Performance Analysis of NTT Algorithms

David García Lleyda[1], Víctor Gayoso Martínez[1(✉)], Luis Hernández Encinas[2], Agustín Martín Muñoz[2], and Óscar Castillo Campo[3]

[1] Centro Universitario de Tecnología y Arte Digital (U-tad), Data Research and Computation Group (DRACO), 28290 Las Rozas de Madrid, Spain
david.lleyda@live.u-tad.com, victor.gayoso@u-tad.com
[2] Institute of Physical and Information Technologies (ITEFI), Spanish National Research Council (CSIC), 28006 Madrid, Spain
{luis.h.encinas,agustin.martin}@csic.es
[3] Department of Industrial Engineering, Universidad Nebrija, 28015 Madrid, Spain
ocastillo@nebrija.es

Abstract. The Number Theoretic Transform is an important tool in Cryptography, as it can be used to multiply polynomials in the rings $\mathbb{Z}_q[x]/\langle x^n - 1\rangle$ and $\mathbb{Z}_q[x]/\langle x^n + 1\rangle$. In its plain version, the NTT is slower than the direct multiplication method for these rings. To overcome this disadvantage, some optimised variants of the NTT have been proposed in recent years. This contribution analyses one of these variants and presents an empirical comparison of the running time for the three methods considered in the study, which allows to determine the length of the polynomials for which each method is the fastest.

Keywords: Number Theoretic Transform · post-quantum cryptography · polynomial multiplication · quotient rings · software implementation

1 Introduction

The Number Theoretic Transform (NTT) is a mathematical tool particularly well known in the field of Cryptography, more specifically in the areas of Post-Quantum Cryptography (PQC) and Homomorphic Encryption (HE). In PQC, it is used intensively by lattice-based algorithms for the multiplication of polynomials [8,10].

In Mathematics, a ring is an algebraic structure that consists of a set of elements equipped with two binary operations, typically referred to as addition and multiplication, which satisfy certain properties. A ring over a finite field is a ring where the elements belong to a finite field such as $\mathbb{F}_q = \{[0], [1], \ldots, [q-1]\}$. In $\mathbb{Z}_q$, the elements of the field are the residue classes of $\mathbb{Z}$. Given $a \in \mathbb{Z}$, the residue class $[a]_q$ (or simply $[a]$) contains all the integer numbers that are congruent to each other modulo q, i.e., the integers that have the same remainder when divided by the modulus q.

H. Quintián et al. (Eds.): ICEUTE 2024/CISIS 2024, LNNS 957, pp. 168–178, 2024.
https://doi.org/10.1007/978-3-031-75016-8_16

The NTT is very closely related to the Discrete Fourier Transform (DFT) but operates using modular arithmetic over finite rings. The NTT represents polynomials as vectors, so polynomial multiplications can be calculated alternatively by combinations of direct and inverse transformations and point-to-point operations.

Compared to traditional polynomial multiplication methods, optimised variants of the NTT can offer significant advantages in terms of efficiency and scalability. However, the threshold at which the NTT is faster than traditional algorithms is not clear in the scientific literature.

In this article, a detailed description of the NTT algorithm and a recursive variant is presented. In addition to that, this contribution includes a performance comparison of the algorithms considered in the study using an application developed in the Java programming language, so it can be checked how the theoretical performance advantages translate into software implementations.

There are several works related to this contribution. Anna Bakkebø's master thesis [2], published in 2020, describes the "divide and conquer" technique for the NTT that is used in this study. She also developed C code to test the performance of the implementation. In comparison to that work, in this contribution we have not only analysed the ring $\mathbb{Z}_q[x]/\langle x^n+1\rangle$, but also $\mathbb{Z}_q[x]/\langle x^n-1\rangle$, which requires fewer multiplications. In addition to that, in this work we have included the plain version of the NTT in the comparison and have taken into account the initialisation time for the NTT, not only the multiplication time.

Zhichuang Liang and Yunlei Zhao published a survey of the NTT in 2022 [6]. This survey mentions several algorithms and tricks for improving the performance of the NTT, including the so-called Cooley-Tukey and Gentleman-Sande techniques [7]. These techniques are closely related to the one used by Bakkebø, and have been described in prior works (e.g., in [3]). However, only the mathematical description of the techniques is included in [6], as it is a theoretical survey.

Satriawan et al. have recently published a complete introductory guide for the NTT [9]. In their article, they describe the Cooley-Tukey and the Gentleman-Sande techniques. As mentioned above, our contribution is based more directly on Bakkebø's recursive design than on the algorithms described in [9], although they are similar. In addition, the authors of [9] did not include a software performance comparison.

The rest of this article is organised as follows: Sect. 2 provides an introduction to the direct multiplication method in the rings $\mathbb{Z}_q[x]/\langle x^n-1\rangle$ and $\mathbb{Z}_q[x]/\langle x^n+1\rangle$, where x^n-1 and x^n+1 are called the cyclotomic and anticyclotomic polynomials, respectively. Section 3 explains the plain version of the NTT and the optimised version described in [2]. After that, Sect. 4 presents the results of the performance comparison using the Java application developed for this study. Finally, conclusions are summarised in Sect. 5.

2 Polynomial Multiplication in the Ring $\mathbb{Z}_q[x]/\langle f(x)\rangle$

Given a prime number $q \in \mathbb{Z}$, the polynomial ring $\mathbb{Z}_q[x]$ is the set of polynomials with coefficients in the finite field $\mathbb{Z}_q$. In this context, $\mathbb{Z}_q[x]/\langle f(x)\rangle$ represents the quotient ring of $\mathbb{Z}_q[x]$, which consists of the residue classes of all polynomials that are congruent to each other modulo $f(x)$, i.e., the ring of the polynomials with coefficients in $\mathbb{Z}_q$ and degree, at most, $\deg(f(x))-1$. The size of the quotient ring $\mathbb{Z}_q[x]/\langle f(x)\rangle$ is equal to the degree of $f(x)$. This means that there are $q^{\deg(f(x))}$ elements in the ring.

If $f(x)$ is irreducible (i.e., it cannot be factored into the product of two or more polynomials of smaller degree), then the quotient ring $\mathbb{Z}_q[x]/\langle f(x)\rangle$ is a finite field. In comparison, if $f(x)$ is not irreducible, then the quotient ring is instead a ring with zero divisors.

This contribution focuses on two polynomial rings, namely $\mathbb{Z}_q[x]/\langle x^n-1\rangle$ and $\mathbb{Z}_q[x]/\langle x^n+1\rangle$, where $n = 2^m$ for some $m \in \mathbb{N}$. These rings, specially the second one, are commonly used in lattice-based cryptography by algorithms such as NewHope [1] or Kyber [4].

Given $a(x) = a_0 + a_1\,x \cdots + a_{n-1}\,x^{n-1}$ and $b(x) = b_0 + b_1\,x \cdots + b_{n-1}\,x^{n-1}$, polynomial addition in $\mathbb{Z}_q[x]/\langle f(x)\rangle$ is a straightforward operation, since the coefficients of $a(x)+b(x)$ are simply computed by adding them modulo q, which is a very fast operation from a computational point of view:

$$a(x) + b(x) = \sum_{i=0}^{n-1} \big((a_i + b_i)(\mathrm{mod}\ q)\big)\, x^i.$$

In contrast, polynomial multiplication is more difficult to perform, as after the regular multiplication of $a(x)$ and $b(x)$ it is necessary not only to reduce the coefficients modulo q, but also to calculate the remainder of the polynomial when divided by $f(x)$.

If we consider $f(x) = x^n - 1$, given that $x^{k+n} \equiv x^k \pmod{x^n-1}$, the multiplication of the polynomials $a(x)$ and $b(x)$ in $\mathbb{Z}_q[x]/\langle x^n-1\rangle$ can be represented by the following formula:

$$a(x)\cdot b(x) = \left(\sum_{i=0}^{n-1}\sum_{j=0}^{n-1-i} a_i\cdot b_j\cdot x^{i+j} + \sum_{i=1}^{n-1}\sum_{j=n-i}^{n-1} a_i\cdot b_j\cdot x^{i+j-n}\right) \pmod q.$$

On the other hand, if $f(x) = x^n + 1$, as $x^{k+n} \equiv -x^k \pmod{x^n+1}$, the multiplication of the polynomials $a(x)$ and $b(x)$ in $\mathbb{Z}_q[x]/\langle x^n+1\rangle$ can be represented by this formula:

$$a(x)\cdot b(x) = \left(\sum_{i=0}^{n-1}\sum_{j=0}^{n-1-i} a_i\cdot b_j\cdot x^{i+j} - \sum_{i=1}^{n-1}\sum_{j=n-i}^{n-1} a_i\cdot b_j\cdot x^{i+j-n}\right) \pmod q.$$

For polynomials with n coefficients, it is clear from the previous formulas that $n \times n$ multiplications need to be performed, so the complexity of the direct multiplication method is $O(n^2)$.

3 NTT

The NTT algorithm allows to transform finite sequences into other finite sequences. In the context of this study, two versions of the NTT are used: the *regular* NTT (also known as Positive-Wrapped NTT or PW-NTT in the scientific literature) and the *negacyclic* NTT (also known as Negative-Wrapped NTT or NW-NTT). The inverse transformations will be referred to as regular INTT and negacyclic INTT, respectively.

In what follows, we will assume that n is a power of 2, so $n = 2^m$ for some $m \in \mathbb{N}$. This is an assumption usually contemplated in the definition of cryptographic algorithms, and allows to include additional improvements in some computations.

3.1 Regular NTT

Let q be a prime number such that $q \equiv 1 \pmod{n}$, and let ω be a primitive root of order n in $\mathbb{Z}_q$, such that $\omega^n \equiv 1 \pmod{q}$. The regular NTT of the polynomial $a(x) = a_0 + a_1\, x + \cdots + a_{n-1}\, x^{n-1} \sim [a_0, a_1, \ldots, a_{n-1}] \in \mathbb{Z}_q^n$ over the ring $\mathbb{Z}_q\,[x]\,/\,\langle x^n - 1\rangle$ is denoted as $\tilde{a}(x) = \mathrm{NTT}(a(x))$, where the coefficients of $\tilde{a}(x)$ are computed in the following way [9]:

$$\tilde{a_i} = \sum_{j=0}^{n-1} a_j \omega^{ij} \big(\mathrm{mod}\ q\big)\ ,\ \forall i \in \{0, 1, ..., n-1\}\,.$$

In the regular INTT, where $a(x) = \mathrm{INTT}(\tilde{a}(x))$, the coefficients of the resulting polynomial are calculated as follows:

$$a_i = \frac{1}{n} \sum_{j=0}^{n-1} \tilde{a}_j \omega^{-ij} \big(\mathrm{mod}\ q\big)\ ,\ \forall i \in \{0, 1, ..., n-1\}\,.$$

Using the NTT, polynomial multiplication in the ring $\mathbb{Z}_q\,[X]\,/\langle x^n - 1\rangle$ can be achieved in this way, where $\circ$ represent the point-to-point multiplication of two vectors:

$$a(x) \cdot b(x) = \mathrm{INTT}\left(\mathrm{NTT}\,(a) \circ \mathrm{NTT}\,(b)\right).$$

Before performing the multiplication, it is necessary to compute two matrices of $n \times n$ elements with the powers of ω and ω^{-1}, respectively. In principle, n^2 multiplications should be performed to obtain each matrix, but in practice the number is lower because the elements in both the first row and the first column are always 1. Furthermore, some symmetries in the powers of both elements can be exploited in order to reduce the number of multiplications.

Regarding the complexity of the polynomial multiplication itself, it is clear that in the regular NTT n^2 multiplications are needed. If in the regular INTT a double matrix with the products $n^{-1}\omega^{-ij}$ is used instead of just ω^{-ij} (which

adds some cost to the precomputation phase), then again n^2 multiplications are required.

In total, $3n^2$ operations would be needed for the whole process, which means that the complexity of the regular NTT is $O(n^2)$.

3.2 Negacyclic NTT

Let q be a prime number such that $q \equiv 1 \pmod{2n}$, and let ω be a primitive root of order n in $\mathbb{Z}_q$. In addition to that, let ψ be a primitive root of order $2n$ in $\mathbb{Z}_q$, such that $\psi^2 = \omega$. If $a(x) = a_0 + a_1\, x + \cdots + a_{n-1}\, x^{n-1} \sim [a_0, a_1, \ldots, a_{n-1}] \in \mathbb{Z}_q^n$, then let us define $\hat{a} = \left[a_0, \psi a_1, \ldots, \psi^{n-1} a_{n-1}\right]$. With these elements, the coefficients of the negacyclic NTT of the polynomial $a(x)$ over the ring $\mathbb{Z}_q\left[x\right] / \langle x^n + 1\rangle$ are computed as follows [7]:

$$\tilde{a}_i = \sum_{j=0}^{n-1} \hat{a}_j \omega^{ij} \big(\mathrm{mod}\ q\big)\ ,\ \forall i \in \{0, 1, ..., n-1\}\,.$$

Regarding the inverse operation, the coefficients of $a(x) = \mathrm{INTT}(\hat{a}(x))$ are computed in this way:

$$\hat{a}_i = \frac{1}{n} \sum_{j=0}^{n-1} \tilde{a}_j \omega^{-ij} \big(\mathrm{mod}\ q\big)\ ,\ \forall i \in \{0, 1, ..., n-1\}\,.$$

After that operation, in order to obtain the coefficients of the polynomial $a(x)$ it is necessary to consider that $a(x) = \left[\hat{a}_0, \psi^{-1}\hat{a}_1, \ldots, (\psi^{-1})^{n-1}\hat{a}_{n-1}\right]$.

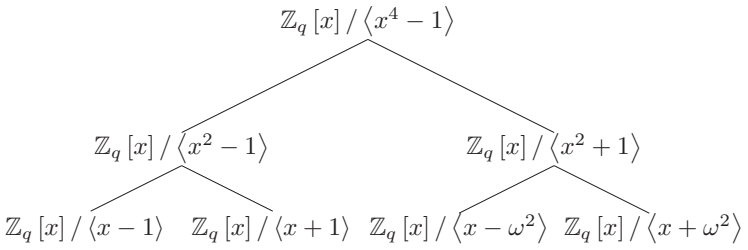
Once again, the multiplication of polynomials in the ring $\mathbb{Z}_q\left[X\right] / \langle x^n - 1\rangle$ can be achieved in this way:

$$a(x) \cdot b(x) = \mathrm{INTT}\left(\mathrm{NTT}\left(a\right) \circ \mathrm{NTT}\left(b\right)\right).$$

Similar considerations can be made to calculate the complexity of the whole process, although in this case the multiplications by the powers of ψ and ψ^{-1} require $2n$ additional multiplications. However, since $2n$ is negligible compared to n^2 for large values of n, the overall complexity of the negacyclic NTT remains $O(n^2)$.

3.3 Optimized NTT

In 2001, Daniel Bernstein showed that it is possible to apply a variant of the Cooley-Tuckey and Gentleman-Sande techniques, typically associated with the Fast Fourier Transform (FFT), to the NTT [3]. By applying these techniques, the time complexity of the NTT/INTT can be reduced from $O(n^2)$ to $O\left(n \log_2(n)\right)$. This is possible because of the special form of $x^n - 1$ and $x^n + 1$, where n is a power of two. For example, if $n = 4$, ω would be a primitive root of order 4, ω^2 would be a primitive root of order 2, etc., so the polynomial could be decomposed as follows:



Following with the $\mathbb{Z}_q[X]/\langle x^n-1\rangle$ example, the double matrix with the powers of ω would have the following aspect:

$$\begin{pmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^3 & \omega^6 & \omega^9 \end{pmatrix} \equiv \begin{pmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^2 & \omega^0 & \omega^2 \\ \omega^0 & \omega^3 & \omega^2 & \omega^3 \end{pmatrix}$$

With this matrix in mind, the equations to obtain the coefficients of $\tilde{a}(x)$ are the following ones:

$$\begin{aligned} \tilde{a}_0 =& \omega^0\, a_0 + \omega^0\, a_1 + \omega^0\, a_2 + \omega^0\, a_3 \\ \tilde{a}_1 =& \omega^0\, a_0 + \omega^1\, a_1 + \omega^2\, a_2 + \omega^3\, a_3 \\ \tilde{a}_2 =& \omega^0\, a_0 + \omega^2\, a_1 + \omega^0\, a_2 + \omega^2\, a_3 \\ \tilde{a}_0 =& \omega^0\, a_0 + \omega^3\, a_1 + \omega^2\, a_2 + \omega^3\, a_3 \end{aligned}$$

However, taking into account that $\omega^2 = -1$ and $\omega^3 = -\omega$, the previous computations can be arranged in a more convenient way:

$$\begin{aligned} \tilde{a}_0 &= \left(a_0 + \omega^0\, a_2\right) + \omega^0 \left(a_1 + \omega^0\, a_3\right) \\ \tilde{a}_1 &= \left(a_0 - \omega^0\, a_2\right) + \omega^1 \left(a_1 - \omega^0\, a_3\right) \\ \tilde{a}_2 &= \left(a_0 + \omega^0\, a_2\right) - \omega^0 \left(a_1 + \omega^0\, a_3\right) \\ \tilde{a}_3 &= \left(a_0 - \omega^0\, a_2\right) - \omega^1 \left(a_1 - \omega^0\, a_3\right) \end{aligned}$$

Out of the 16 multiplications initially needed, only 4 multiplications must now be performed: $\omega^0\, a_2$, $\omega^0\, a_3$, $\omega^0 \left(a_1 + \omega^0\, a_3\right)$, and $\omega^1 \left(a_1 - \omega^0\, a_3\right)$. Even though some of those multiplications may seem trivial, given that one of the operands is $\omega^0 = 1$, the reader must take into account that this example is related to $f(x) = x^4 - 1$. As the degree of the polynomial increases, trivial multiplications become a minority.

Taking this technique as starting point, it is possible to implement this procedure as a recursive function, as shown in [2]. In this optimisation, each polynomial can be recursively decomposed into sub-polynomials with terms of even and odd order being treated in the same way. In order to simplify the notation, in the following formulas $a(x) = X$, the subindex e means *even* and the subindex

o means *odd*.

$$\begin{aligned}\mathrm{NTT}(X)[k] &= \mathrm{NTT}(X_e)[k] + \omega_k \,\mathrm{NTT}(X_o)[k] \mod q \\ \mathrm{NTT}(X)\left[k+\frac{n}{2}\right] &= \mathrm{NTT}(X_e)[k] - \omega_k \,\mathrm{NTT}(X_o)[k] \mod q\end{aligned}$$

Using the polynomial $a(x) = 2 + 5x + 3x^2 + x^3$ with $n = 4$, $q = 17$, and $\omega = 4$, its NTT could be obtained as it is described in the next paragraphs.

$$\begin{aligned}X_o = [5,1] \rightarrow X_{o_o} = [1],\ X_{o_e} = [5] \\ X_e = [2,3] \rightarrow X_{e_o} = [3],\ X_{e_e} = [2]\end{aligned}$$

Given that $\omega = 4$, it is necessary to compute its powers for the omega array:

$$\begin{aligned}\omega^0 = 4^0 \equiv 1 \mod 17 \qquad \omega^1 = 4^1 \equiv 4 \mod 17 \\ \omega^2 = 4^2 \equiv 16 \mod 17 \qquad \omega^3 = 4^3 \equiv 13 \mod 17\end{aligned}$$

Now it is possible to compute the next step:

$$\begin{aligned}\mathrm{NTT}(X_o)[k] &= \mathrm{NTT}(X_{o_e})[k] + \omega_k \,\mathrm{NTT}(X_{o_o})[k] \mod q \\ \mathrm{NTT}(X_o)\left[k+\frac{n}{2}\right] &= \mathrm{NTT}(X_{o_e})[k] - \omega_k \,\mathrm{NTT}(X_{o_o})[k] \mod q\end{aligned}$$

Thus:

$$\begin{aligned}\mathrm{NTT}(X_o)[0] &= 5 + 1 \cdot 1 \mod 17 \equiv 6 \\ \mathrm{NTT}(X_o)[1] &= 5 - 1 \cdot 1 \mod 17 \equiv 4 \\ \mathrm{NTT}(X_e)[0] &= 2 + 1 \cdot 3 \mod 17 \equiv 5 \\ \mathrm{NTT}(X_e)[1] &= 2 - 1 \cdot 3 \mod 17 \equiv 16\end{aligned}$$

In this way, $\mathrm{NTT}([5,1]) = [6,4]$ and $\mathrm{NTT}([2,3]) = [5,16]$. In the last step:

$$\begin{aligned}\mathrm{NTT}(X)[0] &= 5 + 1 \cdot 6 \mod 17 \equiv 11 \\ \mathrm{NTT}(X)\left[0+\frac{4}{2}\right] &= 5 - 1 \cdot 6 \mod 17 \equiv 16 \\ \mathrm{NTT}(X)[1] &= 16 + 4 \cdot 4 \mod 17 \equiv 15 \\ \mathrm{NTT}(X)\left[1+\frac{4}{2}\right] &= 16 - 4 \cdot 4 \mod 17 \equiv 0\end{aligned}$$

As it was expected, $\mathrm{NTT}([2,5,3,1]) = [11,15,16,0]$. Following this procedure, only $\frac{1}{2}n\log_2(n)$ multiplications are necessary for the regular NTT. As it is shown in [6], the regular INTT and the negacyclic NTT have the same number of operations, while the negacyclic INTT requires $\frac{1}{2}n\log_2(n)+n$ multiplications. With this optimisation, the overall time complexity of polynomial multiplication is $O(n\log_2(n))$.

4 Experimental Comparison

In order to compare the performance of these algorithms in practice, a Java application has been developed specifically for this contribution. The code is available at [5].

The application is flexible enough so the three algorithms described in this contribution (direct multiplication, NTT and the optimized version of NTT) can be tested in both $\mathbb{Z}_q[x]/\langle x^n - 1\rangle$ and $\mathbb{Z}_q[x]/\langle x^n + 1\rangle$. It is also possible to select the number of coefficients for the polynomials and the number of multiplications to be performed for each algorithm.

The time required to initialise the data structures for the regular NTT and the optimised NTT has been included in the tables presented in this section, so the reader can check how this initialisation time affects the running time. In order to obtain more accurate results and to avoid possible interferences in the measurement process, batches of 100 consecutive tests were performed with each algorithm using a computer with an Intel Core i7-10750H processor.

Table 1 contains the parameters used in the different tests. In all of them $q = 12289$, which is a value that allows to test a wide range of cases. Another reason for using this value is that it is employed by many PQC algorithms [6].

Table 1. Parameters used in the tests

n	$n^{-1} \pmod q$	ω	$\omega^{-1} \pmod q$	ψ	$\psi^{-1} \pmod q$
4	9217	1479	10810	4043	5146
8	10753	4043	5146	5736	11567
16	11521	722	6553	2545	1212
32	11905	1212	2545	5023	2975
64	12097	563	5828	2747	9238
128	12193	81	11227	9	2731
256	12241	9	2731	3	8193
512	12265	3	8193	1321	8633
1024	12277	49	1254	7	8778
2048	12283	7	8778	5846	9613

Table 2 shows the results for the tests with the ring $\mathbb{Z}_q[x]/\langle x^n - 1\rangle$. The table provides two values in each cell (except for the column on the direct multiplication method): the mean time to perform one multiplication and the mean time plus the initialisation time, which is of interest in cases where only one multiplication has to be performed for a given set of parameters. The figures in bold represent the lower values for each combination of polynomial length and option (including or excluding initialisation time).

Table 3 shows the results in the same testing conditions, with the only difference that, in this case, the ring $\mathbb{Z}_q[x]/\langle x^n + 1\rangle$ has been used.

Table 2. Mean time in nanoseconds for one multiplication (regular NTT)

n	Direct multiplication	Schoolbook NTT	Optimized NTT
4	**2118**	4045/14956945	7078/38878
8	**6164**	10167/21156967	12648/38748
16	**14980**	29517/12195517	**13617**/52417
32	**31338**	90685/13137985	**29617**/65317
64	**83867**	274300/11703600	**61453**/104153
128	220970	821035/8190935	**118805/186505**
256	586513	2871617/15330617	**273837/367737**
512	2237543	10821181/34928081	**373777/472277**
1024	12164922	42695440/103778740	**758134/874234**
2048	54659706	170061732/359191632	**1109974/1285174**

Table 3. Mean time in nanoseconds for one multiplication (negacyclic NTT)

n	Direct multiplication	Schoolbook NTT	Optimized NTT
4	**2380**	6730/14137630	6399/201499
8	**6072**	15292/23220692	10865/276665
16	**18394**	32866/11246966	**15993**/210393
32	**36487**	93322/13553622	**32446**/279346
64	**94732**	271904/11272804	**65731**/230331
128	**220948**	826926/8523026	**123388**/228288
256	575243	2885876/17490876	**200262/325262**
512	2357461	10823606/34045006	**390973/588373**
1024	11041841	42687105/104100905	**577396/760896**
2048	55976373	169737361/351619561	**1353031/1859531**

The results show that, in our tests, direct multiplication is the fastest method for polynomials with up to 8 coefficients in both the regular and negacylic cases when no initialisation time is considered. However, when initialisation time is taken into account, direct multiplication is the fastest method for polynomials with up to 64/128 coefficients in the regular/negacylic case. For the rest of the polynomials, the optimised version of NTT offers a great advantage. For example, when $n = 2048$ its computation time is less than 5% of the time needed by the direct multiplication method.

On the other hand, the plain NTT implementation is around three times slower than the direct multiplication method in most tests when no initialisation is considered. This is to be expected, as each NTT and INTT operation takes approximately the same time as the entire direct multiplication. When

initialisation time is taken into account, the plain NTT method is at least seven times slower than the direct method.

When comparing tests for close values of n, in some cases there is a noticeable variability. This could be due to the nature of the tests, as a new set of random coefficients is generated for each value of n. In addition, both the NTT and its optimised version use the `BigInteger` class, while the direct multiplication method uses only basic datatypes (the integer type `long`), so in the latter case no object management is required.

Regarding the initialisation time, it ranges from 26 picoseconds for $n = 8$ to 175 picoseconds for $n = 2048$. The initialisation values for $n \in \{4, 8, 16, 32, 64\}$ are very similar, which means that the instantiation of the `Random` class takes more time compared to the time spent by successively calling the `nextInt` method. From $n = 128$ onwards, the time needed for calling the `nextInt` method n times exceeds the time needed to instantiate the class and follows a linear trend.

5 Conclusions

The NTT is a relevant algorithm in the field of Cryptography. It can be used to speed up polynomial multiplication in the rings $\mathbb{Z}_q[x]/\langle x^n-1\rangle$ and $\mathbb{Z}_q[x]/\langle x^n+1\rangle$. Even though there have been previous studies about the theoretical performance of both the regular NTT and some of its variants, it is important to obtain accurate data on the performance of software implementations of these algorithms.

In this contribution, three algorithms for computing polynomial multiplications in the rings $\mathbb{Z}_q[x]/\langle x^n - 1\rangle$ and $\mathbb{Z}_q[x]/\langle x^n + 1\rangle$ have been compared: the direct method, the plain NTT version and the optimised NTT variant described in [2]. A Java implementation was created to test these algorithms.

The tests used values defined in actual PQC specifications, such as the value of q and the length of the polynomials. With this data, it can be stated that the direct method is faster for polynomials with a small number of coefficients. However, for polynomials with a large number of coefficients, the optimised version of NTT is significantly faster. In fact, for polynomials with more than 1000 coefficients, its running time is as much as 5% of that of the direct method.

Regarding the plain NTT implementation, it is the worse option in terms of performance, and should not be used in any cryptographic implementation.

Acknowledgements. This work was supported in part by the Spanish State Research Agency (AEI) of the Ministry of Science and Innovation (MICINN), project P2QProMeTe (PID2020-112586RB-I00/AEI/10.13039/501100011033).

References

1. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange—a new hope. In: 25th USENIX Security Symposium (USENIX Security 2016), pp. 327–343 (2016). https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/alkim
2. Bakkebø, A.: Implementation of the Number Theoretic Transform for Faster Lattice-Based Cryptography. https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2778380/no.ntnu
3. Bernstein, D.J.: Multidigit multiplication for mathematicians (2001). https://cr.yp.to/papers/m3-20010811-retypeset-20220327.pdf
4. Bos, J., et al.: CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In: 2018 IEEE European Symposium on Security and Privacy (2018). https://doi.org/10.1109/eurosp.2018.00032
5. Gayoso Martínez, V.: NTT performance test. https://github.com/victor-gayoso/ntt-test
6. Liang, Z., Zhao, Y.: Number theoretic transform and its applications in lattice-based cryptosystems: a survey (2022). https://arxiv.org/pdf/2211.13546.pdf
7. Longa, P., Naehrig, M.: Speeding up the number theoretic transform for faster ideal lattice-based cryptography (2016). https://eprint.iacr.org/2016/504
8. Peikert, C.: Lattice cryptography for the internet. In: Mosca, M. (ed.) PQCrypto 2014. LNCS, vol. 8772, pp. 197–219. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11659-4_12
9. Satriawan, A., Mareta, R., Lee, H.: A complete beginner guide to the number theoretic transform (NTT). Cryptology ePrint Archive, Paper 2024/585 (2024). https://eprint.iacr.org/2024/585
10. Zhang, J., Zhang, Z.: Lattice-Based Cryptosystems. Springer, Singapore (2020). https://doi.org/10.1007/978-981-15-8427-5