cdot g_{even}(y) + f_{odd}(y) \cdot (y \cdot g_{odd}(y)) \in \mathbb{Z}_q[y]/(y^{n/2}+1),$$
$$p_{odd}(y) = f_{odd}(y) \cdot g_{even}(y) + f_{even}(y) \cdot g_{odd}(y) \in \mathbb{Z}_q[y]/(y^{n/2}+1).$$

And $p(x) = p_{even}(x^2) + x \cdot p_{odd}(x^2) \in \mathbb{Z}_q[x]/(x^n+1)$.

If we define $\overrightarrow{g_{odd}}$ as $\big(-g_{odd}[\frac{n}{2}-1], g_{odd}[0], g_{odd}[1], \cdots, g_{odd}[\frac{n}{2}-2]\big)$, and a bow-tie multiplication as

$$\mathsf{1PtNTT}(f) \bowtie \mathsf{1PtNTT}(g) = (\mathsf{NTT}(f_{even}) \circ \mathsf{NTT}(g_{even}) + \mathsf{NTT}(f_{odd}) \circ \mathsf{NTT}(\overrightarrow{g_{odd}}),$$
$$\mathsf{NTT}(f_{odd}) \circ \mathsf{NTT}(g_{even}) + \mathsf{NTT}(f_{even}) \circ \mathsf{NTT}(g_{odd})),$$

where $\circ$ denotes coefficient-wise multiplication. Then the following equation $p = \mathsf{1PtNTT}^{-1}(\mathsf{1PtNTT}(f) \bowtie \mathsf{1PtNTT}(g))$ holds, which is very similar to $p = \mathsf{NTT}^{-1}(\mathsf{NTT}(f) \circ \mathsf{NTT}(g))$ in the NTT algorithm.

1.4 Related Work

Inspired by Kyber [6], D'Anvers et al. [12, 13] also proposed a family of cryptographic primitives, i.e., Saber which includes three IND-CCA secure KEMs LightSaber-KEM, Saber-KEM and FireSaber-KEM. Saber-KEMs have similar public key and ciphertext sizes respectively compared with our small-Kyber-KEMs. Moreover, Saber-KEMs have better efficiency than Kyber-KEMs, so it is easy to get a conclusion that Saber-KEMs are more efficient than our small-Kyber-KEMs. We emphasize that, there exist two main differences between Saber and our small-Kyber.

- In order to get rid of the constraint to modulus q caused by applying the NTT algorithm, D'Anvers et al. [12, 13] invoked the Karatsuba polynomial multiplication method which does not require any special modulus. As a result, all moduli in Saber schemes are powers of 2, i.e., $q = 2^{13}$, while

all moduli in our small-KYBER schemes are primes, i.e., $q = 3329$, which *might* increase confidence in security. It is more popular for constructions of schemes using prime modulus than non-prime modulus.
- Saber-KEMs rely on the hardness of the Module Learning With Rounding (MLWR) problem, while our small-KYBER-KEMs are constructed under the Module Learning With errors problem. As a result, Saber does not require sampling of error polynomials, thus saving in computation time.

1.5 Outline

The remainder of the paper is organized as follows. In Section 2 we review the necessary background. In Section 3 we present our Preprocess-then-NTT technique and describe its complexity. The applications of PtNTT to KYBER and NEWHOPE are given in Section 4, followed by the implementation and performance of new schemes. In Section 5, we give the conclusion.

2 Preliminaries

Polynomial rings and vectors. Let $\mathbb{Z}$ be the ring of rational integers. Let $\mathbb{Z}_q$ denote the quotient ring $\mathbb{Z}/q\mathbb{Z}$, for an integer $q \geq 1$. We denote by $\mathcal{R}$ the ring $\mathbb{Z}[x]/x^n + 1$ and by $\mathcal{R}_q$ the ring $\mathbb{Z}_q[x]/x^n + 1$, where $n = 2^{n'-1}$ such that $x^n + 1$ is the $2^{n'}$-th cyclotomic polynomial. Throughout this paper, the values of n, n' are $256, 9$, respectively. Regular font letters denote element in $\mathcal{R}_q$ and bold lower-case letters represent vectors with coefficients in $\mathcal{R}_q$. By default, all vectors will be column vectors. Bold upper-case letters are matrices.

For an element $a \in \mathcal{R}_q$, we write $a = \sum_{i=0}^{n-1} a_i x^i$, $a_i \in \mathbb{Z}_q$. We also use the same symbol a to denote the coefficient vector $a = (a_0, \cdots, a_{n-1})$. Let $a \circ b$ denote pointwise or coefficient-wise multiplication of $a, b \in \mathcal{R}_q$. For a vector $\mathbf{a}$ (or matrix $\mathbf{A}$), we denote by $\mathbf{a}^\top$ (or $\mathbf{A}^\top$) its transpose.

Sets and distributions. For a set S, we write $s \xleftarrow{\$} S$ to denote s is chosen uniformly at random form S. In case S is a probability distribution over $\mathcal{R}$, then $x \xleftarrow{\$} S$ means the sampling of $x \in \mathcal{R}$ according to S. For a probabilistic algorithm $\mathcal{A}$ we denote by $y \leftarrow \mathcal{A}$ that the output of $\mathcal{A}$ is assigned to y and that $\mathcal{A}$ is running with randomly chosen coins. We define the centered binomial distribution ψ_η for some positive integer η as follows:

$$(a_1, \cdots, a_\eta, b_1, \cdots, b_\eta) \xleftarrow{\$} \{0,1\}^{2\eta} \text{ and output } \Sigma_{i=1}^{\eta}(a_i - b_i).$$

The distribution ψ_η is centered (its mean is 0), has variance $\eta/2$ and gives a standard deviation of $\sqrt{\eta/2}$. The function Sam is an extendable function. If we would like Sam to take as input x and then produce a value y that is distributed according to distribution $\mathcal{S}$, we write $y \sim \mathcal{S} = \mathsf{Sam}(x)$.

Compression and Decompression. We define a function $\mathsf{Compress}_q(x, d)$ that takes an element $x \in \mathbb{Z}_q$ and outputs an integer in $\{0, 1, \cdots, 2^d - 1\}$, where

$d < \lceil \log q \rceil$. We furthermore define a function $\mathsf{Decompress}_q(x, d)$, such that $x' = \mathsf{Decompress}_q(\mathsf{Compress}_q(x, d), d)$ is an element close to x. The two functions are defined as:

$$\mathsf{Compress}_q(x, d) = \lceil (2^d/q) \cdot x \rfloor \mod 2^d;$$
$$\mathsf{Decompress}_q(x, d) = \lceil (q/2^d) \cdot x \rfloor.$$

2.1 Ring LWE and Module LWE problems

The Learning with Errors (LWE) problem was popularized by Regev [19] who proved that, solving a random LWE instance is as hard as solving worst-case instances of certain lattice problems under a quantum reduction. Later on, Lyubashevsky, Peikert and Regev [18] proposed a variant of the LWE problem–the Ring-LWE problem which relies on module lattices, and its hardness can be related to the worst case hardness of finding short vectors in ideal lattices [21, 18]. Recently, Langlois and Stehlé [15] proposed a module version of Ring-LWE, Module-LWE.

The Ring Learning with Errors problem, decisional version. The decisional version of the Ring Learning with Errors problem, $\text{DRLWE}_{m,q,\chi}$, with m unknowns, $m \geq 1$ samples, modulo q and error distribution χ, is defined as follows: for a uniform random secret $s \in \mathcal{R}_q$, and given m samples either all of the form $(a, b = a \cdot s + e \mod q)$ where the coefficients of e are independently sampled following the distribution χ (i.e., $e_i \xleftarrow{\$} \chi$), or from the uniform distribution $(a, b) \in \mathcal{R}_q \times \mathcal{R}_q$, decide whether the samples come from the former or the latter case.

In fact, we will use a variant of the above problem, where the secret $\mathbf{s}$ are chosen from the same distribution as the error e. This variant was proven to be equivalent to the original problem by Applebaum et al. in [5].

The Module Learning with Errors problem, decisional version. The decisional version of the Module Learning with Errors problem, $\text{DMLWE}_{m,q,\chi}$, consists in distinguishing m samples either all of the form $(\mathbf{a}, b = \mathbf{a}^\top \mathbf{s} + e)$ with $\mathbf{s} \xleftarrow{\$} \chi^k$ common to all samples and $e \xleftarrow{\$} \chi$ fresh for every sample, or from the uniform distribution $(\mathbf{a}, b) \in \mathcal{R}_q^k \times \mathcal{R}_q$, decide whether the samples come from the former or the latter case.

2.2 Number-theoretic transform

There exist many efficient algorithms in the literature to compute the multiplication of two polynomials and a survey which introduces fast multiplication algorithms can be found in [7]. In this subsection, we recall the Number Theoretic Transform (NTT) [11] which is a specialized version of the Fast Fourier Transform (FFT) where the roots of unity are taken from a finite ring instead of the complex number.

Let n be a power of 2 and q be a prime satisfying that $q \equiv 1 \mod 2n$. Let ω be an n-th primitive root of unity in $\mathbb{Z}_q$, i.e., $\omega^n \equiv 1 \mod q$. For an

element $f \in \mathcal{R}_q$, the forward transformation $\widehat{f} = \mathsf{NTT}(f)$ is given by $\widehat{f}_i = \sum_{j=0}^{n-1} f_j \cdot \omega^{ij} \mod q$ and the inverse transformation $f = \mathsf{NTT}^{-1}(\widehat{f})$ is defined by $f_i = n^{-1} \sum_{j=0}^{n-1} \widehat{f}_j \cdot \omega^{-ij} \mod q$, where $i = 0, \cdots, n-1$. The following equation $\mathsf{NTT}^{-1}(\mathsf{NTT}(f)) = f$ holds.

Because applying the above NTT transform provides a cyclic convolution, computing $p = f \cdot g \mod x^n+1$ with two polynomials f, g would require applying the NTT of length $2n$ and thus should append n zeros to each input. This effectively doubles the length of the inputs and also requires the computation of an explicit reduction modulo x^n+1. In order to avoid this issue, Lyubashevsky et al. [17] introduced the negative wrapped convolution: let γ be a $2n$-th primitive root of unity such that $\gamma = \sqrt{\omega} \mod q$. Define $\widetilde{f} = (f_0, \gamma f_1, \cdots, \gamma^{n-1} f_{n-1})$ and $\widetilde{g} = (g_0, \gamma g_1, \cdots, \gamma^{n-1} g_{n-1})$, then the negative wrapped convolution of f, g is given by $p = (1, \gamma, \cdots, \gamma^{n-1}) \circ \mathsf{NTT}^{-1}(\mathsf{NTT}(\widetilde{f}) \circ \mathsf{NTT}(\widetilde{g}))$. This operation satisfies $p = f \cdot g$ in $\mathcal{R}_q$ and implicitly includes the reduction modulo $x^n + 1$ without increasing the length of the inputs. More precisely, for a polynomial $f \in \mathcal{R}_q$, we define its forward transformation $\widehat{f} = \mathsf{NTT}(f)$ with $\widehat{f}_i = \sum_{j=0}^{n-1} \gamma^j f_j \cdot \omega^{ij} \mod q$ and its inverse transformation $f = \mathsf{NTT}^{-1}(\widehat{f})$ with $f_i = n^{-1}\gamma^{-i} \sum_{j=0}^{n-1} \widehat{f}_j \cdot \omega^{-ij} \mod q$. Using NTT and NTT^{-1}, we can compute the product $f \cdot g$ very efficiently as $p = \mathsf{NTT}^{-1}(\mathsf{NTT}(f) \circ \mathsf{NTT}(g))$.

The computational cost of a forward NTT transformation NTT is determined by a function $T(n) = n \log n$ in [10]. By comparing the definitions of NTT and NTT^{-1}, we see that by modifying the NTT algorithm to switch the roles of f and $\widehat{f}$, replace ω by ω^{-1}, and divide each element of the result by n, we can compute the inverse transformation NTT^{-1}. So the computational cost of an inverse NTT transformation NTT^{-1} is same as that of its forward NTT transformation NTT, i.e., $T(n) = n \log n$. If we use NTT to compute the multiplication of two polynomials, the total computation includes two forward NTT transformations, one point-wise multiplication of degree bounded by n and one inverse NTT transformation. Obviously, the computational cost of multiplication by using NTT is $T_1(n) = 3n \log n + n$.

3 1-Round Preprocess-then-NTT (1PtNTT)

In 1PtNTT algorithm, we only require that modulus q satisfies that $q - 1$ can be divided by n, i.e., $n \mid (q-1)$. However, in this case, we can not exploit the negative wrapped convolution [17] if $2n \nmid (q-1)$, because there does not exist any $2n$-th root of unity in $\mathbb{Z}_q$.

Our 1-round preprocess-then-NTT technique employs a divide-and-conquer strategy, using the even-indexed and odd-indexed coefficients of $f(x) \in \mathbb{Z}_q[x]/(x^n+1)$ separately to define two new polynomials $f_{even}(y)$ and $f_{odd}(y)$ whose degrees are bounded by $\frac{n}{2}$:

$$f_{even}(y) = f_0 + f_2 \cdot y + f_4 \cdot y^2 + \cdots + f_{n-2} \cdot y^{n/2-1} \in \mathbb{Z}_q[y]/(y^{n/2}+1),$$
$$f_{odd}(y) = f_1 + f_3 \cdot y + f_5 \cdot y^2 + \cdots + f_{n-1} \cdot y^{n/2-1} \in \mathbb{Z}_q[y]/(y^{n/2}+1).$$

It follows that

$$f(x) = f_{even}(x^2) + x \cdot f_{odd}(x^2) \in \mathbb{Z}_q[x]/(x^n + 1). \tag{1}$$

It is easy to see that f_{even} contains all the even-indexed coefficients of f and f_{odd} contains all the odd-indexed coefficients of f.

If we denote x^2 by y, the two polynomials of degree $n/2$, $f_{even}(y)$ and $f_{odd}(y)$, are both in $\mathbb{Z}_q[y]/(y^{n/2}+1)$, then we can apply the NTT to get their transformed representations, i.e., $\mathsf{NTT}(f_{even})$ and $\mathsf{NTT}(f_{odd})$. Define

$$\begin{aligned} \mathsf{1PtNTT}(f) = \widehat{f} &= (\mathsf{NTT}(f_{even}), \mathsf{NTT}(f_{odd}))\,, \\ \mathsf{1PtNTT}^{-1}(\widehat{f}) &= \left(\mathsf{NTT}^{-1}(\widehat{f}_{even}), \mathsf{NTT}^{-1}(\widehat{f}_{odd})\right). \end{aligned}$$

Then the following equation $\mathsf{1PtNTT}^{-1}(\mathsf{1PtNTT}(f)) = (f_{even}, f_{odd}) = f$ holds.

As mentioned above, we first divide $f(x)$ to define two new polynomials of degree $n/2$ and then apply the NTT. We call our technique "1-round Preprocess-then-NTT" (1PtNTT, for short).

3.1 How to compute the product of two polynomials f and g?

As we all know, the NTT provides an efficient algorithm for multiplying large degree polynomials. It is commonly used in cryptographic schemes that are based on the hardness of the RLWE problem to efficiently implement modular polynomial multiplication. Here comes a question how can we use 1PtNTT to compute the product of two polynomials f and g?

As the same way in Equation 1, for $g(x) \in \mathbb{Z}_q[x]/(x^n + 1)$, we can use the coefficients of $g(x)$ separately to define two new polynomials $g_{even}(y), g_{odd}(y) \in \mathbb{Z}_q[y]/(y^{n/2} + 1)$ satisfying the following equation

$$g(x) = g_{even}(x^2) + x \cdot g_{odd}(x^2) \in \mathbb{Z}_q[x]/(x^n + 1). \tag{2}$$

Let $p(x) \in \mathbb{Z}_q[x]/(x^n + 1)$ denote the product of $f(x)$ and $g(x)$ and let

$$\begin{aligned} p_{even}(y) &= f_{even}(y) \cdot g_{even}(y) + f_{odd}(y) \cdot (y \cdot g_{odd}(y)) \in \mathbb{Z}_q[y]/(y^{n/2} + 1), \\ p_{odd}(y) &= f_{odd}(y) \cdot g_{even}(y) + f_{even}(y) \cdot g_{odd}(y) \in \mathbb{Z}_q[y]/(y^{n/2} + 1). \end{aligned}$$

Then, according to Equations 1 and 2, the following equation $p(x) = p_{even}(x^2) + x \cdot p_{odd}(x^2) \in \mathbb{Z}_q[x]/(x^n + 1)$ holds.

An anticirculant vector of g_{odd} is defined by the following Toeplitz vector:

$$\overrightarrow{g_{odd}} := \left(-g_{odd}[\frac{n}{2} - 1], g_{odd}[0], g_{odd}[1], \cdots, g_{odd}[\frac{n}{2} - 2]\right) \in \mathbb{Z}_q^{n/2},$$

which denotes $y \cdot g_{odd}(y) \in \mathbb{Z}_q[y]/(y^{n/2} + 1)$. Using $\mathsf{1PtNTT}$ and $\mathsf{1PtNTT}^{-1}$ we can compute the product p of two elements $f, g \in \mathcal{R}_q$ very efficiently through

the following equation $\mathsf{1PtNTT}^{-1}(\mathsf{1PtNTT}(f) \bowtie \mathsf{1PtNTT}(g))$, where $\bowtie$ denotes bow-tie multiplication defined as following:

$$\begin{aligned}
&\mathsf{1PtNTT}(f) \bowtie \mathsf{1PtNTT}(g) \\
&= (\mathsf{NTT}(f_{even}), \mathsf{NTT}(f_{odd})) \bowtie (\mathsf{NTT}(g_{even}), \mathsf{NTT}(g_{odd}), \mathsf{NTT}(\overrightarrow{g_{odd}})) \\
&= \left(\widehat{f}_{even}, \widehat{f}_{odd}\right) \bowtie (\widehat{g}_{even}, \widehat{g}_{odd}, \mathsf{NTT}(\overrightarrow{g_{odd}})) \\
&= \left(\widehat{f}_{even} \circ \widehat{g}_{even} + \widehat{f}_{odd} \circ \mathsf{NTT}(\overrightarrow{g_{odd}}), \widehat{f}_{odd} \circ \widehat{g}_{even} + \widehat{f}_{even} \circ \widehat{g}_{odd}\right).
\end{aligned}$$

3.2 Complexity of 1PtNTT and its comparison with NTT

In this subsection, we first analyse the theoretical complexity of 1PtNTT algorithm. Then we present some implementation results for different parameters to show the performance of 1PtNTT algorithm and its comparison with NTT .

The complexity of 1PtNTT. As for 1PtNTT, its forward transformation $\mathsf{1PtNTT}$ embeds two forward NTT transformations of two different polynomials of degree $n/2$. As a result, the computational complexity of a $\mathsf{1PtNTT}$ is bounded by $T_2(n) = n\log(n/2)$. In a similar way, we can also get a conclusion that the complexity of an inverse transformation $\mathsf{1PtNTT}^{-1}$ is $T_3(n) = n\log(n/2)$. In order to show the difference between these two algorithms, we present the ratios of the time cost of 1PtNTT to that of NTT as follows:

$$ratio_{\mathrm{1ptntt/ntt}} = \frac{\log n - 1}{\log n} \quad \text{and} \quad ratio_{\mathrm{1ptntt^{-1}/ntt^{-1}}} = \frac{\log n - 1}{\log n}.$$

Next, we analyse the complexity of computing two polynomials' product by using 1PtNTT algorithm. According to the computation rule of 1PtNTT, there exist two forward 1PtNTT transformations (one includes two forward NTT transformations , the other embeds three forward NTT transformations), four point-wise multiplications of two polynomials of degree bounded by $n/2$, and one inverse 1PtNTT transformation. As a result, the computational cost of computing product of two polynomials by using 1PtNTT is

$$T_4(n) = (2+3) \cdot \frac{n}{2}\log\frac{n}{2} + 2n + T_3(n) = \frac{7n}{2}\log\frac{n}{2} + 2n,$$

which is $ratio_1 = \dfrac{7\log n - 3}{6\log n + 2}$ times that using NTT.

Comparison of 1PtNTT and NTT. Although 1PtNTT can use some parameters that are not suitable for NTT, we analyse and compare the computational cost of 1PtNTT and NTT for the same parameters, so that we can make it easy to demonstrate the efficiency of 1PtNTT. In our implementation, we specify the details of the two methods for $(n, q) \in \{(256, 7681), (512, 12289), (1024, 12289)\}$ which are used in [3, 2, 4, 9, 6]. The results are reported in Table 2, and were obtained by running the implementation on a 3.30GHZ Inter Core i5-6600 processor

Table 2: Results of our C implementations of 1PtNTT on a 3.30GHz Inter Core i5-6600 processor with Turbo Boost and Hyperthreading disabled. Results are compared with the implementation of the NTT.

Operation	n=256, q=7681	n=512, q=12289	n=1024, q=12289
1PtNTT	13161	21523	47436
NTT	14056	24057	52034
Experimental-ratio	0.9363	0.8947	0.9116
Theoretical-ratio	0.8750	0.8889	0.9000
$1PtNTT^{-1}$	10940	23038	50512
NTT^{-1}	11845	25091	55075
Experimental-ratio	0.9236	0.9182	0.9171
Theoretical-ratio	0.8750	0.8889	0.9000
multiplication by using 1PtNTT	51213	90116	197427
multiplication by using NTT	42959	81368	180347
Experimental-ratio	1.1921	1.1075	1.0947
Theoretical-ratio	1.0600	1.0714	1.0806

with Turbo Boost and Hyperthreading disabled. We compiled our C implementation with gcc-5.4.0 and flags **-O3 -fomit-frame-pointer -march=native**. For all other routines we report the average of 10000 runs. We denote the ratio of theoretical computational cost of PtNTT operations to that of NTT operations by "Theoretical-ratio". And "Experiment-ratio" represents the ratio of practical cycle counts of 1PtNTT to that of NTT.

4 2-Round Preprocess-then-NTT (2PtNTT)

In 2PtNTT algorithm, we only require that modulus q satisfies that $q-1$ can be divided by $\frac{n}{2}$, i.e., $\frac{n}{2} \mid (q-1)$. However, in this case, we can not exploit the negative wrapped convolution [17] if $2n \nmid (q-1)$, because there does not exist any $2n$-th root of unity in $\mathbb{Z}_q$.

Based on the first round preprocess, we use the even-indexed and odd-indexed coefficients of $f(x) \in \mathbb{Z}_q[x]/(x^n+1)$ separately to define two new polynomials $f_{even}(y)$ and $f_{odd}(y)$ whose degrees are bounded by $\frac{n}{2}$. Then, by using the same preprocess again to $f_{even}(y)$ and $f_{odd}(y)$, we can define four polynomials $f_{ee}(z)$, $f_{eo}(z)$, $f_{oe}(z)$ and $f_{oo}(z)$ of degree-bound $\frac{n}{4}$. In fact, $f_{ee}(z)$ and $f_{eo}(z)$ contains all the coefficients f_i of f satisfying that $i \equiv 0 \mod 4$ and $i \equiv 2 \mod 4$, respectively. $f_{oe}(z)$ and $f_{oo}(z)$ contains all the coefficients f_i of f satisfying that $i \equiv 1 \mod 4$ and $i \equiv 3 \mod 4$, respectively. More precisely,

$$
\begin{aligned}
f_{ee}(z) &= f_0 + f_4 \cdot z + f_8 \cdot z^2 + \cdots + f_{n-4} \cdot z^{n/4} \in \mathbb{Z}_q[z]/(z^{n/4}+1),\\
f_{eo}(z) &= f_2 + f_6 \cdot z + f_{10} \cdot z^2 + \cdots + f_{n-2} \cdot z^{n/4} \in \mathbb{Z}_q[z]/(z^{n/4}+1),\\
f_{oe}(y) &= f_1 + f_5 \cdot z + f_9 \cdot z^2 + \cdots + f_{n-3} \cdot z^{n/4} \in \mathbb{Z}_q[z]/(z^{n/4}+1),\\
f_{oo}(y) &= f_3 + f_7 \cdot z + f_{11} \cdot z^2 + \cdots + f_{n-1} \cdot z^{n/4} \in \mathbb{Z}_q[z]/(z^{n/4}+1).
\end{aligned}
$$

It follows that

$$f(x) = f_{ee}(x^4) + x \cdot f_{oe}(x^4) + x^2 \cdot f_{eo}(x^4) + x^3 \cdot f_{oo}(x^4) \in \mathbb{Z}_q[x]/(x^n+1). \quad (3)$$

Note that the four polynomials of degree $n/4$, $f_{ee}(z)$, $f_{eo}(z)$, $f_{oe}(z)$ and $f_{oo}(z)$, are all in $\mathbb{Z}_q[z]/(z^{n/4}+1)$, then we can apply the NTT to get their transformed representations. Define

$$\begin{aligned} \mathsf{2PtNTT}(f) = \widehat{f} &= (\mathsf{NTT}(f_{ee}), \mathsf{NTT}(f_{oe}), \mathsf{NTT}(f_{eo}), \mathsf{NTT}(f_{oo})), \\ \mathsf{2PtNTT}^{-1}(\widehat{f}) &= \left(\mathsf{NTT}^{-1}(\widehat{f}_{ee}), \mathsf{NTT}^{-1}(\widehat{f}_{oe}), \mathsf{NTT}^{-1}(\widehat{f}_{eo}), \mathsf{NTT}^{-1}(\widehat{f}_{oo})\right). \end{aligned}$$

Then the following equation $\mathsf{2PtNTT}^{-1}(\mathsf{2PtNTT}(f)) = (f_{ee}, f_{oe}, f_{eo}, f_{oo}) = f$ holds.

4.1 How to compute the product of two polynomials f and g?

Here comes a question how can we use 2PtNTT to compute the product of two polynomials f and g?

As the same way in Equation 3, for $g(x) \in \mathbb{Z}_q[x]/(x^n+1)$, we can use the coefficients of $g(x)$ separately to define four new polynomials of degree $n/4$, i.e., $g_{ee}(z)$, $g_{eo}(z)$, $g_{oe}(z), g_{oo}(z) \in \mathbb{Z}_q[z]/(z^{n/4}+1)$ satisfying the following equation

$$g(x) = g_{ee}(x^4) + x \cdot g_{oe}(x^4) + x^2 \cdot g_{eo}(x^4) + x^3 \cdot g_{oo}(x^4) \in \mathbb{Z}_q[x]/(x^n+1). \quad (4)$$

Let $p(x) \in \mathbb{Z}_q[x]/(x^n+1)$ denote the product of $f(x)$ and $g(x)$ and let

$$\begin{aligned} p_{ee}(z) =& f_{ee}(z) \cdot g_{ee}(z) + f_{oe}(z) \cdot (z \cdot g_{oo}(z)) + f_{eo}(z) \cdot (z \cdot g_{eo}(z)) + f_{oo}(z) \cdot (z \cdot g_{oe}(z)), \\ p_{oe}(z) =& f_{ee}(z) \cdot g_{oe}(z) + f_{oe}(z) \cdot g_{ee}(z) + f_{eo}(z) \cdot (z \cdot g_{oo}(z)) + f_{oo}(z) \cdot (z \cdot g_{eo}(z)), \\ p_{eo}(z) =& f_{ee}(z) \cdot g_{eo}(z) + f_{oe}(z) \cdot g_{oe}(z) + f_{eo}(z) \cdot g_{ee}(z) + f_{oo}(z) \cdot (z \cdot g_{oo}(z)), \\ p_{oo}(z) =& f_{ee}(z) \cdot g_{oo}(z) + f_{oe}(z) \cdot g_{eo}(z) + f_{eo}(z) \cdot g_{oe}(z) + f_{oo}(z) \cdot g_{ee}(z). \end{aligned}$$

Note that all of $p_{ee}(z), p_{oe}(z), p_{eo}(z)$ and $p_{oo}(z)$ belong to $\mathbb{Z}_q[z]/(z^{n/4}+1)$. Then, according to Equations 3 and 4, the following equation $p(x) = p_{ee}(x^4) + x \cdot p_{oe}(x^4) + x^2 \cdot p_{eo}(x^4) + x^3 \cdot p_{oo}(x^4) \in \mathbb{Z}_q[x]/(x^n+1)$ holds.

An anticirculant vector of g_{ee} is defined by the following Toeplitz vector:

$$\overrightarrow{g_{ee}} := \left(-g_{odd}[\frac{n}{4}-1], g_{odd}[0], g_{odd}[1], \cdots, g_{odd}[\frac{n}{4}-2]\right) \in \mathbb{Z}_q^{n/4},$$

which denotes $z \cdot g_{odd}(z) \in \mathbb{Z}_q[z]/(z^{n/4}+1)$. Using $\mathsf{2PtNTT}$ and $\mathsf{2PtNTT}^{-1}$ we can compute the product p of two elements $f, g \in \mathcal{R}_q$ very efficiently through the following equation $\mathsf{2PtNTT}^{-1}(\mathsf{2PtNTT}(f) \bowtie \mathsf{2PtNTT}(g))$, where $\bowtie$ denotes

bow-tie multiplication defined as following:

$$
\begin{aligned}
&\mathsf{2PtNTT}(f) \bowtie \mathsf{2PtNTT}(g) \\
=&(\mathsf{NTT}(f_{ee}), \mathsf{NTT}(f_{oe}), \mathsf{NTT}(f_{eo}), \mathsf{NTT}(f_{oo})) \bowtie \\
&(\mathsf{NTT}(g_{ee}), \mathsf{NTT}(g_{oe}), \mathsf{NTT}(g_{eo}), \mathsf{NTT}(g_{oo}), \mathsf{NTT}(\overrightarrow{g_{oe}}), \mathsf{NTT}(\overrightarrow{g_{eo}}), \mathsf{NTT}(\overrightarrow{g_{oo}})) \\
=&\left(\widehat{f}_{ee}, \widehat{f}_{oe}, \widehat{f}_{eo}, \widehat{f}_{oo}\right) \bowtie (\widehat{g}_{ee}, \widehat{g}_{oe}, \widehat{g}_{eo}, \widehat{g}_{oo}, \mathsf{NTT}(\overrightarrow{g_{oe}}), \mathsf{NTT}(\overrightarrow{g_{eo}}), \mathsf{NTT}(\overrightarrow{g_{oo}})) \\
=&(\widehat{f}_{ee} \circ \widehat{g}_{ee} + \widehat{f}_{oe} \circ \mathsf{NTT}(\overrightarrow{g_{oo}}) + \widehat{f}_{eo} \circ \mathsf{NTT}(\overrightarrow{g_{eo}}) + \widehat{f}_{oo} \circ \mathsf{NTT}(\overrightarrow{g_{oe}}), \\
&\widehat{f}_{ee} \circ \widehat{g}_{oe} + \widehat{f}_{oe} \circ \widehat{g}_{ee} + \widehat{f}_{eo} \circ \mathsf{NTT}(\overrightarrow{g_{oo}}) + \widehat{f}_{oo} \circ \mathsf{NTT}(\overrightarrow{g_{eo}}), \\
&\widehat{f}_{ee} \circ \widehat{g}_{eo} + \widehat{f}_{oe} \circ \widehat{g}_{oe} + \widehat{f}_{eo} \circ \widehat{g}_{ee} + \widehat{f}_{oo} \circ \mathsf{NTT}(\overrightarrow{g_{oo}}), \\
&\widehat{f}_{ee} \circ \widehat{g}_{oo} + \widehat{f}_{oe} \circ \widehat{g}_{eo} + \widehat{f}_{eo} \circ \widehat{g}_{oe} + \widehat{f}_{oo} \circ \widehat{g}_{ee}).
\end{aligned}
$$

4.2 Complexity of 2PtNTT and its comparison with NTT

In this subsection, we first analyse the theoretical complexity of 2PtNTT algorithm. Then we present some implementation results for different parameters to show the performance of 2PtNTT algorithm and its comparison with NTT .

The complexity of 2PtNTT. As for 2PtNTT, its forward transformation PtNTT embeds two forward NTT transformations of four different polynomials of degree $n/4$. As a result, the computational complexity of a $\mathsf{2PtNTT}$ is bounded by $T_5(n) = n\log(n/4)$. In a similar way, we can also get a conclusion that the complexity of an inverse transformation PtNTT^{-1} is $T_6(n) = n\log(n/4)$. In order to show the difference between these two algorithms, we present the ratios of the time cost of PtNTT to that of NTT as follows:

$$ratio_{2\text{ptntt}/\text{ntt}} = \frac{\log n - 2}{\log n} \quad \text{and} \quad ratio_{2\text{ptntt}^{-1}/\text{ntt}^{-1}} = \frac{\log n - 2}{\log n}.$$

Next, we analyse the computational cost of multiplication by using 2PtNTT algorithm. As mentioned above, the computational cost of multiplication by using NTT is $T_1(n) = 3n\log n + n$. According to the computation rule of 2PtNTT, there exist two forward 2PtNTT transformations (one includes 4 forward NTT transformations , the other embeds 7 forward NTT transformations), 16 point-wise multiplications of two polynomials of degree bounded by $n/4$, and one inverse 2PtNTT transformation. As a result, the time cost of computing the product of two polynomials by using 2PtNTT is

$$T_7(n) = (4+7)\cdot\frac{n}{4}\log\frac{n}{4} + 4n + T_6(n) = \frac{15n}{4}\log\frac{n}{4} + 4n,$$

which is $ratio_2 = \dfrac{15\log n - 14}{12\log n + 4}$ times that using NTT.

Comparison of 2PtNTT and NTT. Although 2PtNTT can use some parameters that are not suitable for NTT, we analyse and compare the computational

Table 3: Results of our C implementations of 2PtNTT on a 3.30GHz Inter Core i5-6600 processor with Turbo Boost and Hyperthreading disabled. Results are compared with the implementation of the NTT.

Operation	n=256, q=7681	n=512, q=12289	n=1024, q=12289
2PtNTT	10072	17384	38092
NTT	13621	20294	45176
Experimental-ratio	0.7394	0.8566	0.8432
Theoretical-ratio	0.7500	0.7778	0.8000
$2\mathsf{PtNTT}^{-1}$	8728	17374	38840
NTT^{-1}	10232	21858	47790
Experimental-ratio	0.8530	0.7949	0.8127
Theoretical-ratio	0.7500	0.7778	0.8000
multiplication by using 2PtNTT	46356	83648	180252
multiplication by using NTT	37046	69048	152722
Experimental-ratio	1.2513	1.2028	1.1803
Theoretical-ratio	1.0600	1.0804	1.0968

cost of 2PtNTT and NTT for the same parameters, so that we can make it easy to demonstrate the efficiency of 2PtNTT. In our implementation, we specify the details of the two methods for $(n, q) \in \{(256, 7681), (512, 12289), (1024, 12289)\}$ which are used in [3, 2, 4, 9, 6]. The results are reported in Table 3, and were obtained by running the implementation on a 3.30GHZ Inter Core i5-6600 processor with Turbo Boost and Hyperthreading disabled. We compiled our C implementation with gcc-5.4.0 and flags **-O3 -fomit-frame-pointer -march=native**. For all other routines we report the average of 10000 runs. We denote the ratio of theoretical computational cost of 2PtNTT operations to that of NTT operations by "Theoretical-ratio". And "Experiment-ratio" represents the ratio of practical cycle counts of 2PtNTT to that of NTT.

5 Application of 1PtNTT to Kyber

Recently, Avanzi et al. [6] submitted a suite of public-key encapsulation mechanisms denoted as Kyber to NIST as a candidate of the standard of post-quantum cryptography, based on the conjectured quantum hardness of the MLWE problem. The Kyber cryptosystem is based on a variant of their previously proposed Kyber [9] scheme which is a semantically secure public-key encryption (PKE) scheme with respect to adaptive chosen plaintext attacks (CPA) (see **Appendix**). Morever, In **Appendix**, we will present a brief description of Kyber-CPA-PKE, where Kyber-CPA-PKE denotes the IND-CPA secure public key encryption scheme of Kyber.

5.1 Small-Kyber parameter sets

In [6], Avanzi et al. defined three parameter sets for Kyber, which they call these schemes Kyber512, Kyber768, Kyber1024. According to their three Kyber

schemes, we will give new parameter setting. As shown in Table 4, the modulus q in our schemes is smaller, so we call our scheme small-Kyber, i.e., small-Kyber512, small-Kyber768, small-Kyber1024. Note that Table 4 also lists the derived parameter δ, which is the probability that the decryption of a valid Kyber-CPA-PKE ciphertext fails.

The parameters were obtained via the following approaching: 1) n is set to 256 because the goal is to encapsulate 256-bit symmetric keys. 2) q is set to the smallest prime satisfying $n|(q-1)$, which is required to enable the 1PtNTT-based multiplication. 3) k is selected to fix the lattice dimension as a multiple of n. 4) The remaining parameters η, d_u, d_v, d_t were chosen to balance between security, public-key and ciphertext size and failure probability.

The failure probability δ is computed following the approach outlined above using the analysis script `small_Kyber.py` which is available online at `https://github.com/ncepuzs/Preprocess_Then_NTT/tree/master/Small_Kyber`. In addition, we also present the classical and quantum core-SVP hardness of the different proposed parameter sets of small-Kyber with the claimed security level in Table 4. The lower bounds of the cost of the primal and dual attack [20] were computed with the help of the Python script `small_Kyber.py`. Note that `small_Kyber.py` is same as the Python script `Kyber.py` [6] except that parameter sets are different.

Table 4: Parameters of small-Kyber-CPA-PKE and Kyber-CPA-PKE and derived high-level properties.

	n	k	q	η	(d_u, d_v, d_t)	δ	Security (classical,quantum)	Security level
Kyber512	256	2	7681	5	(11,3,11)	2^{-145}	(112,102)	1
Kyber768	256	3	7681	4	(11,3,11)	2^{-142}	(178,161)	3
Kyber1024	256	4	7681	3	(11,3,11)	2^{-169}	(242,219)	5
our schemes:								
small-Kyber512	256	2	3329	2	(10,3,10)	2^{-138}	(111,100)	1
small-Kyber768	256	3	3329	2	(10,5,10)	2^{-144}	(181,164)	3
small-Kyber1024	256	4	3329	1	(10,3,10)	2^{-192}	(232,210)	5

5.2 Interconversion to KEM

Small-Kyber-CPA-PKE can be converted to an IND-CPA-secure key encapsulation mechanism small-Kyber-CPA-KEM by using the public key encryption scheme to convey a secret. Furthermore, we can apply the $\mathsf{QFO}_m^{\perp}$ transform in [14] to construct an IND-CCA-secure key encapsulation mechanism small-Kyber-CCA-KEM from small-Kyber-CPA-PKE and four hash functions same as that in [6]. Instantiating small-Kyber-CCA-KEM by the parameter sets in Table 4, we can provide public key, secret key, and ciphertext sizes in Table 8 for our three KEMs that support the transmission of a 256-bit message or key.

Table 5: Sizes of public keys, secret keys, and ciphertexts of small-Kyber and Kyber in bytes.

Scheme	$\|pk\|$ (Bytes)	$\|sk\|$ (Bytes)	$\|ciphertext\|$ (Bytes)
Kyber512-CCA-KEM	736	1632	800
small-Kyber512-CCA-KEM	672	1504	736
Difference value	64	128	64
Kyber768-CCA-KEM	1088	2400	1152
small-Kyber768-CCA-KEM	992	2208	1120
Difference value	96	192	32
Kyber1024-CCA-KEM	1440	3168	1504
small-Kyber1024-CCA-KEM	1312	2912	1376
Difference value	128	256	128

Table 6: Cycle counts of key generation, encapsulation, and decapsulation of small-Kyber and Kyber.

Scheme	Key Generation	Encapsulation	Decapsulation
small-Kyber512-CCA-KEM	143628	203364	285379
Kyber512-CCA-KEM	122748	175528	209074
Ratio	1.1701	1.1586	1.3650
small-Kyber768-CCA-KEM	251538	332148	427738
Kyber768-CCA-KEM	203356	274274	321248
Ratio	1.2369	1.2110	1.3315
small-Kyber1024-CCA-KEM	390326	490956	620420
Kyber1024-CCA-KEM	293562	381882	438824
Ratio	1.3296	1.2856	1.4138

Performance of reference. Here, we give all the remaining details of results of our implementations for small-Kyber-CCA-KEM in Table 6. Both implementations are fully protected against timing attack. All cycle counts are obtained by running the implementation on a 3.30GHZ Inter Core i5-6600 processor with Turbo Boost and Hyperthreading disabled. They are median cycle counts over 100 measurements. We compiled our C implementation with gcc-5.4.0 and flags **-O3 -fomit-frame-pointer -march=native**. The implementation of our protocols is available at `https://github.com/ncepuzs/Preprocess_Then_NTT/tree/master/Small_Kyber`.

6 Application of 2PtNTT to NewHope

Recently, Alkim et al. [2] submitted a suite of public-key encapsulation mechanisms denoted as NewHope to NIST as a candidate of the standard of post-quantum cryptography, based on the conjectured quantum hardness of the RLWE problem. The NewHope cryptosystem is based on a variant of their previously proposed NewHope-Simple [4] scheme which is a semantically secure public-key encryption scheme with respect to adaptive chosen plaintext attacks (CPA). In

Appendix, we will present a brief description of NewHope-CPA-PKE including 2PtNTT and $2\mathsf{PtNTT}^{-1}$ computations, where NewHope-CPA-PKE denotes the IND-CPA secure public key encryption scheme of NewHope.

6.1 Small-NewHope parameter sets

In [2], Alkim et al. defined two parameter sets for NewHope, which they call these two schemes NewHope512, NewHope1024. According to their NewHope schemes, we will give two new parameter settings. As shown in Table 7, the modulus q in our schemes is smaller, so we call our scheme small-NewHope, i.e., NewHope512, small-NewHope1024. Note that the table also lists the derived parameter δ, which is the probability that the decryption of a valid small-NewHope-CPA-PKE ciphertext fails.

The parameters were obtained via the following approaching: 1) n is set to 512 or 1024 because the goal is to encapsulate 256-bit symmetric keys. 2) q is set to the smallest prime satisfying $\frac{n}{2}|(q-1)$ for $n = 512, 1024$, which is required to enable the 2PtNTT-based multiplication. 4) The remaining parameter η was chosen to balance between security and failure probability.

The failure probability δ is computed following the approach outlined above using the analysis script `small_NewHope.py` which is available online at `https://github.com/ncepuzs/Preprocess_Then_NTT/tree/master/Small_NewHope`. In addition, we also present the classical and quantum core-SVP hardness of the different proposed parameter sets of small-Kyber with the claimed security level in Table 7. The lower bounds of the cost of the primal and dual attack [20] were computed with the help of the Python script `small_NewHope.py`. Note that `small_NewHope.py` is same as the Python script `scripts/PQsecurity.py` [2] except that parameter sets are different.

Table 7: Parameters of small-NewHope-CPA-PKE and NewHope-CPA-PKE and derived high-level properties.

	n	q	η	δ	Security (classical,quantum)	Security level
NewHope512-CPA-PKE	512	12289	8	2^{-213}	(112,101)	1
NewHope1024-CPA-PKE	1024	12289	8	2^{-216}	(257,233)	5
our schemes:						
small-NewHope512-CPA-PKE	512	7681	5	2^{-261}	(112,101)	1
small-NewHope1024-CPA-PKE	1024	7681	5	2^{-224}	(257,233)	5

6.2 Interconversion to KEM

Small-NewHope-CPA-PKE can be converted to an IND-CPA-secure key encapsulation mechanism small-NewHope-CPA-KEM by using the public key encryption scheme to convey a secret. Furthermore, we can apply the $\mathsf{QFO}_m^{\perp}$ transform

in [14] to construct an IND-CCA-secure key encapsulation mechanism small-NewHope-CCA-KEM from small-NewHope-CPA-PKE and three hash functions same as that in [2]. Instantiating small-NewHope-CPA-KEM and small-NewHope-CCA-KEM by the parameter sets in Table 7, we can provide public key, secret key, and ciphertext sizes in Table 8 for our four KEMs that support the transmission of a 256-bit message or key.

Table 8: Sizes of public keys, secret keys, and ciphertexts of small-NewHope and NewHope in bytes.

Scheme	$\|pk\|$ (Bytes)	$\|sk\|$ (Bytes)	$\|ciphertext\|$ (Bytes)
NewHope512-CPA-KEM	928	896	1088
small-NewHope512-CPA-KEM	864	832	1024
Difference value	64	64	64
NewHope512-CCA-KEM	928	1888	1120
small-NewHope512-CCA-KEM	864	1760	1056
Difference value	64	128	64
NewHope1024-CPA-KEM	1824	1792	2176
small-NewHope1024-CPA-KEM	1696	1664	1048
Difference value	128	128	128
NewHope1024-CCA-KEM	1824	3680	2208
small-NewHope1024-CCA-KEM	1696	3424	2080
Difference value	128	256	128

Performance of reference. Here, we give all the remaining details of results of our implementations for small-NewHope KEMs in Table 9. Both implementations are fully protected against timing attack. All cycle counts are obtained by running the implementation on a 3.30GHZ Inter Core i5-6600 processor with Turbo Boost and Hyperthreading disabled. They are median cycle counts over 100 measurements. We compiled our C implementation with gcc-5.4.0 and flags **-O3 -fomit-frame-pointer -march=native**. The implementation of our protocols is available at `https://github.com/ncepuzs/Preprocess_Then_NTT/tree/master/Small_NewHope`.

7 Conclusion

We have presented Preprocess-then-NTT technique to weaken the limination for modulus q of the NTT. We further apply PtNTT to Kyber [6] and NewHope [2], and provide new parameter settings. Because of the usage of PtNTT, our new schemes achieve smaller public key sizes, smaller ciphertext sizes and a similar failure probability at a same security level. Also, it is interesting to see that the order of savings in size of the public keys and ciphertexts are the same for both the NewHope and Kyber schemes, which is due to the fact that only one bit is saved per coefficient due to the reduction in modulus. The PtNTT algorithm

Table 9: Cycle counts of key generation, encapsulation, and decapsulation of small-NewHope and NewHope.

Scheme	Key Generation	Encapsulation	Decapsulation
small-NewHope512-CPA-KEM	114033	167178	73210
NewHope512-CPA-KEM	91292	133902	34534
Ratio	1.2491	1.2485	2.1199
small-NewHope512-CCA-KEM	132886	200420	266202
NewHope512-CCA-KEM	105178	155156	175661
Ratio	1.2634	1.2917	1.5154
small-NewHope1024-CPA-KEM	272060	439358	161848
NewHope1024-CPA-KEM	184488	271180	69974
Ratio	1.4747	1.6202	2.3130
small-NewHope1024-CCA-KEM	289157	427598	538098
NewHope1024-CCA-KEM	210072	314130	361852
Ratio	1.3765	1.3612	1.4870

enables that the aforementioned improvements can be also achieved in a large portion of existing RLWE-based schemes.

8 Acknowledgments

We thank the anonymous Inscrypt'2018 reviewers for their helpful comments. This work was supported by the National Basic Research Program of China (973 project, No.2014CB340603), the National Cryptography Development Fund MMJJ20170116 and the National Natural Science Foundation of China (No. 61572495, No.61602473, No.61772515, No.61672030, No.61272040).

References

1. M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 284–293. ACM, 1997.
2. E. Alkim, R. Avanzi, J. Bos, L. Ducas, A. de la Piedra, T. Pöppelmann, P. Schwabe, and D. Stebila. Newhope-algorithm specifcations and supporting documentation. Available at: https://newhopecrypto.org/.
3. E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A new hope. In *25th USENIX Security Symposium*, pages 327–343, 2016.
4. E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Newhope without reconciliation. *IACR Cryptology ePrint Archive*, 2016:1157, 2016. Available at: http://eprint.iacr.org/2016/1157.
5. B. Applebaum, D. Cash, C. Peikert, and A. Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO 2009*, pages 595–618, 2009.

6. R. Avanzi, J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS - kyber: Algorithm specifications and supproting documentation. Available at: https://pq-crystals.org/.
7. D. J. Bernstein. Fast multiplication and its applications. *Algorithmic number theory*, 44:325–384, 2008.
8. J. W. Bos, C. Costello, M. Naehrig, and D. Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy, SP 2015*, pages 553–570, 2015.
9. J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé. CRYSTALS - kyber: a cca-secure module-lattice-based KEM. *IACR Cryptology ePrint Archive*, 2017:634, 2017.
10. E. Chu and A. George. *Inside the FFT black box: serial and parallel fast Fourier transform algorithms*. CRC Press, 1999.
11. J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.
12. J. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren. Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure KEM. In *AFRICACRYPT 2018*, pages 282–305, 2018.
13. J. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren. Saber: Module-lwr based kem. Available at:https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions.
14. D. Hofheinz, K. Hövelmanns, and E. Kiltz. A modular analysis of the fujisaki-okamoto transformation. In *TCC 2017, Part I*, pages 341–371, 2017.
15. A. Langlois and D. Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography*, 75(3):565–599, 2015.
16. V. Lyubashevsky. Lattice signatures without trapdoors. In *Advances in Cryptology - EUROCRYPT 2012*, pages 738–755, 2012.
17. V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. SWIFFT: A modest proposal for FFT hashing. In *FSE 2008*, pages 54–72, 2008.
18. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT 2010*, pages 1–23, 2010.
19. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.
20. D. Stebila and M. Mosca. Post-quantum key exchange for the internet and the open quantum safe project. In *Selected Areas in Cryptography - SAC 2016*, pages 14–37, 2016.
21. D. Stehlé, R. Steinfeld, K. Tanaka, and K. Xagawa. Efficient public key encryption based on ideal lattices. In *ASIACRYPT 2009*, pages 617–635, 2009.
22. H. Xue, X. Lu, B. Li, B. Liang, and J. He. Understanding and constructing AKE via double-key key encapsulation mechanism. In *ASIACRYPT 2018*, pages 158–189, 2018.

Appendix: Preliminaries

Public-Key Encryption. A public-key encryption $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ consists of three algorithms and a finite message space $\mathcal{M}$. The key generation

algorithm takes as input 1^λ outputs a key pair (pk, sk), where pk also defines a randomness space $\mathcal{R}$. The encryption algorithm Enc takes as input public key pk, a message $m \in \mathcal{M}$, a random string $r \xleftarrow{\$} \mathcal{R}$, and outputs a ciphertext $c \in \mathcal{C}$. The decryption algorithm Dec, on input sk and a ciphertext c, outputs either a message $m = \mathsf{Dec}(sk, c) \in \mathcal{M}$ or a special symbol $\perp$ to indicate that c is not a valid ciphertext. For correctness, we require that $\Pr[m = \mathsf{Dec}(sk, c) | c = \mathsf{Enc}(pk, m, r), r \xleftarrow{\$} \mathcal{R}] = 1 - \mathrm{negl}(\lambda)$, where $\mathrm{negl}(\lambda)$ denotes a negligible function.

Here, we define indistinguishability under chosen plaintext attacks (IND-CPA) for a PKE scheme.

Definition 1. *For any adversary $\mathcal{A}$, we define its IND-CPA advantage against a PKE scheme $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as follows:*

$$\mathrm{Adv}^{\mathrm{ind-cpa}}_{\mathrm{PKE},\mathcal{A}}(\lambda) := \left| 2 \Pr[\mathrm{Expt}^{\mathrm{ind-cpa}}_{\mathrm{PKE},\mathcal{A}}(\lambda) = 1] - 1 \right|,$$

where $\mathrm{Expt}^{\mathrm{ind-cpa}}_{\mathrm{PKE},\mathcal{A}}(\lambda)$ is experiment in Figure 1.

$\mathrm{Expt}^{\mathrm{ind-cpa}}_{\mathrm{PKE},\mathcal{A}}(\lambda)$:	$\mathrm{Expt}^{\mathrm{ind-cpa}}_{\mathrm{KEM},\mathcal{A}}(\lambda)$:	$\mathrm{Expt}^{\mathrm{ind-cca}}_{\mathrm{KEM},\mathcal{A}}(\lambda)$:
$b \xleftarrow{\$} \{0,1\}$	$b \xleftarrow{\$} \{0,1\}$	$b \xleftarrow{\$} \{0,1\}$
$(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$	$(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$	$(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$
$(m_0, m_1, st) \leftarrow \mathcal{A}_1(pk)$	$(c^*, k_0^*) \leftarrow \mathsf{Encaps}(pk)$	$(c^*, k_0^*) \leftarrow \mathsf{Encaps}(pk)$
$c^* \leftarrow \mathsf{Enc}(pk, m_b)$	$k_1^* \xleftarrow{\$} \mathcal{K}$	$k_1^* \xleftarrow{\$} \mathcal{K}$
$b' \leftarrow \mathcal{A}_2(pk, c^*, st)$	$b' \leftarrow \mathcal{A}(pk, c^*, k_b^*)$	$b' \leftarrow \mathcal{A}^{\mathsf{Decaps}(\cdot)}(pk, c^*, k_b^*)$
return $b' \stackrel{?}{=} b$	return $b' \stackrel{?}{=} b$	return $b' \stackrel{?}{=} b$

Fig. 1: Games for PKE and KEM schemes

Key-Encapsulation Mechanism. A Key-Encapsulation Mechanism (KEM) $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ consists of three algorithms and a finite message space $\mathcal{M}$. The key generation algorithm takes as input 1^λ outputs a key pair (pk, sk), where pk also defines a randomness space $\mathcal{R}$ and a finite key space $\mathcal{K}$. The encapsulation algorithm Encaps takes as input public key pk, a random string $r \xleftarrow{\$} \mathcal{R}$, and outputs $(c, k) \in \mathcal{C} \times \mathcal{K}$. The deterministic decapsulation algorithm Decaps, on input sk and a ciphertext c, outputs either a key $k = \mathsf{Dec}(sk, c) \in \mathcal{K}$ or a rejection symbol $\perp \notin \mathcal{K}$. For correctness, we require that $\Pr[k = \mathsf{Decaps}(sk, c) | (k, c) = \mathsf{Encaps}(pk, r), r \xleftarrow{\$} \mathcal{R}] = 1 - \mathrm{negl}(\lambda)$, where $\mathrm{negl}(\lambda)$ denotes a negligible function.

We define indistinguishability under chosen-plaintext and chosen-ciphertext attacks (denoted by IND-CPA, and IND-CCA) for KEMs, respectively.

Definition 2. *For any adversary $\mathcal{A}$, we define its IND-CPA and IND-CCA advantages against a KEM scheme* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *as follows:*

$$\mathrm{Adv}^{\mathrm{ind-cpa}}_{\mathrm{KEM},\mathcal{A}}(\lambda) := \left|2\Pr[\mathrm{Expt}^{\mathrm{ind-cpa}}_{\mathrm{KEM},\mathcal{A}}(\lambda) = 1] - 1\right|,$$
$$\mathrm{Adv}^{\mathrm{ind-cca}}_{\mathrm{KEM},\mathcal{A}}(\lambda) := \left|2\Pr[\mathrm{Expt}^{\mathrm{ind-cca}}_{\mathrm{KEM},\mathcal{A}}(\lambda) = 1] - 1\right|,$$

where $\mathrm{Expt}^{\mathrm{ind-cpa}}_{\mathrm{KEM},\mathcal{A}}(\lambda)$ *and* $\mathrm{Expt}^{\mathrm{ind-cca}}_{\mathrm{KEM},\mathcal{A}}(\lambda)$ *are experiments in Figure 1.*

small-KYBER-CPA-PKE. In Algorithms 1, 2 and 3, we will present a brief description of KYBER-CPA-PKE, where KYBER-CPA-PKE denotes the IND-CPA secure public key encryption scheme of KYBER.

Let k, d_t, d_u, d_v be positive integers, and recall that $n = 256$. Let $\mathcal{M} = \{0,1\}^{256}$ denote the message space, where every message $m \in \mathcal{M}$ can be viewed as a polynomial in $\mathcal{R}_q$ with coefficients in $\{0,1\}$. Consider the public-key encryption scheme KYBER=(Gen, Enc, Dec) as described in Algorithms 1-3.

Algorithm 1: KYBER-CPA-PKE Key Generation

Output: Secret key $sk := \mathbf{s}$
Output: Public key $pk := (\mathbf{t}, \rho)$
Algorithm KYBER-CPA-PKE.Gen()
  $\rho, \sigma \xleftarrow{\$} \{0,1\}^{256}$
  $\mathbf{A} \sim \mathcal{R}_q^{k \times k} := \mathsf{Sam}(\rho)$
  $(\mathbf{s}, \mathbf{e}) \sim \psi_\eta^k \times \psi_\eta^k := \mathsf{Sam}(\sigma)$
  $\mathbf{t} := \mathsf{Compress}_q(\mathbf{As} + \mathbf{e}, d_t)$
end

Algorithm 2: KYBER-CPA-PKE Encryption

Input: Public key $pk = (\mathbf{t}, \rho)$
Input: Message $m \in \mathcal{M}$
Output: Ciphertext $c := (\mathbf{u}, v)$
KYBER-CPA-PKE.Enc()
  $r \xleftarrow{\$} \{0,1\}^{256}$
  $\mathbf{t} := \mathsf{Decompress}_q(\mathbf{t}, d_t)$
  $\mathbf{A} \sim \mathcal{R}_q^{k \times k} := \mathsf{Sam}(\rho)$
  $(\mathbf{r}, \mathbf{e}_1, e_2) \sim \psi_\eta^k \times \psi_\eta^k \times \psi_\eta := \mathsf{Sam}(r)$
  $\mathbf{u} := \mathsf{Compress}_q(\mathbf{A}^\top \mathbf{r} + \mathbf{e}_1, d_u)$
  $v := \mathsf{Compress}_q(\mathbf{t}^\top \mathbf{r} + e_2 + \lfloor \frac{q}{2} \rceil \cdot m, d_v)$
end

Algorithm 3: KYBER-CPA-PKE Decryption

Input: Secret key $sk = \mathbf{s}$
Input: Ciphertext $c = (\mathbf{u}, v)$
Output: Message $\mu \in \mathcal{M}$
Algorithm KYBER-CPA-PKE.Dec()
  $\mathbf{u} := \mathsf{Decompress}_q(\mathbf{u}, d_u)$
  $v := \mathsf{Decompress}_q(v, d_v)$
  $\mu := \mathsf{Compress}_q(v - \mathbf{u}^\top \mathbf{s}, 1)$
end

small-NEWHOPE-CPA-PKE. In Algorithms 4, 5 and 6, we will present a brief description of NEWHOPE-CPA-PKE including $\mathsf{2PtNTT}$ and $\mathsf{2PtNTT}^{-1}$ computations, where NEWHOPE-CPA-PKE denotes the IND-CPA secure public key encryption scheme of NEWHOPE.

Algorithm 4: small-NEWHOPE-CPA-PKE Key Generation

Output: Secret key $sk := s$
Output: Public key $pk := \widehat{b}, seed$
Algorithm small-NEWHOPE-CPA-PKE.Gen()
  $seed \xleftarrow{\$} \{0,1\}^{256}$
  $\widehat{a} \leftarrow \mathsf{Samp}(seed)$
  $s, e \xleftarrow{\$} \psi_\eta$
  $\widehat{s} \leftarrow \mathsf{2PtNTT}(s)$
  $\widehat{b} \leftarrow \widehat{a} \bowtie \widehat{s} + \mathsf{2PtNTT}(\mathbf{e})$
end

Algorithm 5: small-NEWHOPE-CPA-PKE Encryption

Input: Public key $pk = \widehat{b}, seed$
Input: Message $m \in \{0,1\}^{256}$
Output: Ciphertext $c := \widehat{u}, h$

Algorithm small-NEWHOPE-CPA-PKE.Enc()
- $\widehat{a} \leftarrow \mathsf{Samp}(seed)$
- $s', e', e'' \xleftarrow{\$} \psi_\eta$
- $\widehat{t} \leftarrow \mathsf{2PtNTT}(s')$
- $\widehat{u} \leftarrow \widehat{a} \bowtie \widehat{t} + \mathsf{2PtNTT}(e')$
- $v' \leftarrow \mathsf{2PtNTT}^{-1}(\widehat{b} \bowtie \widehat{t}) + e'' + \mathsf{Encode}(m)$ //Encode : $\{0,1\}^{256} \rightarrow \mathcal{R}_q$
- $h \leftarrow \mathsf{Compress}_q(v', 3)$

end

Algorithm 6: small-NEWHOPE-CPA-PKE Decryption

Input: Secret key $sk = s$
Input: Ciphertext $c := \widehat{u}, h$
Output: Message $\mu \in \{0,1\}^{256}$

Algorithm small-NEWHOPE-CPA-PKE.Dec()
- $\widehat{s} \leftarrow \mathsf{2PtNTT}(s)$
- $\mathbf{v}' \leftarrow \mathsf{Decompress}_q(h, 3)$
- $\mu \leftarrow \mathsf{Decode}(v' - \mathsf{2PtNTT}^{-1}(\widehat{u} \bowtie \widehat{s}))$

end