Boosted Performances of NTRUencrypt Post-Quantum Cryptosystem

El Hassane Laaji* and Abdelmalek Azizi

Mohamed First University, Oujda, Morocco
E-mail: e.laaji@ump.ac.ma; abdelmalekazizi@yahoo.fr
**Corresponding Author*

Received 06 March 2021; Accepted 19 May 2021;
Publication 19 June 2021

Abstract

The bottleneck of all cryptosystems is the difficulty of the computational complexity of the polynomials multiplication, vectors multiplication, etc. Thus most of them use some algorithms to reduce the complexity of the multiplication like NTT, Montgomery, CRT, and Karatsuba algorithms, etc. We contribute by creating a new release of NTRUencrypt1024 with great improvement, by using our own polynomials multiplication algorithm operate in the ring of the form $\mathbf{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$, combined to Montgomery algorithm rather than using the NTT algorithm as used by the original version. We obtained a good result, our implementation outperforms the original one by speed-up of a factor up to (X10) for encryption and a factor up to (X11) for decryption functions. We note that our improved implementation used the latest hash function standard SHA-3, and reduce the size of the public key, private key, and cipher-text from 4097 bytes to 2049 bytes with the same security level.

Keywords: Montgomery algorithm, NTT algorithm, lattices-based cryptography, post quantum cryptography, NTRU, polynomials multiplications.

1 Introduction

The recent advanced research in quantum technologies to build quantum computers, quantum communication systems [1], and Quantum devices [2,3], can provide many advantages and new solutions to the big problems in different sciences domains. For example, this technology can solve many problems in chemical, physical sciences, health, and environmental sciences, etc. The high performances of the quantum computer and the quantum communication networks can improve considerably the efficiency of machine learning applications, the Internet search engines, the data transmission, and the security of the sensitive data [4, 5].

But such evolution represents a potential threat to our currently widely deployed public-key cryptographic systems. The quantum computer can break the classical cryptosystems based on computational difficulties, such as RSA and ECDH in polynomial time. Therefore, the challenge of the cryptographic community right now is to build a Post-Quantum cryptosystem able to resist quantum computer attacks. In response, the NIST(National Institute of Standards and Technology) published, in November 2017, an official Call for Proposals and Request for Nominations for Public Key algorithm. The competition is still in process to standardize one or more post-quantum cryptosystems (PQC) [6].

Lattice-based cryptography is an exciting field of research and one of the most promising candidates for post-quantum cryptography standardization project. Besides the conjectured security against quantum attacks, lattice-based schemes tend to be algorithmically simple and highly parallelizable [7].

NTRUencrypt is one the most important post-quantum cryptosystem (Based on Lattices cryptography) candidates submitted to this competition during the first round [8]. the authors proposed three versions with different parameters NTRUencrypt443 and NTRUencrypt743 and NTRUencrypt1024 with both approaches PKE(Public Key Encryption) that achieves CCA-2 security via NAEP transformation and KEM (Key Exchange Mechanism). We notice that during the second round the authors replaced those versions by new versions with other parameters, the candidate is now namely NTRU [9].

In this work, we focused only on NTRUencrypt1024 (referred into the original document by ss-NTRU-pke and ss-NTRU-kem).

1.1 Our Work

The bottleneck for all cryptosystems based on Lattices is the complexity cost of the multiplication (polynomials multiplication, vectors multiplication, etc.) in cryptographic functions. Thus the majority of them use some algorithms to improve the performance of the cryptographic functions (like NTT, Montgomery, CRT, etc.). In particular, The *NTRUencrypt1024* used the NTT algorithm for this end [8].

We contribute by creating an improved release of the *NTRUencrypt1024*, namely *NTRUboost*, named *XKhwarizm* [10] combined to Montgomery algorithm [11] operates in the ring of the form $\mathbf{R}_q = \mathbb{Z}_q[X]/X^N + 1)$ rather than using the NTT algorithm using our own algorithm. After testing (100 times) our release, we obtained a great improvement of cryptographic functions. The result was a speed-up by a factor up to (X10) for encryption function and a speed-up by a factor up to (X11) for decryption function.

Our release used the latest KECCAK hash function, which has recently been standardized as SHA3 in FIPS202 [12], rather than SHA2 hash functions. And also, our implementation generates the polynomials according to Centred Binomial Distribution rather than Gaussian samplers [13]. We also successfully reduce the size of the public key, private key, and cipher-text from 4097 bytes to 2049 bytes by using the modulus value q = 12289 rather than the value q = 1073750017 with the same security level.

1.2 Outline

The remainder of our work is organized as follows:
Section 1: this introduction;
Section 2: preliminaries: Then we recall the necessary knowledge of the Lattices-Based-Cryptography, and a brief description of the NTT algorithm and Montgomery algorithm are presented, and we give some related works that use NTT, Montgomery, or other algorithms;
Section 3: We recall the description of the original NTRUencrypt submitted to NIST competition;
Section 4: We present our contribution by describing our *Xkhwarism* algorithm combined with the Montgomery algorithm and we describe our release $NTRUboost$;
Section 5: We present the obtained result of our implementation compared to the original one;
Section 6: Finally we give a conclusion and our future research orientation.

2 Preliminaries

In the remainder of this paper, we use the following notations: LBC for Lattice-Based-Cryptography [7]; $L_{(B)}$ Lattice of $\mathbb{R}^n$ generated by a base $\mathbf{B} = (\vec{e_1}, \dots, \vec{e_n})$, is defined by $\mathbf{L}_{(B)} = \{\vec{v} \in \mathbb{R}^n$, with $(a_1, \dots, a_n) \in \mathbb{Z}^n$ and $\vec{v} = a_1\vec{e_1} + \cdots + a_n\vec{e_n}\}$; $\mathbf{R} = \mathbb{Z}[X]/(X^N + 1)$ the ring polynomial modulus the ideal $\mathbf{f}(x) = X^N + 1$ with coefficient in $\mathbb{Z}^n$; and R_q the quotient ring polynomial with coefficients in [−q/2, q/2]; $fntt = NTT(f)$ is the NTT transformation function that returns the polynomial into NTT form and the $f = invNTT(fntt)$ is the transformation function that returns the normal polynomial form; we also note $\sum_{i=0}^{n} hntt_i = \sum_{i=0}^{n} fNtt_i.gNtt_i$ the point-wise multiplication of two NTT polynomials form and we also note it $hNtt = fNtt \circ gNtt$; we refer to sampCBD(seed) the polynomial sampled according to Centred Binomial Distribution and sampDGD(seed) the polynomial sampled according to Discrete Gaussian Distribution.

2.1 Number Theoretic Transform (NTT)

The NTT is a particular case of Discrete Fourier Transform (DFT) defined in a positive Integer group and finite fields whereas the DFT is defined in complex numbers group [14]. It is useful for polynomials multiplication and it is allowed to reduce the number of multiplications from $O(n^2)$ to $O(n * log(n))$. In fact, many LBC schemes are based on operations in polynomial rings of the form $\mathbf{R} = \mathbb{Z}[X]/(X^N + 1)$ used this method [8,9,13,15].

The Principe of NTT algorithm is as follow:

1. Having the polynomials *P(x)* of degree *n*, and Modulus *M* where $1 \leq n < M$;
2. Finding an integer $k > 1$ and search prime number such that $N = k.n + 1$ with $N \geq M$;
3. The multiplicative group F^N has size $\varphi(N) = N - 1 = k.n$ and a generator **g** ;
4. Defining the primitive nth root of unity $\omega = g^k \pmod N$ and $\omega^n = g^{kn} = g^{\omega(N)} = 1 \bmod N$.

Note that the multiplication of two polynomials with degree n and the coefficients are almost m, the bound of output is $m^2.n$ then we choose $M = m^2.n + 1$.

2.1.1 NTT function definitions

For a polynomial $f = \sum_{i=0}^{n-1} f_i X^i \in R_q$, the NTT functions are defined by the equation bellows:

$$NTT(f) = fNtt = \sum_{i=0}^{n-1} fNtt_i X^i \pmod q. \tag{1}$$

$$with\ fNtt_i = \sum_{j=0}^{n-1} \gamma^j f_j \omega^{ij} \pmod q. \tag{2}$$

where $\gamma = \sqrt{\omega}$.

The inverse of NTT function to return back to the normal form is:

$$invNTT(fNtt) = f = \sum_{i=0}^{n-1} f_i X^i \pmod q. \tag{3}$$

$$with\ f_i = n^{-1}\gamma^{-i} \sum_{j=0}^{n-1} fNtt_j \omega^{-ij} \pmod q. \tag{4}$$

2.2 Montgomery algorithm

In 1985, In his paper [11] under the title "Modular Multiplication Without Trial Division," Peter Montgomery introduced, an efficient algorithm for modular multiplication without division and increasing significantly the speed of multiplication of two numbers. More concretely it is a method to compute $a \pmod N$. And it is also suitable for computing multiplication $ab \pmod N$.

2.2.1 Algorithm description

We choose two integers R and N, with $R > N$ and $gcd(N, R) = 1$. For $0 < T < NR$ the Montgomery reduction of $T \pmod N$, is $TR^{-1} \pmod N$. The algorithm is as follow:

Algorithm 1: Montgomery reduction of $T \pmod N$.

Input: The integers T, N and R with $R = 2^k > N$ and $gcd(R, N) = 1$.

```
1. $m = T(-N^{-1}) \pmod R$;
2. $t = (T + mN)/R$
3. $if(N < t)t = t - N$.
```

Output: The product $t = T \pmod N$.

The clam is : $t = TR^{-1} \pmod N$; $tR = T \pmod N$ with $0 < t < N$

$$TR^{-1} = (T + T(-N^{-1})(modR)N)/R \pmod N. \tag{5}$$

For the multiplication of two integers a and b modulus N, $ab \pmod N$, by using the Montgomery reduction the process of the algoritm is as follow:

Algorithm 2: The Montgomery modular multiplication.

```
Input: The integers $a, b$ $N$, and $R$ with $R = 2^k > N$ and $gcd(R, N) = 1$.
  1. $a' = aR \pmod N$; $b' = bR \pmod N$
  2. $N' = N^{-1} \pmod R$;
  3. $c' = (a'b')R^{-1} \pmod N$ and $c = c'R^{-1} \pmod N$
  4. $c' = (a'b')R^{-1} = (a'b' + a'b'(-N') \pmod R)N)/R \pmod N$;
  5. $c = c'R^{-1} = (c' + c'(-N') \pmod R)N)/R \pmod N$
Output: The product $c = ab \pmod N$.
```

The clam is : $c = ab \pmod N$.

$$c'R^{-1} = (a'b')R^{-1}R^{-1} = (a'R^{-1})(b'R^{-1}) = ab \pmod N. \tag{6}$$

2.3 Related Works

There are many candidates submitted to NIST who used one or more multiplication algorithms to improve the performance of their cryptosystems. In this section, we present a citation of two important cryptosystems that used the NTT algorithm and the Montgomery algorithm to increase the cryptographic functions.

2.3.1 E. Alkim et al. work (NewHope)

The NewHope cryptosystem is a new version of Ring-LWE(Ring Learning With Error) created by Lyubashevsky, Peikert et Regev and published in 2010 [13]. The Domain of NewHope is the Ring polynomials of the form $\mathbf{R_q} = \mathbb{Z}_q[X]/X^N + 1)$ and all the polynomials are chosen according to the Centred Binomial Distribution rather than the preview version where the polynomials are chosen according to the Discrete Gaussian Distribution [13]. This cryptosystem use only KEM “Key Exchange Mechanism” scheme, and it adopts the NTT algorithm combined with Montgomery Algorithm for increasing the speed performance. The author argues that “the choice of q = 12289 as a prime number, allows an (NTT) to be faster and more efficient”. This cryptosystem is already integrated by GOOGLE in its new release of Google Chrome browser namely “CANARY”.

2.3.2 Jeffry Hoffstein et al. work (FALCON)

FALCON is a lattice-based signature scheme [15]. It is based on NTRU assumption and also submitted to the NIST competition. The same as

NewHope, to improve the speed performance of the system, the authors of FALCON used NTT algorithm and the Montgomery algorithm. The FALCON domain of computation is the ring polynomials of the form $\mathbf{R_q}$ $=\mathbb{Z}_q[X]/X^N + 1)$. The degree N is normally a power of two (typically 512 or 1024) and q is a specific small prime (the recommended parameter is $q = 12289$).

3 NTRU Cryptosystem

3.1 A View of NTRU

NTRU was created in 1996 by the three mathematicians J. Hofstein, J. Pipher, and J. H. Silverman, and published in 1998 [7]. It is completely constructed on Lattice-Based-Cryptography. NTRU was considered reliable by the IEEE P1363.1 standard and in April 2011, NTRUEncrypt was accepted in the X9.98 standard. Its domain of computation is the ring polynomials of the form $\mathbf{R_q}$ $=\mathbb{Z}_q[X]/X^N - 1)$ with N prime number and q power of two, or the ring polynomials of the form $\mathbf{R_q}$ $=\mathbb{Z}_q[X]/X^N + 1)$ where N is power of two and q prime number. There are several versions since its creation and the latest versions are the NTRUencrypt candidate submitted to NIST during the first round and the NTRU candidate submitted during the second round. In terms of security, NTRUencrypt resisted for 20 years of cryptanalysis.

3.2 Lattice Problems and Security

The schemes constructed from lattice-based cryptography typically enjoy a worst-case hardness guarantee [16, 17]. That warrants very strong security. It is based on mathematical concepts and theories to encrypt and decrypt, as well as to demonstrate the complexity and the difficulty of breaking those cryptographic systems.

3.2.1 Lattice problems

Many cryptanalysis works are performed, their principal goal was to check the robustness of the Lattices-Based Cryptography, by posing the hardest problems on points lattices in $\mathbb{R}^n$, and the best tools used to prove the security is Lattice reduction (Gram-Schmidt, LLL, BKZ algorithms) and Meet-in-The-Middle attack (MIM) [16].

The principal Lattice problems are SVP and CVP defined as follow:

The Shortest Vector Problem (SVP): Finding (SVP) in Lattice $L_{(B)}$ is finding a non-zero vector that minimizes the Euclidean norm. Formally the problem SVP is to find a non-zero vector:

$$\tilde{\mathbf{v}} \in \mathbf{L}_{(\mathbf{B})} \quad \forall \tilde{\mathbf{x}} \in \mathbf{L}_{(\mathbf{B})} \quad we\ have\ \|\tilde{\mathbf{v}}\| \leq \|\tilde{\mathbf{x}}\|. \tag{7}$$

The Closest Vector Problem (CVP): Given the Lattice $L_{(B)}$ and a vector $\vec{w} \in \mathbb{R}^n$ to find a vector $\tilde{\mathbf{v}} \in \mathbf{L}_{(\mathbf{B})}$ "Closest" to $\vec{w}$, is to find a vector $\vec{v} \in L_{(B)}$ that minimizes the Euclidean norm $\|\vec{w} - \vec{v}\|$ where:

$$\|\vec{w} - \vec{v}\| = \mathbf{min}\{\|\vec{w} - \vec{v}\| / \tilde{\mathbf{v}} \in \mathbf{L}_{(\mathbf{B})}\}. \tag{8}$$

The NTRU assumption is defined by: "Having $h = g/f$ it is hard to find f and g, and it can be reduced to the *uniqueSVP* for the NTRU lattices" [16].

3.2.2 NTRU security

In fact, mathematicians often estimate the projected security of cryptographic systems by plotting the evolution in "running time" and "space requirements" of the best-known attacks according to the level of security needed.

For measuring the security level of Lattice-Based Cryptosystems, Martin R. Albrecht et al. developed an estimator as described in the paper under the title "Estimate all the LWE, NTRU schemes" [18]. This tool helps the researchers in this area to check the security level of their cryptosystems. The estimator gives the result as follows:

1. The security level of NTRUencrypt1024 with parameters $\{q = 1073750017, n = 1024\, p = 2\}$, provides 256 bits for classical security and 198 bits for quantum security [18].
2. And for the sequence parameters $\{q = 12289, n = 1024\}$ used by FALCON, the NTRU assumption provides 263 bits for classical security and 230 for quantum security [15]. We note that our "NTRUboost" release is based on the same NTRU assumption and uses the same sequence parameters.

In its report[8240], NIST states that the security of NTRU is based on stronger assumptions than LWE or RLWE Lattice-based schemes [19].

It is important to note that IND-CPA security only models passive adversaries. Active adversaries are modeled using IND-CCA security (Indistinguishably under a Chosen-Ciphertext Attack). There are efficient and generic techniques to upgrade IND-CPA schemes to IND-CCA security [20].

3.3 Description of NTRUencrypt1024 Post-Quantum Cryptosystem

The NTRUencrypt candidate [8] submitted to NIST competition during the first round with three versions: The *NTRU443* and the *NTRU743* versions both use Karatsuba multiplication algorithm to increase the speed performance, but the *NTRUencrypt1024* release uses the NTT algorithm. We also note that the NTRUencrypt uses the two approaches, the public key encryption **(PKE)** scheme and Key Exchange Mechanism *(KEM)* scheme.

In this subsection, we describe the original cryptographic algorithms of NTRUencrypt1024, and for illustrating our contribution and our improvement, we modified the original algorithms by introducing the use of the NTT functions, the sampling function sampDGD(seed), and the use of SHA-2 hash function. Fore more details, the reader can see the original documents in [8].

3.3.1 Parameters

As described by the authors, the cryptosystem uses the sequence parameters $\{N = 1024, q = 1073750017, p = 2, \sigma = 724\}$ and *seed* and the domain of NTRUencrypt1024 is the Ring polynomials of the form $\mathbf{R_q}$ $=\mathbf{Z_q}[X]/X^N + 1)$. We also note that this version of NTRUencrypt takes the private key F in the form $F = p * f + 1$ [21]. This form allows us to avoid the computation of the inverse of $f \pmod p$ because $F = p * f + 1 \pmod p = 1$.

Algorithm 3: Keys Generation.

```
Input : the sequence parameters $\{N = 1024, q = 1073750017, p = 2, \sigma = 724\}$ and seed.
1. f, g ← sampDGD(seed) ;
2. FNtt ← Ntt(F=p*f+1);
3. gNtt ← Ntt(g);
4. $F_qNtt \leftarrow inverse(FNtt) \pmod q$;
5. $hNtt \leftarrow F_qNtt \circ gNtt$;
Output: the public key hNtt and the private key FNtt.
```

In the keys generation function we remark that in line2 and line3 we transform the polynomial (F, g) from normal form to NTT form*(FNtt, gNtt)*, and in line4, we compute the inverse *FqNtt* of *FNtt* by just computing the inverse of the *FNtt* coefficients modulus *q*, and in the last line, we compute the public key *hNtt* in the NTT form by the point wise multiplication (○) of *FqNtt and gNtt* polynomials.

Algorithm 4: Encryption .

```
Input: The public key hNtt, the message with its length msg,len, and the seed
1.  $m \leftarrow$ padding(msg,len) ;
2.  rseed $\leftarrow HashSHA2(m\|hNtt)$ ;
3.  $r \leftarrow$ sampDGD(rseed) ;
4.  $e \leftarrow$ sampDGD(rseed) ;
5.  rNtt $\leftarrow$ Ntt(r) ;
6.  eNtt $\leftarrow$ Ntt(e) ;
7.  tNtt $\leftarrow rNtt \circ hNtt \pmod q$ ;
8.  tseed $\leftarrow$ HashSHA2(tNtt) ;
9.  mask $\leftarrow$ sampDGD(tseed) ;
10. $M \leftarrow m - mask \pmod p$ ;
11. cNtt $\leftarrow tNtt + p * eNtt + M$ ;
Output: The cipher-text cNtt.
```

The computation used by the NTT transformation function of the polynomial *r* in line5, and the *r* in line6 and the point wise multiplication ($\circ$) of *rNtt* and *hNtt* in line7 are very costly, whereas the computational cost of the other lines are less important.

Algorithm 5: Decryption .

```
Input: The private key FNtt and the cipher-text cNtt.
1.  MNtt $\leftarrow FNTT \circ cNtt \pmod q$ ;
2.  tNtt $\leftarrow cNTT - MNtt$ ;
3.  tseed $\leftarrow$ HashSHA2(tNTT) ;
4.  mask $\leftarrow$ sampDGD(tseed) ;
5.  maskNtt $\leftarrow$ Ntt(mask) ;
6.  mNtt $\leftarrow MNtt + maskNtt \pmod p$ ;
7.  rseed $\leftarrow$ HashSHA2(mNtt—hNTT) ;
8.  $m \leftarrow$ inverseNtt(mNtt) ;
9.  $r \leftarrow$ sampDGD(rseed) ;
10. rNtt $\leftarrow$ Ntt(r) ;
11. eNtt $\leftarrow p^{-1} * (tNtt - rNtt \circ hNtt)$ ;
12. $e \leftarrow$ inverseNtt(eNtt) ;
13. result $\leftarrow$ check(e) ;
14. msg,len $\leftarrow$ Extract(m).
Output: the message msg and its length $len$.
```

In the Decryption function, the most cost time is taken by the point wise multiplication ($\circ$) of the private key *FNtt* and the cipher-text *cNtt* in line1, the transformation of the *mask* from normal form to NTT form *maskNtt* in line5, and also the inverse transformation of *mNtt* from the NTT form to normal form *m* as in line8 and the transformation of *r* from normal form to NTT form in line10, and the computation of *e, eNtt* in line11 and line12.

4 Our Contribution

Our contribution consists of the creation of a new version of *NTRUencrypt1024*, named *NTRUboost* with great improvement of speed performance of encryption and decryption functions. The principal improvements of our implementation are:

1. We used our multiplication algorithm, named *XKhwarizm* combined to Montgomery algorithm operates in the ring of the form $\mathbf{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$. Our algorithm used the multiplication and the modulus reduction($\%$) as the principal operations. The complexity of the polynomial coefficients multiplication is $O(n^2)$ but the complexity of modulus reduction of the polynomial coefficients is only *O(n)*, and our algorithm does not need any pre-computed data when NTT algorithm needs to pre-compute two arrays (powers of the root of unity), and for increasing the modulus reduction we used the Montgomery algorithm.
2. The polynomials are sampled according to Centred Binomial Distribution (sampCBD() function) rather than Discrete Gaussian Distribution (sampDGD() function). Additionally, the implementation of this sampling algorithms is much easier to protect against timing attack and does not decrease the security", see [13].
3. Our release integrates the new standard SHA-3 (Keccak hash functions) rather than SHA-2, of course, that also increases the security and prevent in the event of a fault of SHA-2 [12].
4. And we can use alternatively the modulus $q = 12289$ like NewHope [13] and FALCON schemes [15] rather than using the big modulus $q = 1073750017$. Both provide almost the same security level, and the same speed performance, as we are going to present in the next subsection.
5. Our implementation reduces the size of the public key, the private key, and the cipher-text from 4097 bytes to 2049 bytes.

In this section, we describe the *XKhwarizm* algorithm and our $NTRUboost$ cryptosystem release.

We note that our release implementation is developed with the C++ programming language rather than the C programming language as in the original implementation, and the test was performed in the platform PC-TOSHIBA –Satellite, Processor Intel, Core™i7 -2630QM CPU, 2GHz, RAM 8GO, under environment Windows 7-32 bits and Dev-C++ 4.9.9.2.

4.1 XKhwarizm Algorithm Description

The domain of our multiplication algorithm combined to Montgomery algorithm is the polynomials ring $\mathbf{R_q}$ $=\mathbb{Z}_q[X]/X^N+1)$.

This algorithm allows us to multiply two polynomials f and g modulus the integer q with degree less than N for each polynomial, and we obtained directly a polynomial h reduced to a degree less than N in $\mathbf{R_q}$. Formally that means:

$$f(x)=\sum_{i=0}^{N-1} f_i x^i \bmod q, \quad g(x)=\sum_{i=0}^{N-1} g_i x^i \bmod q. \tag{9}$$

$$h(x)=\sum_{i=0}^{N-1} h_i x^i \bmod q, \quad Xkhwarizm(f,g)=h \bmod q. \tag{10}$$

Algorithm 6: Polynomial multiplication algorithm in R_q .

Input: Polynomials *f and g*, with their degrees less than *(N)*, and *q*;
Function ***Xkhawarism (f,g)*** **:**

```
1.  . for : int $i = 0$ to $n - 1$ do :
2.  ... if $f_i! = 0$ then
3.  ...... for : int $j = 0$ to $n - 1$ do :
4.  ......... if $g_j! = 0$ then :
5.  ............ if$((i + j) < N)$ then : $h_{i+j} \leftarrow h_{i+j} + f_i.g_j$ ;
6.  ............ else : $h_{i+j-N} \leftarrow h_{i+j-n} + f_i.g_j$ ;
7.  ............ endif
8.  ......... endif
9.  ...... endfor
10. ... endif
11. .endfor
12. for : integer $i = 0$ to $n$ do: $h_i \leftarrow Montgomery(h_i, q)$;
```

End-function.
Output: The result polynomial of product *h*;

Our algorithm used two principal operations the multiplication and the modulus($\%$). The polynomial coefficients multiplication with the complexity cost $O(n^2)$ performed from line2 to line12, but the complexity cost of modulus is only *O(n)* as in line13, used Montgomery reduction.

4.2 Benchmarking Between NTT and XKhwarizm Algorithms

We did a benchmarking between our proposal XKhwarism algorithm and NTT algorithm by using the parameters of NTRUencrypt1024 and NTRUboost releases. To do so, we compare the cost of the polynomials multiplication of two polynomials $\mathbf{X}$ and $\mathbf{Y}$ generated in the ring $\mathbf{R_q} = \mathbf{Z_q}[X]/X^N+1)$ with the sequences parameters $\{N = 1024, q = 1073750017\}$ and $\{N = 1024, q = 12289\}$ respectively.

For using NTT functions we computed, the root[1024] and the inverse of the root[1024], for more details the reader can see the original implementation of *NTRUencrypt1024* at NIST website [8]. The modulus q = 12289 is recommended by the authors of NewHope and FALCON candidates also submitted to NIST competition and the value of modulus q = 12289 increases the security by minimizing the probability of decryption failure.

In their latest report [8240] [19], NIST experts did not appreciate the big modulus q = 1073750017.

In this case, we show that our algorithm is about 10 times faster than NTT algorithm, but in practice, we not always compute the invNTT() function in cryptographic algorithms, the authors of NTRUencrypt used the public key, the private key, and the cipher-text in NTT form as presented by the Algorithm.2 and the algorithm.3, the invNTT function is called only at the end of decryption function to product the plain-text in normal form.

4.3 Our NTRUboost Post-quantum Cryptosystem Description

In this part, we describe the proposed cryptosystem *NTRUboost* release of *NTRUencrypt1024*. And for illustrating our contribution and our

Table 1 Speed benchmarking between NTT algorithm and our algorithm of two polynomials multiplication Z=X*Y. (ms)

NTT Algorithm			Time	Time
q = 1073750017	Time	Xkhwarizm Algorithm	q = 1073750017	q = 12289
XNtt	31 ms	Z=Xkhwarizm(X,Y)	9 ms	9 ms
YNtt	30 ms	...	...	
$ZNTT = XNtt \circ YNtt$	1 ms	...	...	...
$Z = InvNTT(ZNtt)$	30 ms	...	...	...
Total cost	92 ms	...	9 ms	9 ms

improvement, we modified the original algorithms of keys generation, encryption, and decryption by introducing our *Xkhwarizm(.)* function, the sampling function *sampCBD(seed)*, and the use of SHA-3 hash function, the algorithm discription of our release is as follow:

Algorithm 7: Keys Generation.

```
Input : the sequence parameters $\{N = 1024, q = 12289, p = 2\}$ and seed.
1.  $f,g \leftarrow sampCBD(seed)$ ;
2.  $F_q \leftarrow inverse(F=p*f+1)$;
3.  $h \leftarrow Xkhwarizm(F_q, g)$;
Output: the public key $h$ and the private key $F$.
```

Our keys generation algorithm contains just 3 lines. We replace the three NTT functions Ntt(F), Ntt(g) and their point-wise product as in the original algorithm, only by single-function $Xkhwarizm(F_q, g)$ without pre-computed data, when we must pre-computing the roots and the inverse of the roots arrays for using NTT functions.

Algorithm 8: Encryption .

```
Input: The public key $h$, the message with its length msg,len, and the seed
1.  $m \leftarrow padding(msg,len$ ;
2.  $rseed \leftarrow HashSHA3(m—h)$ ;
3.  $r \leftarrow sampCBD(rseed)$ ;
3.  $e \leftarrow sampCBD(rseed)$ ;
4.  $t \leftarrow Xkhwarizm(p * r, h)(modq)$ ;
5.  $tseed \leftarrow HashSHA3(t)$ ;
6.  $mask \leftarrow sampCBD(tseed)$ ;
7.  $M \leftarrow m - mask \pmod p$ ;
8.  $c \leftarrow t + M + p * e$ ;
Output: The ciphertext $c$.
```

The same for our encryption algorithm, we replace the NTT transformation functions and the inverse NTT transformation functions, and the point-wise product as in the original algorithm, by only $Xkhawarizm(.)$ function. Then, all the polynomials are computed in the normal form, and we reduced the code size from 11 lines to 8 lines and we don't need any pre-computation data.

Algorithm 9: Decryption .

```
Input: The private key $F$ and the ciphertext $c$.
1.  $M \leftarrow Xkhwariem(F, c) \pmod q$ ;
2.  $t \leftarrow c - M$ ;
3.  $tseed \leftarrow HashSHA3(t)$ ;
4.  $mask \leftarrow sampCBD(tseed)$ ;
5.  $m \leftarrow M + mask \pmod p$ ;
6.  $rseed \leftarrow HashSHA3(m—h)$ ;
7.  $r \leftarrow sampCBD(rseed)$ ;
8.  $e \leftarrow p^-1(t - Xkhwarizm(r, h))$ ;
9.  $msg,len \leftarrow Extract(m)$.
Output: the message msg and its length len.
```

Our decryption algorithm uses $Xkhwarizm(.)$ function in line1 and line8, it replaces the NTT transformation functions as in the original decryption Algorithm 3, especially the point-wise multiplication, and the inverse NTT transformation $(inverseNtt(.))$. We reduced the code size from 14 lines to 9 lines with no pre-computation.

5 Analysis and Result of Our NTRUboost Implementation

In this section, we present our result by doing a benchmarking between our NTRUboost implementation and the original version of NTRUencrypt1024. We note that all the parameters fixed by the authors are kept. We exchange only the NTT transformation functions by our Xkhwarizm functions and the use of the Montgomery algorithm.

And we used sampCBD() function to generate the polynomials according to Centred Binomial Distribution inspired by NewHope implementation [13] which integrates the SHA3 (shake128, shake256) hash functions, therefore, we included the library files (fips202.c,fips202.h...).

Our implementation reduce the size of the public key, private key, and the cipher-text, as we updated in parameter file "param.h" by SECRETKEYBYTES=2049; PUBLICKEYBYTES=2049; CIPHERTEXTBYTES=2049. Both cryptosystem implementations described in this paper are available on the website of at Sources Code.

We note that Both implementations are performed in the platform PC-TOSHIBA–Satellite, Processor Intel, Core™i7 2630QM CPU, 2GHz, RAM 8GO, under environment Windows 7-32 bits and Dev-C++ 4.9.9.2.

5.1 Results and Analysis

After simulation of our NTRUboost and NTRUencryption, by executing 100 times each implementation with the parameters sets $\{N = 1024, q = 1073750017, p = 2\}$ and $\{N = 1024, q = 12289, p = 2\}$ alternatively, we obtained drastic result by increasing the performance of the cryptographic functions. We reported the median values of the processes key generation, encryption, and decryption, in Table 2, Table 3, and in Figure 1, in Milliseconds by using the platform cited above:

We remark in Table 2, that for the sequence parameters $\{N = 1024, q = 1073750017, p = 2\}$, our NTRUboost achieves $13ms$ and $22ms$ for encryption and decryption functions respectively, when the NTRUencrypt1024 achieves $140ms$ and $240ms$ for encryption and decryption functions respectively. The same result is obtained, as presented in Table 3, by using the sequence parameters $\{N = 1024, q = 12289, p = 2\}$.

We improving the speed performance by factors up-to $\times 10$ for the encryption function, and up-to $\times 11$ for the decryption function as it is illustrated in Figure 1.

In terms of the speed performance, the impact of the value size of parameter q is so negligible when performing the cryptographic functions of NTRUboost and NTRUencrypt by using the parameters q = 12289 and q = 1073750017 alternatively. But in terms of the memory size needed; the sizes of cipher-text, plain-text and keys are 2 times greater when we use the modulus $q = 1073750017$ rather than using the modulus $q = 12289$.

But, as we showed in Table 2, Table 3, and as illustrated in Figure 1 below, the key generation function is not improved, because for NTRU schemes we should computing the inverse polynomial of the private key (see Algorithm.7 line.2), which is very costly in terms of the time by using the extended Euclidean algorithm. The speed of key generation algorithms of our release and NTRUencrypt are almost the same values $80ms$ and $85ms$ respectively. In conclusion, we did better by implementing our $Xkwarizm(.)$ in the cryptographic functions of our NTRUboost post-quantum cryptosystem.

Table 2 Speed performance benchmarking between NTRUboost and NTRUencrypt1024 (ms) with q = 1073750017

Schemes	KeysGen	Encryption	Decryption
NTRUencrypt1024	80 ms	140 ms	240 ms
NTRUboost	85 ms	13 ms	22 ms
Speed-up	1 $times$	10 $times$	11 $times$

Table 3 Speed performance benchmarking between NTRUboost and NTRUencrypt1024 (ms) with q = 12289

Schemes	KeysGen	Encryption	Decryption
NTRUencrypt1024	80 ms	140 ms	240 ms
NTRUboost	85 ms	13 ms	22 ms
Speed-up	1 $times$	10 $times$	11 $times$

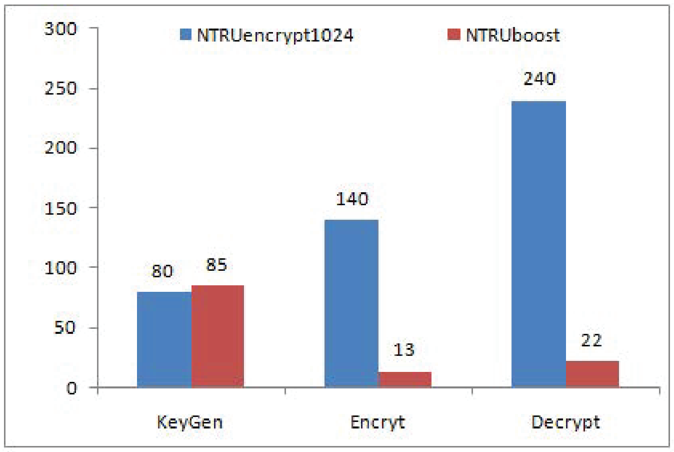

Figure 1 Speed performance benchmarking between our NTRUboost and NTRUencrypt1024 (in Milliseconds).

6 Conclusion

In this paper, we evoked that the cryptographic community is very worried about the security of the systems and the persons' private life when the quantum computer will be generalized, and the necessity to build robust post-quantum cryptosystems able to resist eventual quantum attacks.

In this context, we contribute by improving the performance of NTRUencrypt which were candidates to the NIST standardization project. We create a boosted release of NTRUencrypt post-quantum cryptosystem that uses our "XKhawarism" algorithm combined with the Montgomery algorithm rather than using NTT algorithm. The result obtained after executing both implementations with the same parameters set proof that the performance of our NTRUboost is greater than the original NTRUencrypt by a factor up to $\times 10$ for the encryption and decryption functions.

We also improve the security performance by using the latest standard of hash functions SHA-3, we reduce the rate of the decryption failure by using small modulus integer; we reduce the code size of the software, and our implementation does not use the pre-computed data which reduce the size of the implementation source code.

We not that we can use alternatively our implementation with the modulus parameter $q = 12289$ or $q = 1073750017$ or other prime numbers, by just modify this value in parameters file "param.h" of the software.

The limitation of our solution is that we can't improve the key generation function because the computation of the inverse of the private key (f), which uses the euclidean algorithm, is very expensive.

For our future works, we hope to improve our $Xkhwarism$ polynomial multiplication algorithm which operates in the ring $\mathbf{R_q} = \mathbf{Z_q}[X]/X^N + 1)$, for adapting it to parallel computation and fully exploiting the computing power offered by modern processors, by inspiring from Chmielowiec.A work [22].

References

[1] A. Abd EL-Latifab, B. Abd-El-AttyaSalvador E. Venegas-Andracac, W. Mazurczykd. Efficient quantum-based security protocols for information sharing and data protection in 5G networks. *Future Generation Computer Systems Volume 100, November 2019, Pages 893-906*, 2019.

[2] B. Abd-El-Atty, A. Abd El-Latif, E. Venegas-Andraca. An encryption protocol for NEQR images based on one-particle quantum walks on a circle. *Quantum Information Processing September 2019 https://doi.org/10.1007/s11128-019-2386-3*, 2019.

[3] B. Abd-El-Atty, A. M. Iliyasu, A. Alanezi, A. A. Abd El-latif. Optical image encryption based on quantum walks, *Optics and Lasers in Engineering. Volume 138, 2021, 106403, ISSN 0143-8166, https://doi.org/10.1016/j.optlaseng.2020.106403.*

[4] L. Li, B.abd-El-Atty, A.Ghoeim. Quantum color image encryption based on multiple discrete chaotic systems. *Proceedings of the Federated Conference on Computer Science and Information Systems vol 11, pp. 555–559, 2018.*

[5] A. A. A. El-Latif, B. Abd-El-Atty, W. Mazurczyk, C. Fung and S. E. Venegas-Andraca. Secure Data Encryption Based on Quantum Walks for 5G Internet of Things Scenario. *in IEEE Transactions on Network and Service Management, vol. 17, no. 1, pp. 118-131, March 2020, doi: 10.1109/TNSM.2020.2969863.*

[6] G. Chen, S. Jordan, D. Moody, L. Yi-Kai, R. Peralta, R. Perlner and D. Smith. NISTIR 8105-Report on Post-Quantum Cryptography. Gaithersburg, Washington USA, 2016.

[7] J. Hofstein, J. Pipher, and J. H. Silverman. Introduction Mathematics and Cryptography NTRU. Book. Wilmington USA, 1998.

[8] C. Chen, O. Danba, J. Hoffstein, A. Hulsing, J. Rijneveld, John M. Schanck, P. Schwabe, W. Whyte, Z. Zhang. NIST PQ submission: NTRUencrypt A Lattice-Based encryption algorithm. Brown University and Onboard security company, Wilmington USA, 2017.

[9] C. Chen, O. Danba, J. Hoffstein, A. Hulsing, J. Rijneveld, John M. Schanck, P. Schwabe, W. Whyte, Z. Zhang. Algorithm Specifications And Supporting Documentation. Brown University and Onboard security company, Wilmington USA, 2019.

[10] El. Laaji, A. Azizi, S. Ezzouak. An improvement of NTRU-1024 performance by speeding-up polynomial multiplication. *Smart2019 International conference*, MOROCCO, 2019.

[11] By Peter L. Montgomery. Modular Multiplication Without Trial Division. Math comput. USA 1985.

[12] G. Bertoni, J. Daemen, M. Peters, G. Van Assche, R. Van. Keccak Hash algorithm. Radboud University, 2016.

[13] E. Alkim, L. Ducas, T. Poppelman, P. Schwabe. Post-quantum key exchange" "New Hope". Department of Mathematics, Ege University, Turkey, 2017.

[14] Nayuki Project. Number-Theoric-Transform (Integer DFT). website.

[15] C. Chen, O. Danba, J. Hoffstein, A. Hulsing, J. Rijneveld, J. M. Schanck, P. Schwabe, W. Whyte, Z. Zhang. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU. Brown University and Onboard security company, Wilmington USA, 2019.

[16] J. Hofstein1, Jill Pipher, John M. Schanck, Joseph H. Silverman1, W. Whyte, Z. Zhang. Choosing Parameters for NTRUEncrypt. Wilmington USA, 2016.

[17] M. Hartmann. The Ajtai-Dwork Cryptosystem and Other Cryptosystems Based on Lattices. University de Zurich,29 October 2015.

[18] M. R. Albrecht,R. Benjamin, B Curtis. Estimate all the fLWE, NTRU schemes. USA May 2, 2018.

[19] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, Y. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, D. Smith-Tone. Status Report NISTIR 8240 on the First Round of the NIST

Post-Quantum Cryptography Standardization Process. Gaithersburg, Washington USA 2019.

[20] A. Poppelen, L. Ducas, G. Tel. Cryptographic decoding of the Leech lattice. Utrecht University, 2016.

[21] M. Rao Mamdikar, V. Kumar, D. Ghosh. Enhancement of NTRU public key. National Institute of Technology, Durgapur 2013.

[22] A.Chmielowiec. Parallel Algorithm for Multiplying Integer Polynomials and Integers. *IAENG Transactions on Engineering Technologies Lecture Notes in Electrical Engineering, vol. 229, str. 605-616, 2013*. https://link-1springer-1com-1q12c56zv0182.eczyt.bg.pw.edu.pl/chapter/10.1007/978-94-007-6190-2_46

Biographies

El Hassane Laaji. Engineer in Science computer and Ph.D student at Mohammed First University Oujda Morroco, Science Faculty, Arithmetic, Science computation and Application Laboratory (ASCAL).

Abdelmalek Azizi. Professor and Director of Arithmetic, Science computation and Application Laboratory (ASCAL), Science Faculty, Mohammed First University Oujda Morroco.