Alejandro Royher Rodríguez Rodríguez

Bachelor's thesis

Implementing Lattice-Based Cryptography

June 2022

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Bachelor's thesis

2022

Alejandro Royher Rodríguez Rodríguez

Implementing Lattice-Based Cryptography

Bachelor’s thesis
June 2022

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

The thesis is about implementation of LWE/MLWE encryption schemes in C++ and the use of the Number Theoretic Transform (NTT) in order to get faster multiplication operations over certain rings. We explore how much speed up we get by using NTT in our different schemes, comparing performance of our NTT implementation with the *MulMod* function of the *NTL Library* and the schoolbook multiplication. We use Cooley-Turkey algorithm in the NTT forward step and Gentleman-Sande algorithm in the NTT inverse step with ordinary modular reduction. After some experiments we came to the conclusion that, when we are working on $R_q = \mathbb{Z}_q[X]/(X^N + 1)$, for relative small N with our NTT implementaion we get a performance similar compared to the $MulMod$ function and for all N we get much better results compared to the schoolbook multiplication.

Contents

Chapter 1

Introduction

Over the last years, lattice-based cryptography has been gaining more and more importance and popularity due to the fact that it is supposed to be resistant against quantum computers. In a groundbreaking work, Ajtai [1] gave the first worst-case to average-case reductions for lattice problems, and with them the first cryptographic object with a proof of security assuming the hardness of well-studied computational problems on lattices. In particular, Ajtai's work gave the first cryptographic function based on a standard worst-case complexity assumption of any kind. Ajtai introduced the (average-case) "short integer solution" (SIS) problem and its associated one-way function, and proved that solving it is at least as hard as approximating various lattice problems in the worst case. In a subsequent work from 1997, Ajtai and Dwork [1] gave a lattice-based public-key encryption scheme. In a concurrent work with Ajtai's in 1996, Hoffstein, Pipher, and Silverman [2] devised the public-key encryption scheme NTRU (also known as NTRUEncrypt). This was the first cryptographic construction using polynomial rings, which is most usefully interpreted in terms of *algebraically structured lattices*. In a work published in 2010, Lyubashevsky, Peikert, and Regev [3] introduced ring-LWE, the ring-based analogue of learning with errors [4]. One of the most expensive steps in these types of schemes is the polynomial multiplication over certain rings. In order to make this step faster we take advantage of the Discrete Fourier Transform (DFT) that is called Number Theoretic Transform (NTT) in our case since we work over finite fields. This algorithm is used with the idea of fully splitting the ring so we have a faster multiplication, we are going to test it with our 2 different encryption schemes: Basic LWE Scheme and Module LWE Scheme.

These 2 schemes work with non-binary messages i.e. modulo p over the ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$. The complete implementation[1] is done using C++ making use of the *NTL: A Library for doing Number Theory* [5] .

[1]All the code can be found in the repository `https://github.com/alexroyher/LWECode`.

Chapter 2

Background

In the following sections we will be using [6] and [4] as references for definitions.

2.1 Public Key Cryptosystem

A public key cryptosystem consist of 3 algorithms with different inputs and outputs:

- **Key Generation**: It has a parameter λ, it is the process of generating as outputs a pair of secret key $\boldsymbol{SK}$ and a public key $\boldsymbol{PK}$.
- **Encryption** : It takes as input a message $\boldsymbol{M}$ to be encrypted and the public key $\boldsymbol{PK}$. It gives as output a ciphertext $\boldsymbol{C}$.
- **Decryption** : It takes as input a ciphertext $\boldsymbol{C}$ and a secret key $\boldsymbol{SK}$. It gives as output a message $\boldsymbol{M}$.

Given *Enc* and *Dec* algorithms to encrypt and decrypt a message, we need, for a cryptosystem to be valid, that it satisfies $Dec(Enc(M,PK)),SK) = M$ for any message M and $(SK,PK) = KeyGen(\lambda)$ generated by the Key Generation algorithm.

This let us to define another important concept like the security of a scheme.

2.2 Security Definition

To introduce the CPA security we need first to know what *indistinguishability* means. Let's see what an indistinguishability security game would be.

- Key Generation algorithm is run and it gives as output a pair (PK,SK)
- An adversary $\mathcal{A}$ is given the public key PK and he chooses a pair of messages (M_0,M_1) of equal length.
- A bit $b \leftarrow \{0,1\}$ is chosen randomly and is encrypted $C \leftarrow Enc(M_b,PK)$ and this ciphertext is given to $\mathcal{A}$.
- Then, $\mathcal{A}$ outputs a bit b' and we say that $\mathcal{A}$ succeeds if $b' = b$ (and the result of the experiment is 1) and he fails otherwise (the resultt of the experiment is 0).

Now, before defining what is a *CPA*-secure scheme, we will give a brief definition of what an indistinguishable encryption is.

Definition 1: A public key encryption scheme $\prod$ has indistinguishable encryptions to an eavesdropper if for all probabilistic polynomial-time adversaries A there is a negligible function negl $(\cdot)$ s.t

$$Pr\Big[\Big|PubKey^{EAV}_{\mathcal{A},\prod}(n) = 1\Big|\Big] \leq \frac{1}{2} + negl(n)$$

where $PubKey^{EAV}_{\mathcal{A},\prod}(n)$ is the eavesdropping indistinguishability experiment.

Now we are able to give a right definition of the CPA security.

Definition 2: A public key encryption scheme $\prod$ is CPA secure if and only if it has single-message indistinguishable encryptions.

2.3 The LWE Problem

In order to test the application of the NTT on lattice-based cryptography we will make use of the previous mentioned schemes. These schemes are actually based on the following problem:

Definition 3: For a set S, we write $a \leftarrow S$ to mean that a is chosen uniformly at random from the set . In addition, for any positive integer β, we will define the set:

$$[\beta] = \{-\beta, \ldots -1, 0, 1, \ldots, \beta\}$$

Definition 4: For positive integers $m, n, q, and \beta \ll q$, the $LWE_{n,m,q,\beta}$ problem asks to distinguish between the following two distributions:

1. $(A, As + e)$, where $A \leftarrow \mathbb{Z}_q^{n\times m}$, s $\leftarrow [\beta]^m$, $e \leftarrow [\beta]^n$
2. (A, u) , where $A \leftarrow \mathbb{Z}_q^{n\times m}$ and $u \leftarrow \mathbb{Z}_q^n$

Chapter 3

NTT

This chapter is largely based on the paper [7]. The Number Theoretic Transform is heavily based on the Chinese Remainders Theorem. Let $m_1, m_2, ..., m_k$ be pairwise coprime integers such that $\prod_{i=1}^{k} m_i = M$. Then the system of equations

$$x = a_1 \; mod \; m_1$$

$$x = a_2 \; mod \; m_2$$

$$...$$

$$x = a_k \; mod \; m_k$$

has a unique $x \in \mathbb{Z}_M$ that solves the equations. This version of the Chinese Remainders Theorem is related to coprime integers [8].

Theorem 1: Let $f(X) \in \mathbb{Z}_q[X]$ be so that $f(X) = \prod_{i=1}^{k} f_i(X)$ where $< f_i(X) > + < f_j(X) > = \mathbb{Z}_q[X] \; \forall i \neq j \in \{1, ..., k\}$. Then

$$\mathbb{Z}_q[X]/ < f(X) > \cong \prod_{i=1}^{k} \mathbb{Z}_q[X]/ < f_i(X) >$$

To define the NTT algorithm we first need to know what is FFT and DFT.

Definition 5: The Discrete Fourier Transform (DFT) is the discrete version of the Fourier Transform (FT) that transforms a signal (or discrete sequence) from the time domain representation to its representation in the frequency domain.

Definition 6: The Fast Fourier Transfrom (FFT) is an algorithm for evaluating the traditional DFT, for complex valued vectors $(f_0, \ldots, f_{N-1})$ of length N which is defined over the complex field $\mathbb{C}$ using the complex root of unity of order N. Here, we have

$$F(\lambda) = \sum_{k=0}^{N-1} f_k e^{2\pi i \lambda k/N} = \sum_{k=0}^{N-1} f_k \xi_N^{\lambda k}, \quad \xi_N := e^{2\pi i/N}, \lambda = 0, 1, \ldots, N-1$$

Such a complex root of unity exists for all N, however the FFT is more efficient if N is composite, the highest efficiency being obtained for $N = 2^m$, for some integer m.

Now, we have problems with DFT: For cryptography we work with finite objects, and can actually obtain full precision which does not apply in practice in the complex field, since the argument of the complex root of unity is irrational and exact arithmetic is impossible in general.

The NTT's, whether based on a finite field or integer roots of unity modulo certain rings, give us full precision (complex DFTs cannot do this and can't be used for crypto), and are evaluated in the native ring that is used in cryptography.

With parameters N being a power of 2 and q a prime with $q \equiv 1 \mod 2N$, let $a = (a[0], ..., a[n-1]) \in \mathbb{Z}_q^N$, and let ω be a primitive $N-th$ root of unity in $\mathbb{Z}_q$, which means that $\omega^n \equiv 1 \mod q$.

The forward transformation $\tilde{a} = NTT(a)$ is defined, as stated in Patrick Longa and Michael Naehrig, as

$$\tilde{a} = \sum_{j=0}^{N-1} a[j] w^{ij} \mod q$$

for $i = 0, \dots, n-1$. In the same paper is stated that the inverse is computed as $b = INTT(\tilde{a})$ where:

$$b = \frac{1}{n} \sum_{j=0}^{N-1} \tilde{a}[j] w^{-ij} \mod q$$

for $i = 0, \dots, n-1$. Since applying the NTT transform as described above to our target scheme provides a cyclic convolution, computing $c = a \cdot b \mod (X^N + 1)$ with two polynomials a and b would require applying the NTT of length $2N$ and thus N zeros to be appended to each input; this effectively doubles the length of the inputs and also requires the computation of an explicit reduction modulo $X^N + 1$. As stated in the paper, to avoid these issues, one can exploit the negative wrapped convolution.

Letting ϕ be a primitive $2N-th$ root of unity in $\mathbb{Z}_q$ such that $\phi^2 = \omega$, and let $a = (a[0], ..., a[n-1])$, $b = (b[0], ..., b[n-1]) \in \mathbb{Z}_q^N$ be two vectors. Also, we define

$$\tilde{a} = (a[0], \phi a[1], \dots, \phi^{n-1} a[n-1])$$
$$\tilde{b} = (b[0], \phi b[1], \dots, \phi^{n-1} b[n-1])$$

The negative wrapped convolution of a and b is defined as

$$c = (1, \phi^{-1}, \phi^{-2}, \dots, \phi^{-(n-1)}) \circ INTT(NTT(\tilde{a}) \circ NTT(\tilde{b}))$$

where $\circ$ denotes component-wise multiplication and where $INTT(NTT(a)) = a$.

Now, let's took a look at how the Forward/Inverse NTT steps are performed. As mentioned in the introduction, we will use *Coley-Tukey NTT Algorithm* for the forward step and *Gentleman-Sande Inverse NTT Algorithm* fot the inverse step. These both algorithms and its pseudocode can be found in the referenced paper [9]. Below we show how both algorithms have been implemented in C++.

First of all, we need the $2N$-th root of unity given the modulus q and N. To achieve this, we have the following functions:

```
/* This function just checks if r^k is congruent with 1 mod N */
bool existSmallN(ZZ r, ZZ M, ZZ N){
        for (int k = 2; k<N; k++){
                if (PowerMod(r,k,M) == 1){
                        return true;
                }
        }
        return false;
}
```

```
INPUT:
    - q : Modulus of the field
    - N : The order of the root of unity we want to find
OUTPUT:
    - The N-th root of unity
ZZ NthRootOfUnity(ZZ q, ZZ N){
        assert((q-1) % N == 0);
        ZZ phi = q-1;
        srand(time(NULL));
        while(true){
                ZZ alpha = ZZ(rand() % conv<int>(q) + 1);
                ZZ beta = PowerMod(alpha,phi / N, q);
                if(!existSmallN(beta,q,N)){
                        return ZZ(beta);
                }
        }
}
```

Before performing the NTT forward step, first we need to compute the powers of the $2N$-th root of unity but since we are using the iterative version of the NTT algorithm we have to store them in bit reversed order. So given a root g, while the natural ordering of its powers would be $[g^0, g^1, g^2, \cdots g^{n-1}]$, in bit reverse order the element g^i would actually be stored at an index in the table found by reversing the m bits in i where $N = 2^m$. Similarly, we also have to store the inverse powers of the root in bit reversed order for the inverse step.

To perform the bit-reverse operation we have the following 2 auxiliary functions:

```
INPUT:
    - n : A number which bits we want to reverse
    - len : Number of bits that are needed to represent ’n’
OUTPUT:
    - res : Number resulting of reversing the bits of ’n’.
ZZ reverseBits(ZZ n, ZZ len) {
    ZZ res = ZZ(0);
    for (int i = 0; i < len; i++) {
        if (conv<int>(n) & (1 << i))
            res |= 1 << (conv<int>(len) - 1 - i);
    }
    return res;
}
```

```
INPUT:
    - a : The array we want to bit reverse
    - N_bit : log2(N)
OUTPUT:
    - a : Array with elements in bit reversed order.
ZZX orderReverse(ZZX a, ZZ N_bit){
        for(int i=0;i<deg(a);i++){
                ZZ rev_i = reverseBits(ZZ(i), N_bit);
                if (rev_i > i){
                        ZZ coef = a[conv<int>(rev_i)];
                        a[conv<int>(rev_i)] = a[i];
                        a[i] = coef;
                }
        }
        return a;
}
```

The forward step is performed in the following way:

```
INPUT :
    - a : The polynomial we want to apply the NTT forward step.
    - q : Modulus of the field we are working on, it satisfies q = 1 mod 2N
    - N : Length of the polynomial ’a’ (we work on Z_q[X]/X^N+1)), it is a power of 2
OUTPUT:
    - a : Polynomial ’a’ once the NTT forward step is performed
ZZX FNTT(NTTScheme& NTT,ZZX a, ZZ q, ZZ N){
        int t = (conv<int>(N)/2);
        int m=1;
        while(m<N){
                int k = 0;
                for (int i=0;i<m; i++){
                        ZZ S = NTT.power_of_roots[m+i];
                        for (int j=k; j<k+t; j++){
                                ZZ U =a[j];
                                ZZ V =a[j+t]*S % q;
                                a[j] = (U+V) % q;
                                a[j+t] = (U-V) % q;
                        }
                        k = k+2*t;
                }
                t = t/2;
                m = 2*m;
        }
        return a;
}
```

Analogously, we have that the inverse step is performed in the following way:

```
INPUT:
    - a : The polynomial we want to apply the NTT inverse step
    - q : Modulus of the field we are working on, it satisfies q = 1 mod 2N
    - N : Length of the polynomial 'a' (we work on Z_q[X]/X^N+1)), it is a power of 2

OUTPUT:
    - a : Polynomial 'a' once the NTT inverse step is performed

ZZX INTT(NTTScheme& NTT,ZZX a, ZZ q, ZZ N){
        ZZ inv_N = InvMod(N,q);
        int t = 1;
        ZZ m = (N/2);
        while(m>0){
                int k=0;
                for (int i=0;i < m; i++){
                        ZZ S = NTT.inv_power_of_roots[conv<int>(m)+i];
                        for (int j=k;j<k+t; j++){
                                ZZ U = a[j];
                                ZZ V = a[j+t];
                                a[j] = (U+V) %q;
                                ZZ W = (U-V) %q;
                                a[j+t] = (W*S) % q;
                        }
                        k = k+2*t;
                }
                t = 2*t;
                m = m/2;
        }
        for (int i=0;i <N; i++){
                a[i] = (a[i] * inv_N ) % q;
        }
        return a;
}
```

Chapter 4

Encryption

4.1 Basic LWE Encryption Scheme

We now will define our 2 encryption schemes before using NTT in order to later have a clear idea about how the NTT influences in the process of encryption/decryption.

We start defining the **Basic LWE$_{\mathbf{n,m,q},\beta}$ Encryption Scheme** that encrypts a message message $M \in \mathbb{Z}_p^{k\times l}$. This definition is partially based on [10] and [11]. The key generation works in the following way:

$$\mathbf{sk} : S \leftarrow [\beta]^{m\times l} \quad , \quad \mathbf{pk} : (A \leftarrow \mathbb{Z}^{m\times m}, t = AS + pE_1)$$

where $E_1 \leftarrow [\beta]^{m\times l}$. Now , to encrypt message M the encryptor chooses $R, E_2 \leftarrow [\beta]^{k\times m}$ and $E_3 \leftarrow [\beta]^{k\times l}$. Then we have the following ciphertext

$$(\mathbf{U} = RA + pE_2 \quad , \quad \mathbf{V} = RT + pE_3 + M).$$

Now, given the ciphertext (U, V) and the secret key S the decryption proceeds as follows

$$\mathbf{V\text{ - }US} = RT + pE_3 + M - (RA + pE_2)S = p(RE_1 + E_3 - E_2S) + M$$

Then the $(i, j)^{th}$ coefficient of $V - US$ is

$$\mathbf{output}_{i,j} = p(r^T e_1 + e_3 - e_2^T s) + M_{i,j}$$

if we compute

$$\mathbf{M_{i,j}} = \begin{cases} (output_{i,j}) \mod p & \text{if output}_{i,j} \le q/2 \\ (output_{i,j} - q) \mod p & \text{if output}_{i,j} > q/2 \end{cases}$$

we get the original message back. We note that in the output message we have some noise that can be handed if the following inequality holds

$$\beta^2 m + \beta < \frac{q}{2p}$$

[10]. Basically, every term in the *output* expression is bounded by β since they are chosen from a space $[\beta]^{p\times t}$ for some p and t and we see that the space size p of the field we are working on is multiplying all these elements, this explains the p in the right hand denominator. All the details about the inference of this inequality are carefully explained in the cited paper.

Now, based on this scheme, we will give a definition of the other scheme that is quite similar to the *Basic LWE Encryption Scheme* with the difference that matrices do not contain just numbers. Instead, matrices operates with polynomials.

4.2 Module LWE Encryption Scheme

This section is largely based on [10]. The polynomial ring $(\mathbb{Z}[X], +, \times)$, with an indeterminate X, consists of elements of the form

$$a(X) = \sum_{i=0}^{\infty} a_i X^i$$

for $a_i \in \mathbb{Z}$ with the usual polynomial addition and multiplication operations. In the Basic LWE Encryption Scheme we worked with the ring $(\mathbb{Z}, +, \times)$. A generalization of this ring with which we will be working with is the ring $(R_f, +, \times)$, where $f \in \mathbb{Z}[X]$ is a monic polynomial of degree N. The elements of R_f are the polynomials

$$a(X) = \sum_{i=0}^{d-1} a_i X^i$$

Now, for a vector $a \in R_f^m$, we write $a \leftarrow [\beta]^m$ to be the distribution in which every coefficient of every polynomial in a is chosen uniformly from $[\beta]$. Analogously, we will be working over the ring $R_{q,f}$, which is like the ring R_f except that the polynomial coefficients are in $\mathbb{Z}_q$ rather than in $\mathbb{Z}$.

For positive integers $m, n, q, \beta \ll q$, and ring $R_{q,f}$, the $\mathbf{R_{q,f}LWE_{n,m,\beta}}$ problem asks to distinguish between the following two distributions:

1. $(A, As + e)$, where $A \leftarrow R_{q,f}^{n\times m}$, s $\leftarrow [\beta]^m$, $e \leftarrow [\beta]^n$
2. (A, u) , where $A \leftarrow R_{q,f}^{n\times m}$ and $u \leftarrow R_{q,f}^n$

Now we can define the **Module $\mathbf{LWE_{n,m,\beta}}$ Encryption Scheme**. The main advantage of the scheme will be that the message μ, being in R_f and whose coefficients are in $\mathbb{Z}_p$, allows us to pack d bits into it. The encryption and decryption steps work similar to the basic scheme commented above so details won't be written again.

It is important to know that in this scheme we can optimize the polynomial multiplication by using our NTT multiplication in every product of polynomials instead of using the school book multiplication.

Chapter 5

Experiments

5.1 Setting up the experiments environment

Since we want to do experiments about how fast we can perform the multiplication of polynomials, it is important to vary the degree of the polynomials we are working with (i.e the N parameter) and the field modulus of the ring we are working (i.e the q parameter).

The parameter q has to satisfy $q \equiv 1 \mod 2N$, so it is enough to vary the N parameter. We will choose random polynomials on these scenarios and perform different experiments. The computer where the experiments will take place has the following characteristics:

- 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
- CPU MHz: 2800.000
- CPU max MHz: 4700,0000
- CPU min MHz: 400,0000
- CPU(s): 8
- Thread(s) per core: 2
- Core(s) per socket: 4

Since the polynomials we are working with are random, if we run, for every parameter N and every scheme among schoolbook multiplication, our NTT implementation and $MulMod$ (of the NTL Library) we would not get a *real* measure of the performance of every scheme. To achieve a more precise measure, we will run every scheme with every different parameter N 10 times and we will take the mean.

Our main objective is to test if, as the parameter N varies, the execution time difference between schemes varies as well. It is important to know that parameters like the dimension of the matrices used in every scheme will be kept constant keeping in mind that our goal.

5.2 Running the experiments

In the following table are shown the results of the experiments for every parameter N. Note that, for every N a parameter q is computed. Measures are taken in seconds so it is easy to compare.

Since what we want is to analyze how the performance is affected by the increment of N, we kept the following parameters constant:

- m = 32
- n = 32
- p = 3

Where m and n are the number of rows and columns of the public key matrix A and p indicates the space of the message we are working with, i.e. since $p = 3$ we perform the encryption and decryption of polynomials with coefficients in $\mathbb{Z}_3$.

N	**q**	**MulMod**	**NTT**	**School book**
256	7681	1.60	3.62	7.76
512	12289	1.70	4.99	29.61
1024	12289	1.91	8.22	116.7
4096	12289	4.40	39.53	-
8192	65537	9.91	86.55	-
16384	65537	15.25	178.45	-

As we can see, from a certain point ($N = 1024; q = 12289$) performing school book multiplication becomes infeasible and so we stop using it since we have already enough information about its performance.

Note that, knowing that q has to satisfy $q \equiv 1 \mod 2N$ we can compute it in the following way:

```
long generate_prime(long N){
        long k=1;
        while(!(isPrime(ZZ(k*N+1)))){
                k++;
        }
        return k*N+1;
}
```

And once we have set the N, we can use the function above to generate our parameter q:

```
q = generate_prime(2*N);
```

In order to get another perspective of the results, we plot the results obtained in the table in form of plot.

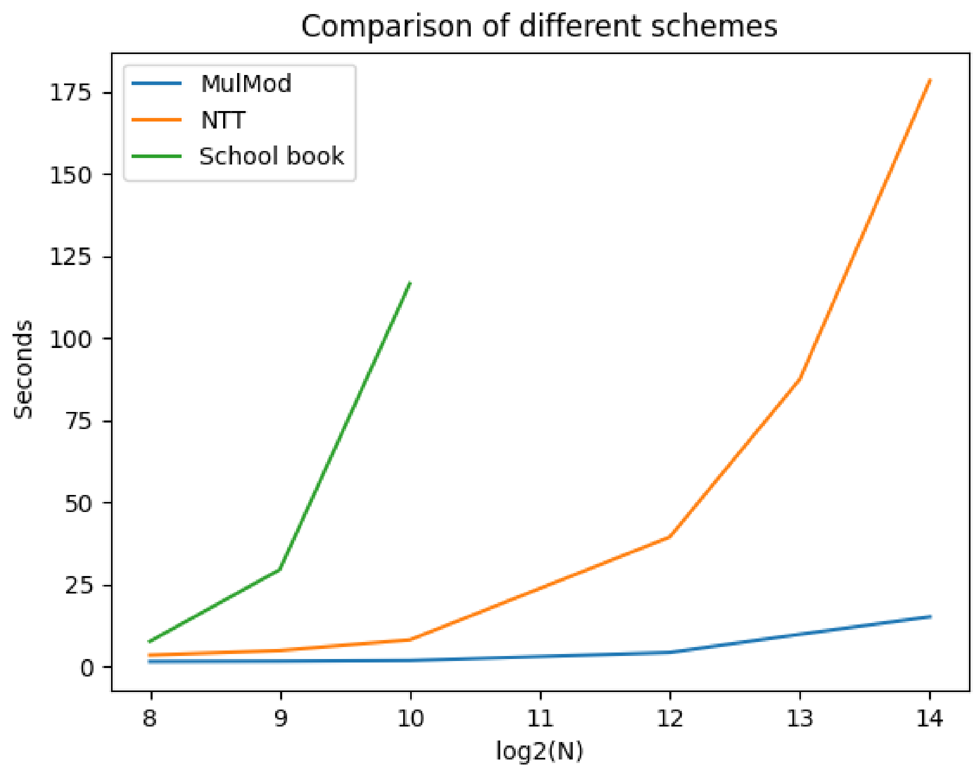

Figure 5.1: Seconds vs log2(N) of the different schemes

Chapter 6

Conclusion

To begin with the conclusion, we should highlight that we have used ordinary modular reduction (i.e. with the C++ operator '%') and it is highly possible that the *NTL Library* function $MulMod$ implements some optimized modular reduction.

Despite this, we can clearly observe that for relative small N and q the $MulMod$ function and our implementation of NTT perform similarly. From $N = 2^{10} = 1024$ the difference between this 2 functions are bigger and bigger. Whereas our NTT implementation increase in an exponential way as the N increase, the $MulMod$ function keeps a reasonable low time as the N increases.

On the other hand, from the beginning we can observe that the school book multiplication is, by far, the worst way to perform the polynomial multiplication. Its time increases really fast as the N increases, making even infeasible to perform more experiments with a higher N.

Keeping in mind all of this, we can conclude that with a relative simple NTT implementation we can get a great speed up in the time. We have seen how this speed up can be used in nowadays schemes such the Module LWE.

Bibliography

[1] M. Ajtai, *Generating hard instances of lattice problems*, 2004.

[2] J. P. Hoffstein and J. H. Silverman, *Ntru: A ring-based public key cryptosystem*, 1998.

[3] C. P. V. Lyubashevsky and O. Regev., *On ideal lattices and learning with errors over rings*, 2010.

[4] C. Peikert, *A decade of lattice cryptography*, `https://eprint.iacr.org/2015/939.pdf`, 2016.

[5] V. Shoup, *Ntl: A library for doing number theory*, `https://libntl.org`, 2022.

[6] W. Stallings, *Cryptography and network security: Principles and practice*, 2010.

[7] P. Longa and M. Naehrig, *Speeding up the number theoretic transform for faster ideal lattice-based cryptography*, `https://eprint.iacr.org/2016/504.pdf`, 2016.

[8] Wikipedia, *Chinese remainder theorem : Statement*, `https://en.wikipedia.org/wiki/Chinese_remainder_theorem`.

[9] M. Scott, *A note on the implementation of the number theoretic transform*, `https://eprint.iacr.org/2017/727.pdf`, 2017.

[10] V. Lyubashevsky, *Basic lattice cryptography: Encryption and fiat-shamir signatures*, 2019.

[11] T. Silde, *Short paper: Verifiable decryption for bgv*, 2022.