On Large Polynomial Multiplication in Certain Rings

by

Khan Shagufta Shagufa

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2018

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Multiplication of polynomials with large integer coefficients and very high degree is used in cryptography. Residue number system (RNS) helps distribute a very large integer over a set of smaller integers, which makes the computations faster. In this thesis, multiplication of polynomials in ring $Z_p/(x^n + 1)$ where n is a power of two is analyzed using the Schoolbook method, Karatsuba algorithm, Toeplitz matrix vector product (TMVP) method and Number Theoretic Transform (NTT) method. All coefficients are residues of p which is a 30-bit integer that has been selected from the set of 30-bit moduli for RNS in NFLlib. NTT has a computational complexity of $O(n \log n)$ and hence, it has the best performance among all these methods for the multiplication of large polynomials. NTT method limits applications in ring $Z_p/(x^n + 1)$. This restricts size of the polynomials to only powers of two. We consider multiplication in other cyclotomic rings using TMVP method which has a subquadratic complexity of $O(n^{\log_2 3})$. An attempt is made to improve the performance of TMVP method by designing a hybrid method that switches to schoolbook method when n reaches a certain low value. It is first implemented in $Z_p/(x^n + 1)$ to improve the performance of TMVP for large polynomials. This method performs almost as good as NTT for polynomials of size 2^{10}. TMVP method is then exploited to design multipliers in other rings Z_p/Φ_k where Φ_k is a cyclotomic trinomial. Similar hybrid designs are analyzed to improve performance in the trinomial rings. This allows a wider range of polynomials in terms of size to work with and helps avoid unnecessary use of larger key size that might slow down computations.

Acknowledgements

I express my deep gratitude to my supervisor, Professor Anwar Hasan for his constant support, encouragement and patience during my graduate studies. He consistently steered me in the right direction whenever I needed it. I thank Professor Mark Aagaard and Professor Mahesh Tripunitara for reviewing this thesis.

I am grateful to Julien Eynard and Nusa Zidaric for their guidance and support. I thank my colleagues for all the discussions and motivation: Ahmed, Tanushree, Chris and Mohannad.

I must express my profound thankfulness to my parents and my brother for always believing in me, encouraging me and making it possible to fulfill my dreams. Thanks to Rameez, Sonia bhabi and my entire family for all the support.

Finally, special thanks to Niyaz and my friends here for never letting me lose hope and for all the laughs I had throughout my stay: Nivedita, Meghana, Kritika, Jacqueline, Namrah and Adi. A big shout-out to Shamma for always being there.

Dedication

This thesis is dedicated to my parents, Shahin Anjum and Shafiq Ullah Khan and my brother, Md Shafquat Ullah Khan who have always believed in me and provided unfailing support. This accomplishment has been possible only because of their blessings and sacrifices.

Table of Contents

List of Tables

List of Abbreviations

DFT Discrete Fourier Transform

FFT Fast Fourier Transform

FHE Fully Homomorphic Encryption

FPGA field programmable gate array

HE Homomorphic Encryption

LWE Learning with Error

NTT Number Theoretic Transform

RLWE Ring-Learning with Error

RNS Residue number system

SHE Somewhat Homomorphic Encryption

TMVP Toeplitz matrix vector product

Chapter 1

Introduction

1.1 Motivation

Multiplication of polynomials with large coefficients is a very common application when it comes to cryptography especially with Ring-Learning with Error (RLWE) or Somewhat Homomorphic Encryption (SHE). Homomorphic Encryption (HE) is an encryption that allows specific mathematical operations to be carried out on the encrypted data and when decrypted, the result would be the same as obtained by performing those specific mathematical operations on the data before encryption [18]. Fully Homomorphic Encryption (FHE), suggested by Rivest, Adleman and Dertouzos back in 1978, is a scheme that makes extremely complex encrypted data programs evaluable [34][7][8]. The first feasible construction of the scheme, recognized as SHE was presented by Gentry in 2009 [17][10]. RLWE is a ring variant of Learning with Error (LWE) [32] in which all computations are considered in ring $R = Z/\Phi_k$ where Φ_k is the kth cyclotomic polynomial of degree $n = \phi(k)$ [10] [22].

With the help of RNS, polynomials with extremely large coefficients are represented by a set of smaller coefficient polynomials. Arithmetic operations on all these smaller polynomials can be executed independently. *Somewhat Practical Fully Homomorphic Encryption* (FV) scheme by Junfeng Fan and Frederik Vercauteren is an efficient practical SHE scheme that handles sufficient number of operations. In 2015, a full RNS variant of the FV scheme has been presented with practical benefits of the RNS variant [5]. The polynomial multiplication involved in this SHE scheme is handled efficiently using NTT and Fast Fourier Transform (FFT) in *power-of-two* cyclotomic rings.

Cyclotomic rings quotiented in $x^n + 1$ are simple to work with because n is a power of two and by just making simple adjustments to n-dimensional FFT, arithmetic operations can be carried out efficiently [23]. Multiplication of polynomials can be carried out with quasi-linear complexity of $O(n \log n)$. For the convenience of operations, these cyclotomics are most preferred and commonly considered in recent ring-based cryptographic schemes.

It is also important to consider other cyclotomics. For efficient FHE, it is required to explore cyclotomic polynomials that are not powers of two because *power-of-two* cyclotomics fail to suffice for properties required for certain implementation features and also restricts diversity of security assumptions [23]. Another reason is that with these cyclotomics, the jump from a power of two to the next one is very distributed for large polynomials. If a desired ring size is slightly higher than a certain power of two, it will be required to consider the next power of two that might lead to unnecessary increase in runtimes.

1.2 Scope of Work

In this thesis, multiplication of polynomials of size n is presented over polynomial ring $\mathrm{Z}/(x^n + 1)$ where n is a power of two (i.e., $x^n + 1$ is the cyclotomic polynomial Φ_{2n}) and all operations are considered in the RNS. This polynomial multiplication in the RNS is useful in many cryptographic schemes that deal with huge integers. The main modulus of the RNS is the product of multiple smaller moduli. Throughout the entire thesis, residues in prime p is considered which is a 30-bit modulus from the set of moduli for the RNS used in NFLlib. Multiplication of polynomials in ring $\mathrm{Z}/(x^n + 1)$ is investigated using the schoolbook method, Karatsuba algorithm, TMVP and NTT. A comparison in terms of CPU-time is presented for the methods through software implementation.

The practicality of using Toeplitz matrix vector product is considered as it does not limit the choice of ring to $\mathrm{Z}/(x^n+1)$ only and it can be efficiently used for rings quotiented by other cyclotomic polynomials. Using schoolbook method or Karatsuba algorithm, the product of size $2n - 1$ is first computed and then adjustments are made for corresponding cyclotomics. In $\mathrm{Z}/(x^n+1)$, this adjustment is a simple subtraction of $(n+i)$-th coefficients from i-th coefficients where $0 \leq i \leq n$ but it is not as simple in other rings. Whereas with TMVP, the Toeplitz matrix for each of the rings is formed with necessary adjustments so that the products are directly modulo corresponding cyclotomic ring. Multiplication of polynomials in cyclotomic trinomials are explored using TMVP method. Efficiencies for multiplying polynomials of different sizes n where $2^h \leq n \leq 2^{h+1}$ are analyzed.

In order to demonstrate the practical feasibility and efficiency of TMVP, a hardware implementation of two-way split TMVP in ring $\mathrm{Z}/(x^n + 1)$ is carried out for polynomials of smaller sizes. The designs are synthesized in field programmable gate array (FPGA).

The objective of this thesis is to compare the efficiency of TMVP method against efficiencies of schoolbook method, NTT and the Karatsuba method in $\mathrm{Z}/(x^n+1)$ for different sizes of polynomials. All implementations are executed in software. Hybrid designs, based on the comparison are implemented to speed up TMVP. Another objective is to make use of TMVP method for multiplication in other cyclotomics in order to allow multiplication of polynomials whose sizes are not restricted to powers of 2.

1.3 Thesis Organization

This thesis is organized as follows. Chapter 2 gives an overview of the required mathematical background about finite field arithmetic. It describes different polynomial multiplication methods, RNS and its role in cryptography. Chapter 3 presents detailed explanation of each of those multiplication methods in ring $Z/(x^n + 1)$ and provides appropriate algorithms for the ease of understanding. It introduces multiplication using TMVP method in other rings quotiented by cyclotomic trinomials. Chapter 4 is an organization of the results obtained from the software implementations of polynomial multiplication. CPU-time required for the implementation in software is measured for different sizes of polynomials using each of the methods and tabulated comparisons are provided. Chapter 4 also provides estimated area in terms of LUTs and registers used for the hardware synthesis of TMVP implementation using Xilinx. Chapter 5 is a discussion based on the implementation and its analysis. It also talks about the scopes and possibilties of future work related to this thesis.

Chapter 2

Background

This chapter provides an overview of the required mathematical knowledge for the multiplication of polynomial multiplication in a given ring. The organization of the chapter is as follows: A brief introduction to the mathematical terms in finite field arithmetic that concerns our work is provided. Different methods of implementing polynomial multiplication that are examined in this thesis are explained. This chapter also identifies a number of cyclotomic polynomials that can be considered for the multiplication of polynomials using proper variants of Toeplitz matrix vector product.

2.1 Finite Field Arithmetic

2.1.1 Ring

A *ring* R is a set of elements with the binary operations of addition $(+)$ and multiplication $(-)$ satisfying the following properties.

- $(+, R)$ forms an abelian or a commutative group such that $a + b = b + a$ with 0 as the identity.
- Multiplication is associative with 1 as the multiplicative identity.
- The multiplication operation is distributive over the addition operation.

When $a \times b = b \times a$ for all elements a and b in the ring, the ring is said to be a commutative ring. If there exists an element b such that $a \times b = 1$ for an element a of the ring, then a is called a unit or an invertible element [25].

2.1.2 Polynomial Ring

A *polynomial* over the ring R can be represented in the form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

where each of the coefficients a_i belongs to the ring R and $n \geqslant 0$. If m is the largest integer for which $a_m \neq 0$, then the degree of $f(x)$ is m. All polynomials in the indeterminate x with coefficients from commutative ring R forms a *polynomial ring* $R[x]$ [25][12].

- $f(x)$ is a *constant polynomial* if it is equal to a_0 and has a degree 0.
- If the highest nonzero term, x^m has the coefficient of 1 then the polynomial $f(x)$ is a *monic polynomial.*
- $f(x) \in R[x]$ is an *irreducible polynomial* if it cannot be factorized into smaller polynomials in $R[x]$.

2.1.3 Finite Field

When all nonzero elements of a commutative ring R have multiplicative inverses, the *ring* is called a *field*. When addition or multiplication is performed on elements of a field, the resultant element is also a member of the same set. The number of elements in a field is called the order of the field [9].

nFinite field, also known as Galois Field, is a field with finite number of elements. A finite field, F_{p^m} has p^m elements where p is prime and m is a positive integer. For any prime number p, Z_p is a field and any field of order p is isomorphic to Z_p. Also, if F_q is a finite field of order $q = p^m$, then it is seen as an extension field of Z_p of degree m [25] [12][28].

2.2 Modular Reduction

In modular arithmetic, all values are reduced to or wrapped around a specific value which is called the modulus. A simple example of modular reduction is the reduction of integers modulo n. In this case all the integers limit from 0 to $n-1$ and so does the result of all arithmetic operations. Z represents all integers. Consider all integers reduced in modulo n and let that be denoted as $Z_n = Z/nZ$. So the elements in this ring are $0, 1, \ldots, (n-1)$. There are different algorithms that can be used to perform modular reduction efficiently. The classical method, Montgomery's algorithm and Barrett's algorithm are the popular ones [6].

2.3 Residue Number System (RNS)

A *residue number system* RNS enables us to represent a large integer as a set of smaller integers. The RNS comprises a set of moduli that are independent of each other and large integer is represented by its residue in each modulus. Mathematical operations are performed on each of the residue and this allows avoiding carry in addition or multiplication. This makes computations more efficient [16].

Let us consider a set of integers **B**$= \{q_1, \ldots, q_k\}$, where $\gcd(q_i, q_j) = 1,\ i \neq j$. If **B** is the base of the residue number system, then any integer x in the residue class Z_q with $q = q_0 q_1, \ldots, q_k$ is represented as a k-tuple, $(x_1, x_2, \ldots, x_k)$ where $x_i \equiv x \bmod q_i$ [37][31][27]. RLWE-based encryption in lattice based crytography involves operations on polynomials of very large coefficients. RNS can be used to represent each of those coefficients as a set of smaller integers, leading to simpler polynomial operations.

Residue Arithmetic

For any integer x and modulo q, $(-x) \bmod q = (q - x) \bmod q$. This additive inverse property is useful in dealing with subtractions. If $xy \bmod q = 1$ for any integer $0 \leq yq - 1$, then y is the multiplicative inverse of x. *Chinese Remainder Theorem* (CRT) commonly requires the use of this property.

In RNS, the result of addition and subtraction is also in reduced form with respect to the moduli. $x \pm y = \{(x_1 \pm y_1) \bmod q_1, \ldots, (x_k \pm y_k) \bmod q_k\}$. Similarly, multiplication in RNS provides the product in the corresponding modulus.

2.4 Polynomial Multiplication Methods

Lattice based cryptography uses *ideal lattices* for better performance in terms of area and speed. These lattices are *ideals* in $R = Z_p/f(x)$ where $f(x)$ is some monic polynomial of size n in Z_p and p is a prime number [10].
The product $C(x)$ of two polynomials $A(x)$ and $B(x)$ of size n is evaluated as

$$C(x) = \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} a_i b_j x^{i+j} \tag{2.1}$$

where $A(x) = \sum_{i=0}^{n-1} a_i x^i$ and $B(x) = \sum_{i=0}^{n-1} b_i x^i$.

Polynomial multiplication has a complexity of $O(n^2)$ in schoolbook and is significantly time consuming for large values of nn. Apart from the *schoolbook method*, there are various approaches for carrying out polynomial multiplication. Fast Fourier Transform, Karatsuba approach and Toeplitz Matrix Vector Product are some of the algorithms commonly used in cryptography. This section provides a basic understanding for each of the above mentioned methods. Here, multiplication using each of the following methods is executed considering ring $R = Z/(x^n + 1)$. The resultant polynomial is in modulo $x^n + 1$.

2.4.1 Karatsuba Algorithm

The Karatsuba algorithm can be used to improve the complexity of polynomial multiplication from quadratic ($O(n^2)$) to subquadratic ($O(n^{\log_2 3})$). The algorithm was originally proposed to make digital multiplication simpler. Its use in polynomial multiplication has been introduced later. Each of the polynomials to be multiplied is divided into half sized polynomials and the multiplication is replaced by three half-sized polynomial multiplications. This is repeated and divided recursively and finally the product is achieved by expansion [15] [40] [26].

Consider two polynomials A and B of size n where n is even, i.e.,

$$A = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} \text{ and}$$
$$B = b_0 + b_1 x + \cdots + b_{n-1} x^{n-1}$$

A and B are each split in half sized polynomials A_H,A_L, B_H and B_L respectively:

$$A = A_L + A_H x^{n/2} \text{ and } B = B_L + B_H x^{n/2}$$

Therefore,

$$A = (a_0 + \ldots + a_{n/2-1} x^{n/2-1}) + (a_{n/2} + \ldots + a_{n-1} x^{n/2-1}) x^{n/2} \text{ and}$$
$$B = (b_0 + \ldots + b_{n/2-1} x^{n/2-1}) + (b_{n/2} + \ldots + b_{n-1} x^{n/2-1}) x^{n/2}$$

Let K_0, K_1 and K_2 be three polynomials where

$$K_0 = A_L B_L$$
$$K_2 = A_H B_H \text{ and}$$
$$K_1 = (A_L + A_H)(B_L + B_H) - K_0 - K_2$$

Assuming that n is a power of 2, the Karatsuba algorithm is applied to multiply these half sized polynomials and in this way the algorithm repeats recursively until $n = 1$. The number of coefficient halves every recursion, hence it has a total of $\log_2 n$ recursive steps [29]. The product is then reduced modulo x^n+1 because we are considering ring $Z/(x^n+1)$. It can be done by a simple step $c_i = c_i - c_{i+n}$ for $0 \leq i \leq n-1$, where c_i represents the coefficients of the product.

2.4.2 Toeplitz Matrix Vector Product

Unlike Schoolbook method and Karatsuba algorithm, TMVP method gives us the result in ring R directly. The way the Toeplitz matrix is formed depends on the ring R and the matrix product is the residue of the product of the polynomials in the chosen ring.

Toeplitz Matrix

A Toeplitz matrix $\boldsymbol{T}$ is an $n \times n$ square matrix where entries at coordinates (i, j) and $(i+1, j+1)$ for $0 \leq (i, j) \leq n-2$ are the same. This property allows the matrix to be defined by only its $2n-1$ different entries as the rest are just repetitions [20].

$$\boldsymbol{T} = \begin{pmatrix} t_0 & t_{-1} & t_{-2} & \dots & t_{-(n-1)} \\ t_1 & t_0 & t_{-1} & \dots & t_{-(n-2)} \\ t_2 & t_1 & t_0 & \dots & t_{-(n-3)} \\ . & & & & \\ . & & & & \\ . & & & & \\ . & & & & \\ t_{n-1} & t_{n-2} & t_{n-3} & \dots & t_0 \end{pmatrix}.$$

Addition of two Toeplitz matrices only requires addition of these elements and hence cost is the same as $2n-1$ additions over the field.

Multiplication using Toeplitz Matrix

The multiplication of a $1 \times n$ vector $\boldsymbol{v}$ and an $n \times n$ Toeplitz Matrix $\boldsymbol{T}$ is described below [13].

Let

$$\boldsymbol{T} = \begin{pmatrix} T_0 & T_1 \\ T_2 & T_0 \end{pmatrix} \text{ and } \boldsymbol{V} = \begin{pmatrix} V_0 & V_1 \end{pmatrix}$$

where T_0, T_1 and T_2 are $\frac{n}{2} \times \frac{n}{2}$ Toeplitz matrices and V_0 and V_1 are $1 \times \frac{n}{2}$ matrices.

$$\boldsymbol{VT} = \begin{pmatrix} V_0 & V_1 \end{pmatrix} \begin{pmatrix} T_0 & T_1 \\ T_2 & T_0 \end{pmatrix} = \begin{pmatrix} k_2 + k_1 & k_2 + k_0 \end{pmatrix} \tag{2.2}$$

where,

$$k_0 = V_0(T_1 - T_0)$$
$$k_1 = V_1(T_2 - T_0)$$
$$k_2 = (V_0 + V_1)T_0$$

So, the multiplication of a vector of size n with an $n \times n$ Toeplitz matrix is broken down into three multiplications of vector of size $\frac{n}{2}$ with $\frac{n}{2} \times \frac{n}{2}$ Toeplitz matrix. The splitting is continued recursively until each of the sub-matrices is of size 1.

Multiplying Polynomials in Ring $\mathbf{Z}_p/(x^n + 1)$ using Toeplitz matrix

Consider vectors $\boldsymbol{A} = \begin{pmatrix} a_0 & a_1 & \dots & a_{n-1} \end{pmatrix}$ and $\boldsymbol{B} = \begin{pmatrix} b_0 & b_1 & \dots & b_{n-1} \end{pmatrix}$ representing the coefficients of the polynomials $A(x)$ and $B(x)$ of size n. Let

$$D(x) = A(x)B(x)$$
$$C(x) = A(x)B(x) \mod (x^n + 1)$$

The coefficients of $D(x)$ can be represented as a vector $\boldsymbol{D}$. The vector $\boldsymbol{D}$ is obtained by multiplying vector $\boldsymbol{A}$ of length n with an $n \times (2n-1)$ matrix $\boldsymbol{B'}$ where

$$\boldsymbol{B'} = \begin{pmatrix} b_0 & b_1 & b_2 & \dots & b_{n-1} & 0 & 0 & \dots & 0 & 0 \\ 0 & b_0 & b_1 & \dots & b_{n-2} & b_{n-1} & 0 & \dots & 0 & 0 \\ . \\ . \\ . \\ . \\ 0 & 0 & 0 & \dots & b_0 & b_1 & b_2 & \dots & b_{n-1} & 0 \end{pmatrix}$$

The matrix $\boldsymbol{B'}$ is formed from vector $\boldsymbol{B}$ such that each entry of the matrix product $\boldsymbol{D}$ represents the corresponding coefficient of polynomial $D(x)$.

$$\boldsymbol{D} = \boldsymbol{AB'} = \begin{pmatrix} a_0 & a_1 & \dots & a_{n-1} \end{pmatrix} \begin{pmatrix} b_0 & b_1 & b_2 & \dots & b_{n-1} & 0 & 0 & \dots & 0 & 0 \\ 0 & b_0 & b_1 & \dots & b_{n-2} & b_{n-1} & 0 & \dots & 0 & 0 \\ . \\ . \\ . \\ . \\ 0 & 0 & 0 & \dots & b_0 & b_1 & b_2 & \dots & b_{n-1} & 0 \end{pmatrix}$$

Since, $C(x)$ is basically $D(x)$ reduced in ring $Z_p/(x^n+1)$, all terms in $D(x)$ with degree greater than equal to n are reduced. In ring $Z_p/(x^n+1)$, $x^n = -1$, $x^{n+1} = -x$ and so on. Therefore, if d_i is the coefficient of x^i then the equivalent of $d_n x^n$ in the ring is $-d_n$.
Given,

$$D(x) = d_0 + d_1 x + d_2 x^2 + \cdots + d_{n-1}x^{n-1} + d_n x^n + \cdots + d_{2n-2}x^{2n-2} \tag{2.3}$$

then

$$C(x) = (d_0 - d_n) + (d_1 - d_{n+1})x + \cdots + (d_{n-2} - d_{2n-2})x^{n-2} + d_{n-1}x^{n-1} \tag{2.4}$$

In matrix representation,

$$\boldsymbol{C} = \begin{pmatrix} a_0 & a_1 & \dots & a_{n-1} \end{pmatrix} \begin{pmatrix} b_0 & b_1 & b_2 & \dots & b_{n-1} \\ -b_{n-1} & b_0 & b_1 & \dots & b_{n-2} \\ -b_{n-2} & -b_{n-1} & b_0 & \dots & b_{n-3} \\ . & & & & \\ . & & & & \\ . & & & & \\ . & & & & \\ -b_1 & -b_2 & -b_3 & \dots & b_0 \end{pmatrix}.$$

Here, the $n \times n$ square matrix is a Toeplitz matrix allowing the evaluation of $\boldsymbol{C}$ done using the recursive method of Toeplitz matrix vector product [13]. The computational complexity of multiplication of polynomials by this method is subquadratic ($O(n^{\log_2 3})$) [19].

2.4.3 Number Theoretic Transform (NTT)

Fast Fourier Transform (FFT) algorithm is a fast way to perform polynomial multiplication. It requires lesser operations than the other methods described earlier in this chapter. It has a quasi-linear complexity of $O(n \log n)$ [10].

The Discrete Fourier Transform (DFT) is a reversible procedure for mapping in time series. The coefficients of the DFT can be calculated by iteration. FFT is an efficient method for the computation of DFT of a time series [11] which makes the computation fast.

DFT over a finite field F_p is defined as the NTT over a ring. NTT does not use complex numbers or complex arithmetic as it is in finite field. Let us represent a polynomial $A(x)$ as vector $\boldsymbol{A} = (a_0, \ldots, a_{n-1})$ where $a_i \in Z_p$ and consider ω to be the n-th root of unity. Then we can define $NTT_\omega(A)$ as:

$$A_i = \sum_{j=0}^{n-1} a_j \omega^{ij} \mod p \tag{2.5}$$

and inverse Number theoretic Transform $INV\text{-}NTT_{\omega^{-1}}(A)$ is defined as:

$$a_i = n^{-1} \sum_{j=0}^{n-1} A_j \omega^{-ij} \mod p \tag{2.6}$$

where $0 \leq i \leq n-1$ [10].

Polynomial Multiplication using NTT

The procedure to perform polynomial multiplication requires us to calculate the FFT of the coefficients of each of the polynomial multiplicants, followed by component-wise multiplication. The inverse FFT of the component-wise product gives us the polynomial product of the two initial polynomials.

Let $\boldsymbol{A} = (a_0, \ldots, a_{n-1})$ and $\boldsymbol{B} = (b_0, \ldots, b_{n-1})$ be the vector representations of polynomials $A(x)$ and $B(x)$ of size n. In order to find their polynomial product we have to take the extended vectors $\tilde{\boldsymbol{A}} = (a_0, \ldots, a_{n-1}, 0, \ldots, 0)$ and $\tilde{\boldsymbol{B}} = (b_0, \ldots, b_{n-1}, 0, \ldots, 0)$, each of length $2n-1$.

$$\boldsymbol{AB} = INV\text{-}NTT_{\omega^{-1}}(NTT_\omega(\tilde{A}) \circ NTT_\omega(\tilde{B})) \tag{2.7}$$

where $\circ$ represents component-wise multiplication and ω is the $2n$-th root of unity [10].

Polynomial Multiplication using NTT in ring $Z/(x^n + 1)$

$\boldsymbol{AB}$ is the polynomial multiplication of vectors of size n so it has a length of $2n - 1$. We are dealing with polynomial ring $R = Z/(x^n + 1)$ so we need to reduce the product vector accordingly. One of the advantages of using NTT for polynomial multiplication in ring R is that we can use negative wrap convolution to take into account the modular reduction and avoid the use of the extended vectors of twice the input length [10]. In ring $Z[x]/(x^n + 1)$, x$^n = -1$. When $\boldsymbol{A} = (a_0, \ldots, a_{n-1})$, negative wrapping of $\boldsymbol{A}$ is given by $\boldsymbol{A'} = (a'_0, \ldots, a'_{n-1})$ where $a'_i = a_i\phi^i$ and $\phi^2 = \omega \mod p$. Similarly, let $\boldsymbol{B'}$ and $\boldsymbol{C'}$ represent the negative wrapping of $\boldsymbol{B}$ and $\boldsymbol{C}$. Then we can say, $\boldsymbol{C'} = INV\text{-}NTT_{\omega^{-1}}(NTT_{\omega}(\boldsymbol{A'}) \circ NTT_{\omega}(\boldsymbol{B'}))$. The coefficients c'_i of vector $\boldsymbol{C'}$ are then multiplied by ϕ^{-i} to obtain vector $\boldsymbol{C}$ where $\boldsymbol{C} = \boldsymbol{AB} \mod Z_p/(x^n + 1)$ [10] [21] [36][30][33].

2.5 Multiplication in Cyclotomic Polynomial Ring using RNS

A *cyclotomic polynomial* Φ_k is a monic polynomial whose roots are the primitive kth roots of unity and has a degree of $\phi(k)$ where ϕ is the Euler totient function [4]. NFLlib is a library in C++ designed for ring $Z/(x^n + 1)$ and dedicated to ideal lattice cryptography. It uses RNS to store the polynomial coefficients which means it breaks up the polynomial with extremely large coefficients into a set of polynomials with coefficients that are within machine word size. It uses NTT for computations in lattice cryptography as all computation is in ring $Z/(x^n + 1)$ [2]. This library provides sets of moduli for 16-bits, 32-bits and 64-bits representation.

In this thesis, we are working with different methods of polynomial multiplication in ring

$Z/(x^n+1)$ and in RNS base. We implement all our multiplications considering residues in one of the moduli from the set of moduli provided by NFLlib for 32-bits integer coefficients. So all our implementation results are not more than 32-bits.

2.5.1 Other Cyclotomic Polynomials

Even though it is easy to work with cyclotomic polynomials of two nonzero coefficients, i.e., of form $x^n + 1$ where $n = 2^h$, it is also essential to consider other forms. For large values of n, the next possible cyclotomic polynomial with two nonzero coefficients will have a huge increase in the value of the degree. If the desired security level requires ring which is larger than a certain 2^h but much smaller than 2^{h+1} then this restriction leads to the use of unnecessarily long key sizes and runtimes [23]. NTT used in lattice based cryptography for polynomial multiplication in ring $Z_p/(x^n + 1)$ is the fastest amongst all three methods that we are dealing with. But NTT limits the application to only $Z_p/(x^n + 1)$ where n is a power of two. TMVP method keeps the scope of extending the application to rings with other cyclotomic polynomials. Few of the other cyclotomic polynomials are mentioned below and are grouped as Class I to Class VI.

I. $\Phi_k = x^{n'} + 1, \quad n' = 2^h, \quad k = 2^{h+1}$

II. $\Phi_k = x^{2n'} + x^{n'} + 1, \quad n' = 3^i, \quad k = 3^{i+1}$

III. $\Phi_k = x^{4n'} + x^{3n'} + x^{2n'} + x^{n'} + 1, \quad n' = 4.5^j, \quad k = 5^{j+1}$

IV. $\Phi_k = x^{2n'} - x^{n'} + 1, \quad n' = 2.2^h 3^i, \quad k = 2^{h+1}.3^{i+1}$

V. $\Phi_k = x^{4n'} - x^{3n'} + x^{2n'} - x^{n'+1}, \quad n' = 4.2^h 5^j, \quad k = 2^{h+1}.5^{j+1}$

VI. $\Phi_k = x^{8n'} + x^{7n'} - x^{5n'} - x^{4n'} - x^{3n'} + x^{n'} + 1, \quad n' = 8.2^h 3^i 5^j, \quad k = 2^{h+1}.3^{i+1}.5^{j+1}$

In the next chapter, we discuss multiplication in rings Z/Φ_k where $\Phi_k = x^{2n'} + x^{n'} + 1$ or $x^{2n'} - x^{n'} + 1$ for the values of n' as mentioned for Class II and IV above.

2.6 Summary

In this chapter, we have given a brief introduction to the neccessary mathematical background including details about finite field arithmetic, modular reduction, residue number system, polynomial multiplication and some basic introduction to the different ways of multiplying polynomials in ring $Z/(x^n + 1)$. Some cyclotomic polynomials are mentioned which can possibly be used as qoutient for other rings.

Chapter 3

Multiplication of Polynomials in $\mathbf{Z}_p/\Phi_k(x)$

This chapter provides a comparison amongst the efficiencies of different methods of implementing polynomial multiplication of n coefficients where n is a power of 2. The focus is on polynomials of degrees as high as $2^{11} - 1$ or more. Residue number system is used to store imformation within machine word size. Each coefficient is reduced in modulo q_i which is of 30 bits. Polynomial multiplication is performed in ring $Z/(x^n + 1)$.

An approach is made to expand the size of the polynomials beyond powers of two by utilizing rings quotiented by certain special cyclotomic polynomials. Toeplitz matrices are evaluated for the multiplication using TMVP method in two other rings quotiented by cyclotomic trinomials. Corresponding procedures, algorithms and computational complexities are discussed in this chapter.

All implementations were done in modulo $q = 1073479681$, one of the 30 bits long moduli from the RNS modulus in NFLlib. The software implementation can be carried on of each of the members q_i of the RNS base as moduli. The RNS base considered here is a product of 291 30 bits long integers. The base B consisting of the 30 bits long moduli is mentioned in the appendix. All the implementations can be repeated with any of the moduli and their linked variables. All the data together will represent the encrypted information it carries.

3.1 Polynomial Multiplication in ring $Z/(x^n + 1)$

The irreducible polynomial for the ring is chosen to be $x^n + 1$, it allows to make use of the property $x^n \equiv -1$. This property enables us to simplify the polynomial multiplication as

$$A(x)B(x) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (-1)^{\lfloor \frac{i+j}{n} \rfloor} a_i b_j x^{(i+j \bmod n)} \tag{3.1}$$

Even with this property, the complexity of multiplication using schoolbook method remains quadratic requiring n^2 multiplications and $(n-1)^2$ additions or subtractions to evaluate the result. Different methods of polynomial multiplication are implemented in $C{+}{+}$ to compare their efficiencies in software. $g{+}{+}$ compiler is used in linux machine to compile the codes and $clock()$ function to record the time taken by each of the processes. Karatsuba and TMVP are recursive methods which require lesser number of multiplication than the general schoolbook method. These recursive methods are more effective for polynomials of higher degrees so we have also implemented some hybrids, where normal schoolbook method for multiplication is carried our for smaller values of n where n is the size of the polynomial and switch to recursive method for higher values.

3.1.1 Polynomial Multiplication using the Schoolbook Method

Schoolbook method is the most straightforward way of multiplying two polynomials. When we multiply two polynomials A and B of size n and m respectively, the product is of size $n + m - 1$. In this method, each of the coefficients of A is multiplied with each of the coefficients of B and therefore, it requires a total of nm multiplications. The complexity of multiplication by this method is $O(nm)$.

$$C = A \times B = A \times \sum_{i=0}^{m-1} b_i x^i$$

where b_i represents the coefficients of polynomial B.
Algorithm 1 represents multiplication of two polynomials with n coefficients by schoolbook method. The multiplication is in the ring $Z/(x^n + 1)$ so the algorithm includes the modular reduction process in the last step. The conditions of the modular reduction step would be different for different cyclotomic polynomials.

So, the complexity to multiply two polynomials of size n by schoolbook method is $O(n^2)$.

Algorithm 1 Polynomial Multiplication by Schoolbook Method in ring $Z/(x^n + 1)$

Input: A, B, n, q

```
initialize: c_i = 0 for 0 ≤ i ≤ 2n − 2
for i = 0 to (n − 1) do
    for j = 0 to (n − 1) do
        c_{i+j} ← c_{i+j} + (a_i × b_j)
    end for
end for
for i = 0 to (n − 2) do
    c'_i ← (c_i − c_{n+i})
end for
c'_{n−1} ← c_{n−1}
return C'
```

3.1.2 Multiplication in $Z/(x^n+1)$ using Karatsuba Algorithm

A simple example for multiplication of two polynomials using the Karatsuba algorithm is given below. Let $A = 5 + 10x + 9x^2 + 4x^3$ and $B = 10 + 8x + 3x^2 + 9x^3$. Then,

$$A = (5 + 10x) + (9 + 4x)x^2$$
$$B = (10 + 8x) + (3 + 9x)x^2$$

Therefore,

$$k_0 = (5 + 10x)(10 + 8x)$$
$$k_2 = (9 + 4x)(3 + 9x)$$
$$k_1 = \big((5 + 9) + (10 + 4)x\big)\big((10 + 3) + (8 + 9)x\big) - k_0 - k_2$$

For the three half sized polynomial multiplications we repeat the procedure. That is, we divide k_0 to evaluate three smaller products k_0', k_1' and k_2' which we then recombine to form k_0. Similarly we also evaluate k_1 and k_2. So, for $k_0 = a_0 \times b_0 = (5 + 10x)(10 + 8x)$,

$$k_0' = 50,\ k_2' = 80,\ k_1' = 140$$

Therefore, $k_0 = 50 + 140x + 80x^2$ and similarly, $k_2 = 27 + 93x + 36x^2$ and $k_1 = 105 + 187x + 122x^2$

$$A \times B = k_0 + k_1x + k_2x^2 = 50 + 140x + 185x^2 + 187x^3 + 149x^4 + 93x^5 + 36x^6$$
$$C = A \times B \mod (x^4 + 1) = -99 + 47x + 149x^2 + 187x^3$$

Using the same concept, multiplication of polynomials of higher degrees can be performed with increasing number of recursions.

The asymototic complexity of the Karatsuba algorithm depends on the number of coefficients of the polynomials. For the number of coefficients $n = 2^h$, the Karatsuba algorithm requires $O(n^{\log_2 3})$ basic arithmetic operations. Assuming that $n = m^l$ where and m and l are integers, the number of additions and multiplications required by the Karatsuba algorithm can be generalized as follows [40]

$$\text{number of multiplications} = (\frac{1}{2}m^2 + \frac{1}{2}m)^l = n^{\log_m(\frac{1}{2}m^2+\frac{1}{2}m)} \tag{3.2}$$

$$\text{number of additions} = u.n^{\log_m(\frac{1}{2}m^2+\frac{1}{2}m)} - 8n + v \text{ where } u \leq 6 \text{ and } v \leq 3 \tag{3.3}$$

The Karatsuba algorithm requires smaller number of multiplications and additions compared to schoolbook method when m is small and l is large [40].

The multiplication of polynomials using the Karatsuba algorithm has been implemented in C++ for polynomials with 32 bits long coefficients and in modulo q where $q = 1073479681$. We have considered the same q through out the entire implementation and for all different methods. The size of the polynomials are varied from 2 to 2^{16}. The multiplication code is then verified for associative and commutative properties as follows.

$$A \times B = B \times A$$
$$(A + B) \times C = A \times C + B \times C$$

For smaller values of n, its functionality is checked manually by comparing the output product with the precomputed expected results. We have also made a comparison of the results obtained by implementation of all different methods for the same pair of polynomials to verify for functional correctness. The algorithm for multiplication using the Karatsuba algorithm is given in Algorithm 2. The modular reduction in the ring quotiented by $x^n + 1$ is also considered in the algorithm. The reduction process will not be the same for any other cyclotomic polynomial.

Algorithm 2 Polynomial Multiplication using the Karatsuba algorithm (KA)

Input: A, B, n, q

procedure KA($\boldsymbol{A}$, $\boldsymbol{B}$)
  if $n = 1$ **then**
    $r \leftarrow A(0) \times B(0)$
  else
    for $j = 0$ to $(\frac{n}{2} - 1)$ **do**
      $A_0(i) \leftarrow A(i)$
      $A_1(i) \leftarrow A(i + \frac{n}{2})$
      $B_0(i) \leftarrow B(i)$
      $B_1(i) \leftarrow B(i + \frac{n}{2})$
    end for
    $\boldsymbol{k}_0 \leftarrow \mathrm{KA}(\boldsymbol{A}_0, \boldsymbol{B}_0)$
    $\boldsymbol{k}_{01} \leftarrow \mathrm{KA}\big((\boldsymbol{A}_0 + \boldsymbol{A}_1), (\boldsymbol{B}_0 + \boldsymbol{B}_1)\big)$
    $\boldsymbol{k}_2 \leftarrow \mathrm{KA}(\boldsymbol{A}_1, \boldsymbol{B}_1))$
    $\boldsymbol{k}_1 \leftarrow \boldsymbol{k}_{01} - \boldsymbol{k}_0 - \boldsymbol{k}_2$
    $\boldsymbol{r} \leftarrow \boldsymbol{k}_0 + \boldsymbol{k}_1 x^{n/2} + \boldsymbol{k}_2 x^n$
  end if
  for $i = 0$ to $(n - 2)$ **do**
    $r'_i \leftarrow (r_i - r_{n+i})$
  end for
  $r'_{n-1} \leftarrow r_{n-1}$
  return $\boldsymbol{r}'$

3.1.3 Multiplication using Toeplitz Matrix Vector Product

The product of two polynomials of size n in ring $Z/f(x)$ where $f \in Z$ can be computed as Toeplitz matrix-vector product. It has a subquadratic space complexity [3]. $f(x)$ is an irreducible polynomial of size n over Z. Let A and B be two polynomials with n 32-bit coefficients in ring $Z/(x^n+1)$. The product of A and B in the given ring can be computed by multiplying the polynomials and then reducing the product in modulo x^n+1. TMVP is a different approach where we modify one of the multiplicants to form a $n \times n$ Toeplitz, T matrix such the product of T and the other multiplicant gives product in the desired ring.

As mentioned for the schoolbook method, the product $C = A \times B \mod (x^n+1)$ can be written as $A \times \sum_{i=0}^{n-1} b_i x^i \mod (x^n+1)$. We can also express C as $\sum_{i=0}^{n-1} A^{(i)} b_i$ where $A^{(i)} = (x^i \times A) \mod (x^n+1)$ [1] [24]. This multiplication can also be represented in matrix-vector product format. The matrix product can be represented as follows

$$\boldsymbol{C} = \boldsymbol{A} \times \begin{pmatrix} \boldsymbol{B}^{(0)} \\ \boldsymbol{B}^{(1)} \\ . \\ . \\ . \\ \boldsymbol{B}^{(n-1)} \end{pmatrix}$$

where $\boldsymbol{B}^{(i)}$ is the matrix representation of $(x^i \times B) \mod (x^n+1)$. In this case the $n \times n$ matrix of $\boldsymbol{B}$ in $\mathrm{mod}(x^n+1)$ is a Toeplitz matrix and does not need further modifications. Let this $n \times n$ matrix be $\boldsymbol{M}_B$. Then

$$\boldsymbol{M}_B = \begin{pmatrix} b_0 & b_1 & b_2 & \dots & b_{n-1} \\ -b_{n-1} & b_0 & b_1 & \dots & b_{n-2} \\ -b_{n-2} & -b_{n-1} & b_0 & \dots & b_{n-3} \\ . & & & & . \\ . & & & & . \\ -b_1 & -b_2 & -b_3 & \dots & b_0 \end{pmatrix}$$

Since, n is a power of 2 the multiplication of vector $\boldsymbol{A}$ to matrix $\boldsymbol{M}_B$ is carried out by 2-way split Toeplitz matrix vector product as discussed in the previous chapter. The example given for multiplication using the karatsuba algorithm is repeated using the TMVP method. Given, $A = 5 + 10x + 9x^2 + 4x^3$ and $B = 10 + 8x + 3x^2 + 9x^3$. The product $C = A \times B$ The matrix representation of the coefficients of the polynomials is as follows.

$$\boldsymbol{A} = \begin{pmatrix} 5 & 10 & 9 & 4 \end{pmatrix} \text{ and } \boldsymbol{B} = \begin{pmatrix} 10 & 8 & 3 & 9 \\ -9 & 10 & 8 & 3 \\ -3 & -9 & 10 & 8 \\ -8 & -3 & -9 & 10 \end{pmatrix}$$

The matrices are broken down into half sized matrices.

$$\boldsymbol{A}_0 = \begin{pmatrix} 5 & 10 \end{pmatrix} \text{ and } \boldsymbol{A}_1 = \begin{pmatrix} 9 & 4 \end{pmatrix}$$

$$\boldsymbol{B}_0 = \begin{pmatrix} 10 & 8 \\ -9 & 10 \end{pmatrix}, \boldsymbol{B}_1 = \begin{pmatrix} 3 & 9 \\ 8 & 3 \end{pmatrix} \text{ and } \boldsymbol{B}_2 = \begin{pmatrix} -3 & -9 \\ -8 & -3 \end{pmatrix}$$

$\boldsymbol{k}_0 = \boldsymbol{A}_0 \times (\boldsymbol{B}_1 - \boldsymbol{B}_0)$, $\boldsymbol{k}_1 = \boldsymbol{A}_1 \times (\boldsymbol{B}_2 - \boldsymbol{B}_0)$ and $\boldsymbol{k}_2 = (\boldsymbol{A}_0 + \boldsymbol{A}_1) \times \boldsymbol{B}_0$

$\boldsymbol{r}_0 = \boldsymbol{k}_2 + \boldsymbol{k}_1$ and $\boldsymbol{r}_1 = \boldsymbol{k}_2 + \boldsymbol{k}_0$

$\boldsymbol{C} = \begin{pmatrix} \boldsymbol{r}_0 & \boldsymbol{r}_1 \end{pmatrix}$

$\boldsymbol{k}_0 = \begin{pmatrix} 5 & 10 \end{pmatrix} \times \begin{pmatrix} -7 & 1 \\ 17 & -7 \end{pmatrix}$, $\boldsymbol{k}_1 = \begin{pmatrix} 9 & 4 \end{pmatrix} \times \begin{pmatrix} -13 & -17 \\ 1 & -13 \end{pmatrix}$ and $\boldsymbol{k}_2 = \begin{pmatrix} 14 & 14 \end{pmatrix} \times \begin{pmatrix} 10 & 8 \\ -9 & 10 \end{pmatrix}$

The process is repeated recursively to evaluate $\boldsymbol{k}_0$, $\boldsymbol{k}_1$ and $\boldsymbol{k}_2$.

$$\boldsymbol{k}'_0 = 40,\ \boldsymbol{k}'_1 = 240 \text{ and } \mathbf{k}'_2 = -105$$
$$\boldsymbol{r}'_0 = 135 \text{ and } \mathbf{r}'_1 = -65$$

Therefore,

$$\boldsymbol{k}_0 = \begin{pmatrix} 135 & -65 \end{pmatrix},\ \boldsymbol{k}_1 = \begin{pmatrix} -113 & -205 \end{pmatrix} \text{ and } \boldsymbol{k}_2 = \begin{pmatrix} 14 & 252 \end{pmatrix}$$
$$\boldsymbol{r}_0 = \begin{pmatrix} -99 & 47 \end{pmatrix} \text{ and } \boldsymbol{r}_1 = \begin{pmatrix} 149 & 187 \end{pmatrix}$$
$$\boldsymbol{C} = \begin{pmatrix} -99 & 47 & 149 & 187 \end{pmatrix}$$

The implementation is verified by comparing results of $\boldsymbol{A} \times \boldsymbol{B}$ with $\boldsymbol{B} \times \boldsymbol{A}$ and $(\boldsymbol{A}+\boldsymbol{B})\boldsymbol{C}$ with $(\boldsymbol{A} \times \boldsymbol{C} + \boldsymbol{B} \times \boldsymbol{C})$ similar to the Karatsuba application. Manual comparison with precomputed results were also performed similar to the one done for the former method.

Algorithm 3 provides a simple presentation of the Toeplitz matrix vector product method for polynomial multiplication that is explained here. In this thesis, this method is implemented both in software and hardware .

As mentioned before, multiplication using the recursive TMVP method has a sub-quadratic complexity. Since the implementation is in ring $Z/(x^n + 1)$ with $n = 2^h$ we have used two-way split TMVP. The number of 32-bit multiplications $M_n = 3$ and the number

Algorithm 3 Multiplication of polynomials using TMVP method in $Z_p/(x^n+1)$

Input: A, B, n, q

procedure TMVP(A, B)
    if $n = 1$ **then**
        $\boldsymbol{Z} \leftarrow \boldsymbol{A} \times \boldsymbol{B}$
    else
        for $i = 0$ to $(\frac{n}{2} - 1)$ **do**
            $A_0(i) \leftarrow A(i)$
            $A_1(i) \leftarrow A(i + \frac{n}{2})$
            $B_0(i) \leftarrow B(i)$
            $B_1(i) \leftarrow B(i + \frac{n}{2})$
            $B_2(i) \leftarrow -B(\frac{n}{2} - i)$
        end for
        for $i = 0$ to $(\frac{n}{2} - 2)$ **do**
            $B_0(i + \frac{n}{2}) \leftarrow -B(n - i)$
            $B_1(i + \frac{n}{2}) \leftarrow B(\frac{n}{2} - 1 - i)$
            $B_2(i + \frac{n}{2}) \leftarrow -B(\frac{n}{2} + 1 - i)$
        end for
        $\boldsymbol{k}_0 \leftarrow \text{TMVP}\big(\boldsymbol{A}_0, (\boldsymbol{B}_1 - \boldsymbol{B}_0)\big)$
        $\boldsymbol{k}_1 \leftarrow \text{TMVP}\big(\boldsymbol{A}_1, (\boldsymbol{B}_2 - \boldsymbol{B}_1)\big)$
        $\boldsymbol{k}_2 \leftarrow \text{TMVP}\big((\boldsymbol{A}_0 + \boldsymbol{A}_1), \boldsymbol{B}_0\big)$
        $\boldsymbol{r}_0 \leftarrow \boldsymbol{k}_2 + \boldsymbol{k}_1$
        $\boldsymbol{r}_1 \leftarrow \boldsymbol{k}_2 + \boldsymbol{k}_0$
        $\boldsymbol{Z} \leftarrow \begin{pmatrix} \boldsymbol{r}_0 & \boldsymbol{r}_1 \end{pmatrix}$
    return $\boldsymbol{Z}$

of 32-bit additions $A_n = 5$ when $n = 2$. For an arbitrary $n = 2^h$, $M_n = 3M_{\frac{n}{2}} = n^{\log_2 3}$ and $A_n = 3A_{\frac{n}{2}} + 3n - 1 = 5.5n^{\log_2 3} - 6n + 0.5$ [13].

3.1.4 NTT-based Polynomial Multiplication and Algorithms

Different methods of multiplication of polynomial multiplications in ring $Z/(x^n + 1)$ are discussed in this section of the thesis. Number Theoretic Transform is a very efficient method for such implementation. NTT has a computational complexity of $O(n \log n)$. So, it is expected to be faster than the quadratic schoolbook method and other methods with subquadratic complexity for the multiplication of higher degree polynomials.

The modulus q is a prime number and in this case we have considered it to be an element of the RNS base. ω is the nth primitive root and is precomputed. Other related variables are also evaluated before starting the computation. The general algorithm for overview of polynomial multiplication using Number Theoretic Transform is given in Algorithm 4. We are considering polynomials of size n where n is only 2^h. The algorithm computes the negative wrapped convolution of the polynomials so it is not required to double the length of the polynomials. The resultant polynomial evaluated by negative wrapped convolution NTT is already reduced in the ring qoutiented by $x^n + 1$.

For a small toy example for multiplication of two polynomials using Number Theoretic Transform let the polynomials be $A = 5 + 10x$ and $B = 6 + 8x$. They can be represented in matrix form as

$$\boldsymbol{A} = \begin{pmatrix} 5 & 10 \end{pmatrix} \text{ and } \boldsymbol{B} = \begin{pmatrix} 6 & 8 \end{pmatrix}$$

Each of the matices are of length $n = 2$. We pad $\boldsymbol{A}$ and $\boldsymbol{B}$ with two zeroes. The new matrices are

$$\boldsymbol{A'} = \begin{pmatrix} 5 & 10 & 0 & 0 \end{pmatrix} \text{ and } \boldsymbol{B'} = \begin{pmatrix} 6 & 8 & 0 & 0 \end{pmatrix}$$

When we multiply using NTT we perform pointwise multiplication, so without the padding we will not get the third element of the product. For the new matrices $n = 4$. The minimum working modulus is 11, since all the inputs are less than 11. N be a prime number such that, $N = kn + 1$ and $N \geq 11$. Selecting $k = 3$ gives $N = 13$ which satisfies all the conditions. Generator for Z_{13}, $g = 6$ since $g^f \not\equiv 1 \bmod 13$ where f is any factor of 12. Therefore, $\omega = g^k = 6^3 \equiv 8 \bmod 13$. ω is the primitive 4th root of unity. The square matrix which multiplied to a given matrix of length 4 is

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

Now, the values required to do the inverse NTT are ω^{-1} and n^{-1}. When $\omega = 8$ and $n = 4$, $\omega^{-1} = 5 \bmod 13$ and $n^{-1} = 10 \bmod 13$. The matrix for $INV\text{-}NTT$ is

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 5 & 12 & 8 \\ 1 & 12 & 1 & 12 \\ 1 & 8 & 12 & 5 \end{pmatrix}$$

Evaluating $NTT(\boldsymbol{A'})$, $NTT(\boldsymbol{B'})$ and $\boldsymbol{C'} = NTT(\boldsymbol{A'}) \circ NTT(\boldsymbol{B'})$:

$$NTT(\boldsymbol{A}') = \begin{pmatrix}5 & 10 & 0 & 0\end{pmatrix} \begin{pmatrix}1 & 1 & 1 & 1 \\ 1 & 8 & 12 & 5 \\ 1 & 12 & 1 & 12 \\ 1 & 5 & 12 & 8\end{pmatrix} \text{ and } NTT(\boldsymbol{B}') = \begin{pmatrix}6 & 8 & 0 & 0\end{pmatrix} \begin{pmatrix}1 & 1 & 1 & 1 \\ 1 & 8 & 12 & 5 \\ 1 & 12 & 1 & 12 \\ 1 & 5 & 12 & 8\end{pmatrix}$$

$$\boldsymbol{C}' = NTT(\boldsymbol{A}') \circ NTT(\boldsymbol{B}') = \begin{pmatrix}2 & 7 & 8 & 3\end{pmatrix} \circ \begin{pmatrix}1 & 5 & 11 & 7\end{pmatrix} = \begin{pmatrix}2 & 9 & 10 & 8\end{pmatrix}$$

Therefore,

$$\boldsymbol{C} = INV\text{-}NTT\big(NTT(\boldsymbol{A}') \circ NTT(\boldsymbol{B}')\big)$$

$$= (10) \begin{pmatrix}2 & 9 & 10 & 8\end{pmatrix} \begin{pmatrix}1 & 1 & 1 & 1 \\ 1 & 5 & 12 & 8 \\ 1 & 12 & 1 & 12 \\ 1 & 8 & 12 & 5\end{pmatrix}$$

$$= \begin{pmatrix}30 & 100 & 80 & 0\end{pmatrix}$$

Hence, $c = (5 + 10x)(6 + 8x) = 30 + 100x + 80x^2$

Algorithms 4, 5 and 6 summarize the procedure for multiplying polynomials using NTT. $NTT^{\,n}_{w}(\hat{a})$ is the number theoretic transform of negative wrapped a i.e., $\hat{a}$ which is a recursive process and explained in Algorithm 5.

Algorithm 4 Polynomial multiplication using NTT

Input: $\mathbf{a}$, $\mathbf{b}$, ω, ω^{-1}, n, n^{-1}, q

```
Precompute: ω^i, ω^{-i}, φ^i, φ^{-i} for i ∈ (0, n − 1)
for i = 0 to (n − 1) do
    á_i ← a_i φ^i mod p
    b́_i ← b_i φ^i mod p
end for
Â ← NTT_w^n(â)
B̂ ← NTT_w^n(b̂)
for i = 0 to (n − 1) do
    Ĉ ← Â B̂ mod p
end for
ĉ ← InvNTT_w^n(Ĉ)
for i = 0 to (n − 1) do
    ć_i ← ĉ_i φ^{-i} mod p
end for
return c
```

Algorithm 5 Number Theoretic Transform(NTT)

Input: **a**, n, $index = 1$

procedure $NTT(\boldsymbol{a}, n, index)$
    for $n > 1$ **do**
        $n_2 = n/2$
        for $i = 0$ to $n_2 - 1$ **do**
            $a_0(i) \leftarrow a(2i)$
            $a_1(i) \leftarrow a(2i + 1)$
        end for
        $NTT(\boldsymbol{a}_0, n_2, 2 \times index)$
        $NTT(\boldsymbol{a}_1, n_2, 2 \times index)$
        for $i = 0$ to $n_2 - 1$ **do**
            Butterfly$\big(a(i), a(i + n_2), a_0(i), a_1(i), i \times index\big)$
        end for
    end for
    return $\boldsymbol{a}$

Algorithm 6 Butterfly

Input: $a(i), a(i + n_2), a_0(i), a_1(i), S = i \times index$

$T \leftarrow \big(S \times a(i + n_2)\big) \bmod q$
$a_1(i) \leftarrow a(2i + 1)$
$a(i) \leftarrow \big(a_0(i) + T\big) \bmod q$
$a(i_{n2}) \leftarrow \big(a_0(i) - T\big) \bmod q$ =0

3.1.5 Hybrid Design with Karatsuba and Toeplitz

Apart from the typical *Karatsuba* and TMVP methods where the recursion is repreated until the polynomials are down to the size of one element, we have implemented different versions of *Hybrid* designs where the recursive method is applied until n is down a certain value m, for example $4, 8$ or 16. We call it the *break-point* m where the recursion ends and all polynomial products are evaluated using general schoolbook method. The designs are functionally correct. Time taken to evaluate the polynomial multiplication is recorded for polynomials of size $n = 2^h$ for $2 \leq n \leq 16384$. For each value of n, 1000 different input sets are passed through the system and average time is recorded. This hybrid method is implemented to make imrpovements to the performance of the recursive methods.

Hybrid Karatsuba

For Karatsuba, the modular reduction is done only in the final stage after expansion and evaluation of the product. Wherever we decide to stop the recursion and compute the polynomial product of polynomials of size 2^h, we have to perform three mainstream multiplication of polynomials and obtain the polynomial products of size $2 \times 2^h - 1$. Then we combine the three polynomial and recursively keep expanding as previously in the divide and conquer method. The depth of the recursive method is reduced by $h - 1$ since the schoolbook method is more efficient for polynomials with lower sizes.

Since, the multiplication is in ring $\mathrm{Z}/(x^n + 1)$, the final product of size $2 \times 2^h - 1$ is then reduced by simply subtracting the $(2^h + i)$th coefficient from ith coefficient for $0 \leq i \leq n - 2$.

Hybrid TMVP

In case of Toeplitz matrix vector product, we are multiplying a vector of length n to an $n \times n$ matrix. The product is already reduced in the ring $\mathrm{Z}/(x^n + 1)$ and no further adjustments are required. The multiplication recursively keeps splitting into three multiplications of polynomials of half the size until the size of the vector and the Toeplitz matrix goes down to a single value. We modify this method so that instead of executing the recursion all the day to the point where the matrices are of size= 1, the recursion stops at a point where the size of the matrices is greater than one and all the required multiplications are done using the schoolbook method. For example, for any value of n we implement *Hybrid*-TMVP method with *break-point* $m = 16$. When the size of the polynomial $n = 16$, we switch to schoolbook method and perform multiplication of the polynomials by schoolbook method.

We are implementing the *Hybrid*-TMVP with different break-points and improvising further modifications to the design to improve performance. All the timing data from software implementation are tabulated and further decriptions are provided in Chapter 4.

3.2 Polynomial Multiplication in Ring $Z_p(x)/\Phi_k(x)$

In the previous chapter, it is mentioned that Toeplitz matrix approach for multiplication of polynomials of size n can be used so that we can multiply polynomials of size $n \neq 2^h$. Considering ring $Z/(x^n + 1)$, the difference in size between two consecutive polynomials is huge for higher values of n. For example, the size of polynomial jumps from 2^{10} to 2^{11} or 2^{11} to 2^{12}. We explore rings quotiented by other cyclotimic polynomials to allow multiplication of polynomials of different sizes within the different limits set by powers of two.

$\Phi_k(Class)$	h	i	j	$1024 \leq n \leq 2048$
I	10	–	–	1024
IV	6	2	–	1152
VI	1	1	2	1200
V	6	–	1	1280
IV	3	4	–	1296
VI	2	2	1	1440
II	–	6	–	1458
V	5	–	2	1600
IV	5	3	–	1728
VI	4	1	1	1920
IV	2	5	–	1944
V	2	–	3	2000
I	11	–	–	2048

Table 3.1: Range of sizes of polynomial within the range $1024 \leq n \leq 2048$ with various cyclotomic polynomials

Table 3.1 gives a detailed range of possible sizes that can be considered using various cyclotomic polynomials, Φ_k. In the table, h, i and j represents the powers of 2, 3 and 5 respectively and Class I to VI represents the six n-th cyclotomic polynomials mentioned in Chapter 2.

3.2.1 Multiplication of Polynomials in Ring $Z/(x^{2.3^i} + x^{3i} + 1)$

One of the reasons for using the TMVP method is for the multiplication of polynomials of sizes other than powers of two by considering other cyclotomic polynomials as mentioned in the previous section. This gives us a wide range of options in terms of the sizes of polynomial. For example, if n is a power of 2 the size of the polynomials jumps from 1024 to 2048. We can use the cyclotomic trinomial $x^{2.3^i} + x^{3i} + 1$ which would make multiplication of polynomials of size 1458 possible in the ring $Z/(x^{2.3^i} + x^{3i} + 1)$ where $i = 6$.

The multiplication of polynomials A and B of size $n = 2.3^i$ modulo $x^{2.3^i} + x^{3i} + 1$ is

$$A \times B \mod (x^{2.3^i} + x^{3i} + 1) = A \times (\sum_{i=0}^{2.3^i} b_i x^i) \mod (x^{2.3^i} + x^{3i} + 1) = \sum_{i=0}^{2.3^i} A^{(i)} \times b_i$$

Similar to the two-way split TMVP method we represent the product in matrix-vector product form $\boldsymbol{C} = \boldsymbol{M_A} \times \boldsymbol{B}$ where $\boldsymbol{M_A}$ is an $n \times n$ matrix representation of $[A^{(0)} A^{(1)} ... A^{(n-1)}]$ and $A^{(i)} = (x^i \times A) \mod (x^{2.3^i} + x^{3i} + 1)$ [1] [24]. Vectors $\boldsymbol{A} = \begin{pmatrix} a_0 & a_1 & a_2 & \dots & a_{n-1} \end{pmatrix}$ and $\boldsymbol{B} = \begin{pmatrix} b_0 & b_1 & b_2 & \dots b_{n-1} \end{pmatrix}$ represent polynomials A and B respectively. Let $\boldsymbol{A}_T^{(i)}$ be the transpose of the vector representation of $A^{(i)}$. Then

$$\boldsymbol{M_A} = \begin{pmatrix} \boldsymbol{A}_T^{(0)} & \boldsymbol{A}_T^{(1)} & \boldsymbol{A}_T^{(2)} & \dots & \dots & \boldsymbol{A}_T^{(n-1)} \end{pmatrix}$$

$$
\begin{pmatrix}
a_0 & -a_{2.3^i-1} & \dots & -a_{3^i+1} & -a_{3^i} & -(a_{3^i-1}-a_{2.3^i-1}) & \dots & -(a_1-a_{3^i+1}) \\
a_1 & a_0 & \dots & -a_{3^i+2} & -a_{3^i+1} & a_{3^i} & \dots & -(a_2-a_{3^i+2}) \\
. & & & & & & & . \\
. & & & & & & & . \\
. & & & & & & & . \\
a_{3^i-1} & a_{3^i-2} & \dots & a_0 & -a_{2.3^i-1} & -a_{2.3^i-2} & \dots & -a_{3^i} \\
a_{3^i} & a_{3^i-1}-a_{2.3^i-1} & \dots & a_1-a_{3^i+1} & a_0-a_{3^i} & -a_{3^i-1} & \dots & -a_1 \\
a_{3^i+1} & a_{3^i} & \dots & a_2-a_{3^i+2} & a_1-a_{3^i+1} & a_0-a_{3^i} & \dots & -a_2 \\
. & & & & & & & . \\
. & & & & & & & . \\
. & & & & & & & . \\
a_{2.3^i-1} & a_{2.3^i-2} & \dots & a_{3^i} & a_{3^i-1}-a_{2.3^i-1} & a_{3^i-2}-a_{2.3^i-2} & \dots & a_0-a_{3^i}
\end{pmatrix}
$$

Matrix $\boldsymbol{M_A}$ is a $2.3^i \times 2.3^i$ matrix. A toeplitz matrix $\boldsymbol{T_A}$ can be formed from $\boldsymbol{M_A}$ by shifting the last 3^i rows to the top and the rest of the 3^i to the bottom [1]. The matrix product $\boldsymbol{D} = \boldsymbol{T_A} \times \boldsymbol{B}$ is then computed using the TMVP algorithm. Since n is a multiple of 2, one iteration of 2-way split TMVP algorithm is carried out. This results in 3 half sized matrix multiplications. We can derive the product $\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B} \mod (x^{2.3^i} + x^{3i} + 1)$ by switching the top 3^i rows of $\boldsymbol{D}$ with the bottom 3^i rows.

$$
\boldsymbol{D} = \begin{pmatrix} \boldsymbol{T_0} & \boldsymbol{T_1} \\ \boldsymbol{T_2} & -\boldsymbol{T_0} \end{pmatrix} \begin{pmatrix} \boldsymbol{B}_0 \\ \boldsymbol{B}_1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{P}_0 + \boldsymbol{P}_2 \\ \boldsymbol{P}_1 - \boldsymbol{P}_2 \end{pmatrix}
$$

where $\boldsymbol{T_0}$, $\boldsymbol{T_1}$ and $\boldsymbol{T_2}$ are $3^i \times 3^i$ Toeplitz matrices.

$$\boldsymbol{T_0} = \begin{pmatrix} a_{3^i} & a_{3^i-1} - a_{2.3^i-1} & \dots & a_1 - a_{3^i+1} \\ a_{3^i+1} & a_{3^i} & \dots & a_2 - a_{3^i+2} \\ . & & & . \\ . & & & . \\ . & & & . \\ a_{2.3^i-1} & a_{2.3^i-2} & \dots & a_{3^i} \end{pmatrix}, \boldsymbol{T_2} = \begin{pmatrix} a_0 & -a_{2.3^i-1} & \dots & -a_{3^i+1} \\ a_1 & a_0 & \dots & -a_{3^i+2} \\ . & & & . \\ . & & & . \\ . & & & . \\ a_{3^i-1} & a_{3^i-2} & \dots & a_0 \end{pmatrix} \text{ and}$$

$$\boldsymbol{T_1} = \boldsymbol{T_2} - \boldsymbol{T_0}$$

$$\boldsymbol{P}_0 = (\boldsymbol{T_0} + \boldsymbol{T_1})\boldsymbol{B_1},\ \boldsymbol{P}_1 = (\boldsymbol{T_2} - \boldsymbol{T_0})\boldsymbol{B_0} \text{ and } \boldsymbol{P}_2 = \boldsymbol{T_0}(\boldsymbol{B_0} - \boldsymbol{B_1})$$

Hence,

$$\boldsymbol{C} = \begin{pmatrix} \boldsymbol{P}_1 - \boldsymbol{P}_2 \\ \boldsymbol{P}_0 + \boldsymbol{P}_2 \end{pmatrix}$$

$\boldsymbol{P}_0$, $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ are computed by three-way split TMVP method since each has $3^i \times 3^i$ Toeplitz matrix as a multiplier. Here, the number of multiplications $M_{2.3^i} = 3M_{3^i}$ and the number of additions $A_{2.3^i} = 5A_{3^i}$.

3.2.2 Multiplication of Polynomials in Ring $Z_p/(x^{2.2^h.3^i} - x^{2^h.3^i} + 1)$

Another trinomial that can be considered for the multiplication of two polynomials is $x^{2.2^h.3^i} - x^{2^h.3^i} + 1$. The Toeplitz matrix can be generated in a similar manner as the previous trinomial and then we perform a single iteration of two-way split. The multiplication is then carried out with h iterations of two-way split TMVP followed by i iterations of three-way split TMVP multiplication method. The subquadratic complexity of polynomial multiplication in the ring $Z_p/(x^{2.2^h.3^i} - x^{2^h.3^i} + 1)$ using TMVP is

$$M_{2.2^h.3^i} = 3M_{2^h.3^i} = 3 \times (2^h)^{\log_2 3} \times M_{3^i} = 3 \times (2^h)^{\log_2 3} \times (3^i)^{\log_3 6}$$

3.2.3 Multiplication using Three-Way Split TMVP

The two preceeding sections discuss computation of the product of polynomials of size $n = 2.3^i$ or $2.2^h.3^i$ using TMVP method. For $n = 2.3^i$, the first step is a single level of two-way splitting method for Toeplitz matrix-vector product followed by i iterations of three-way splitting method. Similarly, when $n = 2.2^h.3^i$, we have a single iteration of two-way split TMVP followed by h iterations of two-way split TMVP and lastly i iterations of three-way split TMVP method. The three-way split TMVP method is briefly discussed below.

$$\boldsymbol{C} = \begin{pmatrix} \boldsymbol{T}_0 & \boldsymbol{T}_1 & \boldsymbol{T}_2 \\ \boldsymbol{T}_3 & \boldsymbol{T}_0 & \boldsymbol{T}_1 \\ \boldsymbol{T}_4 & \boldsymbol{T}_3 & \boldsymbol{T}_0 \end{pmatrix} \begin{pmatrix} \boldsymbol{B}_0 \\ \boldsymbol{B}_1 \\ \boldsymbol{B}_2 \end{pmatrix} = \begin{pmatrix} \boldsymbol{P}_2 + \boldsymbol{P}_3 + \boldsymbol{P}_4 \\ \boldsymbol{P}_1 - \boldsymbol{P}_4 + \boldsymbol{P}_5 \\ \boldsymbol{P}_0 - \boldsymbol{P}_3 - \boldsymbol{P}_5 \end{pmatrix}$$

where $\boldsymbol{T}_0$, $\boldsymbol{T}_1$, $\boldsymbol{T}_2$, $\boldsymbol{T}_3$ and $\boldsymbol{T}_4$ are $3^{i-1} \times 3^{i-1}$ Toeplitz matrices and $\boldsymbol{B}_0$, $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$ are $3^{i-1} \times 1$ matrices.

$$\boldsymbol{P}_0 = (\boldsymbol{T}_0 + \boldsymbol{T}_3 + \boldsymbol{T}_4)\boldsymbol{B}_0$$

$$\boldsymbol{P}_1 = (\boldsymbol{T}_0 + \boldsymbol{T}_1 + \boldsymbol{T}_3)\boldsymbol{B}_1$$

$$\boldsymbol{P}_2 = (\boldsymbol{T}_0 + \boldsymbol{T}_1 + \boldsymbol{T}_2)\boldsymbol{B}_2$$

$$\boldsymbol{P}_3 = \boldsymbol{T}_0(\boldsymbol{B}_0 - \boldsymbol{B}_2)$$

$$\boldsymbol{P}_4 = \boldsymbol{T}_1(\boldsymbol{B}_1 - \boldsymbol{B}_2)$$

$$\boldsymbol{P}_5 = \boldsymbol{T}_3(\boldsymbol{B}_0 - \boldsymbol{B}_1)$$

$M_{3^i} = 6M_{3^{i-1}}$ and $A_{3^i} = 8A_{3^{i-1}}$ The subquadratic complexity of polynomial multiplication by three-way split Toeplitz for TMVP method is such that the $M_n = n^{\log_3 6}$ and $A_n = 4.8n^{\log_3 6} - 5n + 0.2$ [1].

So, the complexity of multiplication of polynomials of size $n = 2.3^i$ in ring $Z/(x^{2.3^i} + x^{3^i} + 1)$ is $M_{2.3^i} = 3 \times 3^{i^{\log_3 6}}$ and $A_{2.3^i} = 5(4.8 \times 3^{i^{\log_3 6}} - 5 \times 3^i + 0.2)$. Referring to the example of polynomials with 1458 coefficients which is 2.3^i , we can perform polynomial multiplication in the ring $Z_p/(x^{2.3^i}x^{3^i} + 1)$ with $i = 6$. The subquadratic complexity of this implementation is less than multiplication of polynomials with 2048 coefficients in ring $Z/(x^n + 1)$. A wider range of polynomials in terms of size can be considered.

The number of multiplications required for the multiplication of polynomials of size $n = 2.2^h.3^i$ in ring $Z/(x^{2.2^h.3^i} - x^{2^h.3^i} + 1)$ is $M_{2.2^h.3^i} = 3 \times 2^{h^{\log_2 3}} \times 3^{i^{\log_3 6}}$.

3.3 Hardware Implementation

Polynomial multiplication using *Karatsuba* and TMVP methods are also coded in VHDL. The specifications are similar to that of the software implementation in terms of ring, moduli, size and degree. Since, recursive approach is not always advisable in terms of hardware, components are created for each n where $n = 2^h$. The top level creates three half sized modules. The half sized module is also designed to create three even smaller components that takes polynomials of size $n/4$. In this way many components are created and together they work exactly as it would in the recursive configuration.

3.3.1 Functional Simulation of TMVP and Karatsuba

The application of TMVP and Karatsuba in Hardware is done considering similar approaches. The explanation below is true in case of both the apporaches.

Muliplication of two polynomials of size n, where each coefficient is reduced in 30 bits long modulo q is implemented in VHDL. The entire implementation is performed combinationally. The design is compiled and simulated using ModelSim. Separate modules and test benches for multiplying polynomials of size varying from 2 to 2^{11} are created and verified for functional correctness via functional simulation. The results obtained from each method are compared to the precomputed product for the same set of polynomials.

If we recall the software version, same implementation was carried out recusively. In case of hardware, recursive application gets complicated and results in synthesis issues. In order to avoid such circumstances, separate components were defined depending on the size of the input polynomial and for each polynomial product, the design for multiplying halved sized polynomials are created for three instances. In a way this implementation is same as the software implementation only creating copies of hardware each time the product splits into three half-sized products.

3.3.2 Field Programmable Gate Array (FPGA)

Field Programmable Gate Array (FPGA) is an array of programmable and reconfigurable gates. Xilinx, Microsemi, Altera and some other tools are available for FPGA synthesis. For synthesizing our design for FPGA, Xilinx ISE Design Suite is used. The design platform VC707, which is supposedly a high-performance, high-speed design platform, is available for use with license. Since, the number of IOBs limit to 600, it was a challenge to deal with large coefficients. We can only afford to have limited number of 32 bits long coefficients as input or as output every clock cycle. So, we store the two input arrays of 32 bits long coefficients in registers over a number of clock cycles and once we have both the input arrays ready, we start the multliplication process by TMVP method.

We have programmed the polynomial multiplication by TMVP in VHDL using components operating in combinational circuits.

3.4 Summary

This chapter presents detailed description of multiplication methods that are implemented in the thesis. Algorithms for multiplication in the ring $Z/(x^n + 1)$ using NTT, $Karatsuba$, TMVP and Schoolbook method and their complexities are described. Small examples are given for the ease of understanding. A hybrid method is then introduced that performs polynomial multiplication using the *Karatsuba* algorithm and TMVP for large values of

n and then switches to schoolbook method when n reduces to a small number such as 4, 8, 16 or 32. Two trinomials are presented and their roles in expanding the range of the sizes of polynomials are discussed. Multiplication in $Z/(x^n + 1)$ has been implemented in hardware using the method and here we have briefly discussed simulation and synthesis in FPGA.

Chapter 4

Analysis of Implementation

The implementations are run on a laptop with Intel® Core™ i7-5600U CPU @ 2.60GHz, under Linux. The machine has two cores, each having two HW threads and 4MB cache size. Multiplication of polynomials in ring $Z_p/(x^n + 1)$is executed using different methods. A comparison is made amongst the performance of each of these methods for a wide range of polynomials. *Hybrid* designs, as mentioned in the previous chapter are implemented in an attempt to improve the performance of the recursive methods. Modifications are made to the hybrid designs to further boost the performance. Other rings are considered for multiplication of polynomials of size $n \neq 2^h$. Using TMVP, multiplication of polynomials of sizes $n = 2.3^i$ or $n = 2.2^h 3^i$ are implemented in software. All these methods are implemented in software using $C++$. The two-way split TMVP method is implemented in hardware to multiply polynomials of size $n = 2^h$. All results are presented in tabular form in this chapter.

4.1 Multiplication in Ring $Z_p/(x^n + 1)$

Multiplication of polynomials are implemented using the schoolbook method which is the most basic method of multiplication, the *Karatsuba* algorithm, TMVP multiplication method and NTT. Schoolbook method which has the highest asymptotic complexity is the slowest while dealing with large polynomials and NTT is the fastest offering the best computational complexity. The *Karatsuba* algorithm and TMVP multiplication method have similar subquadratic computational complexities. All these methods are implemented in software for the multiplication polynomials in the ring quotiented by $x^n + 1$ where n is a power of two. Values of n in the range 2^2 to 2^{16} are considered and corresponding data is tabulated in the first section of this chapter. Schoolbook method is the fastest when smaller polynomials are multiplied. *Hybrid* version of the TMVP algorithm is implemented to acheive better performance with the recursive methods.

4.1.1 Comparison in Software for Different Methods of Multiplication

The average time taken to multiply polynomials with n coefficients has been recorded by n varying from 2^2 to 2^{16}. Time taken for 1000 differents pairs of polynomials are recorded and averaged for each value of n. The procedure is carried out using schoolbook method, the *Karatsuba* algorithm , TMVP method and NTT. The results are shown in Table 4.1. It is also helpful to check the range for which the general multiplication method is most efficient. This helps us design the hybrid TMVP and Karatsuba models as efficiently as possible.

The table shows that the schoolbook method is faster than all other methods for polynomials of sizes 2^2 to 2^5. For $n = 2^5$, TMVP and Karasuba are slower than NTT which is the slowest method for smaller polynomials. Multiplication by NTT method has complexity of $O(n \log n)$ which is significantly better compared to the complexities of Schoolbook method and the recursive methods which are $O(n^2)$ and $O(n^{\log_2 3})$ respectively. For polynomials with sizes $n = 2^6$ and above, NTT is always the fastest. Implementations with schoolbook method is the slowest among all when $n \geq 2^8$.

Size, n	Schoolbook (ms)	Karatsuba (ms)	TMVP (ms)	NTT (ms)
2^2	**0.00044**	0.00064	0.00064	0.00072
2^3	**0.00076**	0.00172	0.00188	0.00153
2^4	**0.00446**	0.00519	0.00524	0.00783
2^5	**0.00995**	0.02950	0.01629	0.01387
2^6	0.03367	0.05649	0.05788	**0.01673**
2^7	0.13362	0.14624	0.14280	**0.03557**
2^8	0.50831	0.43696	0.43178	**0.07924**
2^9	2.06235	1.34559	1.29159	**0.18295**
2^{10}	8.05378	3.95002	3.92518	**0.37736**
2^{11}	32.74930	12.13550	11.89820	**0.75523**
2^{12}	131.18400	37.13610	36.27160	**1.61358**
2^{13}	516.74300	110.36300	107.53300	**3.41800**
2^{14}	2073.79000	324.10700	323.70400	**7.34510**
2^{15}	8476.68000	969.18000	948.98100	**15.56130**
2^{16}	33331.80000	2961.11000	2864.13000	**34.57590**

Table 4.1: Timing report from Software Implementation using different multiplication methods

4.1.2 Hybrid Implementation

Table 4.1 shows that the schoolbook method is the fastest method for multiplication of polynomials of size $n \leq 2^5$. We have modified the recursive method in TMVP so that after the polynomials are broken down into polynomials with very small value of n, the method of multiplication switches to schoolbook method. We call it a *Hybrid* implementation of the the TMVP method that improvises schoolbook method. Hybrid designs have been tested with *break-points* m (points where the recursion stops and switches to schoolbook method) at different values of n in order to find the fastest practical implementation in software. We implement the design by switching to schoolbook method at $n = 4, 8, 16$ and 32.

Table 4.2 displays the implementation time for *Hybrid* variation of TMVP with *break-points* m at different values of n. This comparison is mainly done to identify the *break-point* that give us the best increase in performance.

Size, n	$m = 2$ (ms)	$m = 4$ (ms)	$m = 8$ (ms)	$m = 16$ (ms)	$m = 32$ (ms)	$m = 64$ (ms)
2^5	0.013	0.007	**0.006**	0.009	0.010	0.012
2^6	0.0303	0.022	**0.019**	0.024	0.037	0.046
2^7	0.089	0.070	**0.067**	0.072	0.087	0.110
2^8	0.267	0.207	**0.181**	0.202	0.289	0.336
2^9	0.806	0.614	**0.551**	0.619	0.885	1.026
2^{10}	2.405	1.807	**1.714**	1.846	2.624	2.942
2^{11}	7.276	5.403	**5.010**	5.593	7.877	8.856
2^{12}	21.620	17.185	**15.131**	17.179	23.719	26.781
2^{13}	65.103	48.401	**45.3502**	51.655	72.207	78.921
2^{14}	202.640	148.316	**136.954**	157.289	191.705	237.287
2^{15}	628.418	470.201	**423.522**	464.763	578.910	716.137
2^{16}	1826.970	1386.160	**1221.150**	1375.650	1720.210	2141.92

Table 4.2: Timing report for *Hybrid*-TMVP implementation with different *break-points* m

The highlighted column of Table 4.2 represents the best result for our hybrid design. According to the data collected, this design gives the best improvement to our TMVP implementation when we switch to schoolbook method at $n = 8$. With this design, a performance better than NTT can be achieved for polynomials of size 2^5 and almost as good as NTT for $n = 2^6$.

Further Improvements

The *Hybrid* design shows a good improvement in the performance. Considering other factors, we attempt to increase the performance even more in software. The schoolbook method requires a large number of modular reduction operations. We try to reduce the number of *mod* operations and also perform *loop unroll* to speed up the process since we know the upper limit for m.

1. Decreasing the Number of Modular Reduction

All the products need to be reduced in modulo q. Methods like NTT requires n modular reduction since it needs only n multiplications and the recursive methods require $n^{\log_2 3}$ modular reduction. On the other hand, with schoolbook method we need n^2 multiplications and hence n^2 modular reductions. We modify our hybrid implementation to get better results. We maintain the same concept except we modify the code for schoolbook method to improve the performance by reducing the number of *mod* operations. Therefore, the number of *mod* operations are reduced from m^2 to $\frac{m^2}{16}$. The new design uses an array to store the 64-bit integer products until 16 such multiplications are done and then the stored 64-bit integers are added in five batches followed by a *mod* operation. This design works for ≥ 16. Similar design can be implemented for lower values of n as well. We are trying to increase the performance when the design switches to schoolbook method when $n \geq 16$.

Table 4.3 represents the timing data for the modified hybrid version of TMVP method with *break-points* at $n = 16$ and $n = 32$. We did not repeat it with lower *break-points* since our design is modified to reduce the number of modular reduction by working with 16 values at a time.

Size, n	$m = 16$ (ms)	$m = 32$ (ms)
2^5	**0.005**	0.009
2^6	**0.016**	0.035
2^7	**0.052**	0.062
2^8	**0.146**	0.182
2^9	**0.455**	0.543
2^{10}	**1.373**	1.661
2^{11}	**4.117**	5.165
2^{12}	**12.370**	15.411
2^{13}	**38.047**	45.631
2^{14}	**116.502**	136.601
2^{15}	**353.999**	412.746
2^{16}	**1015.930**	1243.570

Table 4.3: Timing report for comparing modified *Hybrid*-TMVP implementation with reduced number of *mod* at breakpoints $m = 16$ and $m = 32$

The initial hybrid design shows best result with break-point at $n = 8$ and that is why we have this modified design to see if the performance with break-points at $n = 16$ and $n = 32$ can be improved or not. This modified implementation performs faster than NTT till $n = 2^6$ which is an improvement from our first Hybrid design which was better than glsntt for $n \leq 2^5$.

2. Effect of Loop Unrolling

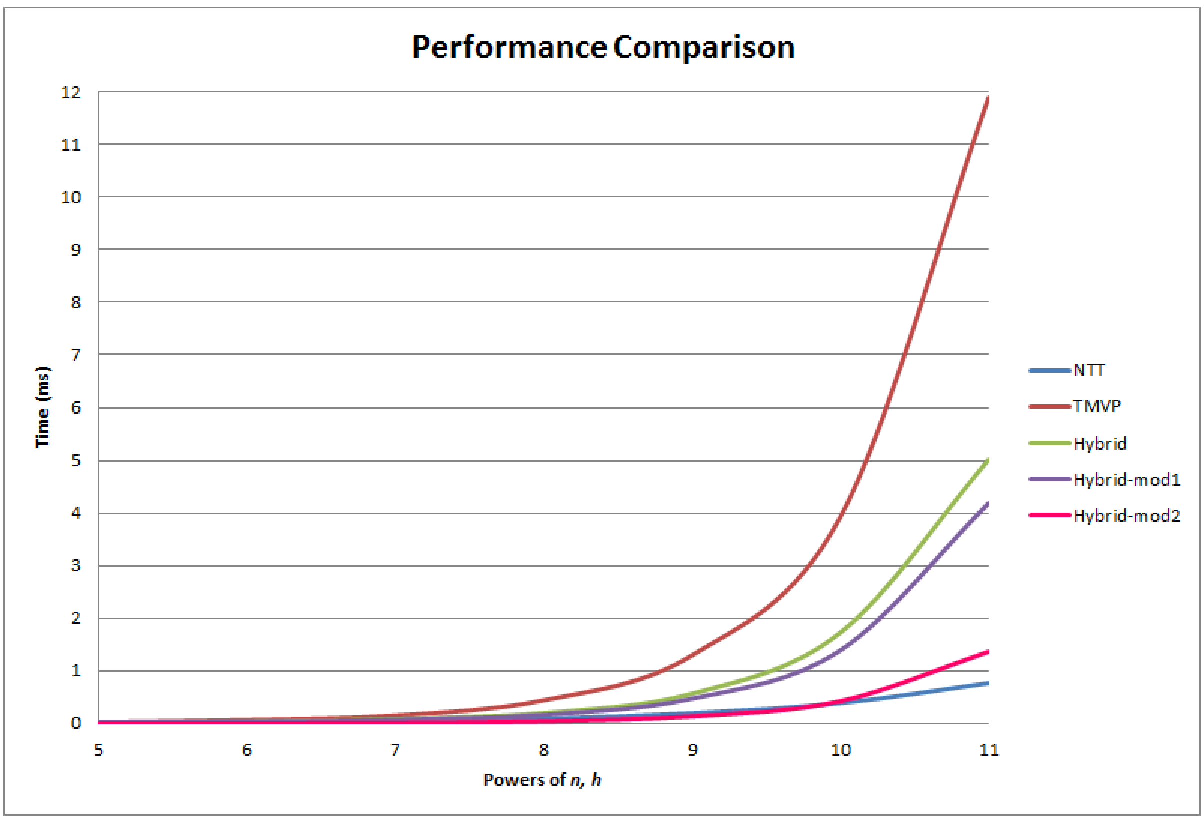
Another way of implementing the Hybrid design is by unrolling the *for-loops* in the school-book method and using the least number of operations. Consdering large value of n as the break-point is not ideal for this implementation as it needs all the required multiplications to be defined since the *loops* are *unrolled*. For larger polynomials, it becomes tricky with higher chances of inducing error to the code so we limit our break-point to a maximum of 32. Apart from unrolling the *for-loops*, here we have also tried to minimize the number of *mod* operations used. In *C++*, *mod* operation is basically equal to three basic operations. For this method we require only m *mod* operations for reducing the 64-bit integer product by modulo p. m is the point where the process switches to schoolbook method. The result with the modified version is given in Table 4.4.

Size, n	$m = 4$ (ms)	$m = 8$ (ms)	$m = 16$ (ms)	$m = 32$ (ms)
2^5	0.0037	0.0021	0.0015	**0.0013**
2^6	0.0117	0.0071	0.0063	**0.0046**
2^7	0.0405	0.0226	0.0187	**0.0138**
2^8	0.1101	0.0702	0.0484	**0.0456**
2^9	0.3351	0.2054	0.1525	**0.1386**
2^{10}	0.9855	0.6248	0.4680	**0.4230**
2^{11}	2.9945	1.9104	1.4389	**1.3469**
2^{12}	8.9957	5.8133	4.2516	**3.9322**
2^{13}	27.3733	17.7335	12.9279	**11.9301**
2^{14}	82.6420	53.4080	39.4232	**36.2145**
2^{15}	249.6320	158.8880	116.4950	**108.6110**
2^{16}	740.7690	471.9190	358.2720	**325.7600**

Table 4.4: Timing report for comparing modified *Hybrid*-TMVP implementation with *loop* unrolling

This unrolled *for-loops* version of schoolbook method gives a better performance with higher break-points. The elimination of *for-loops* and *mod* operations cause a great improvement in performance. With this design, the implementation is faster than NTT for polynomials of size $\leq 2^9$. The performance is very similar to NTT at $n = 2^{10}$ and above that NTT performs better than the hybrid. The plain TMVP method is as good as or even than better NTT for multiplication of polynomials of sizes $\leq 2^4$ whereas using the modified hybrid version of TMVP we can acheive an NTT-like performance for polynomials with sizes upto 2^{10}. We are comparing the new design's performance to that of NTT because is the fastest of all methods being investigated for multiplying larger polynomials as shown in Table 4.1.

The plot presents a comparison among the performances of TMVP, NTT, *Hybrid* design and the two variations of the *Hybrid* design.



For polynomials of size $n = 2^h$, The plot shows h along the x-axis and runtime in *ms* along the y-axis. The graph represents improvement in performance of TMVP when a *Hybrid* design is considered. The pink line representing performance *unrolled for-loop* version of hybrid intersects the line for NTT at $h \approx 10$.

4.2 Multiplication in Rings Quotiented by Cyclotomic Trinomials

NTT limits the size of the polynomials to only powers of two. Polynomial multiplication in rings quotiented by other cyclotomic polynomials allow a wider range of the size of the polynomials to choose from. The TMVP method with a subquadratic complexity is used in this thesis to implement of polynomials with sizes that are not powers of two. We are considering rings $Z_p/(x^{2.3^h} + x^{3^h} + 1)$ and $Z_p/(x^{2.2^i.3^h} - x^{2^i.3^h} + 1)$. The implementation is then modified to increase the performance by considering adequate *hybrid* method.

4.2.1 Multiplication of Polynomials in Ring $Z_p/(x^{2.3^h} + x^{3^h} + 1)$ using TMVP

An implementation of polynomial multiplication in the ring $Z_p/(x^{2.3^h} + x^{3^h} + 1)$ is done to multiply of two polynomials that are not size n =powers of two efficiently. Our goal is to make the implementation efficient in terms of performance. Using the algorithm mentioned in Chapter 3 we have developed a C++ code for multiplying two polynomials of size $n = 2.3^h$ using the TMVP method. We have considered a combined TMVP method where one iteration of two-way split TMVP is performed followed by recursive application of three-way split TMVP. We compare the results of pure two-way split TMVP with the result from this implementation to show how the runtime can be reduced by considering other rings when polynomials of size $n \leq 2^h$ are desired. This is an implementation that we can not consider using NTT. The range of sizes that can be implemented in the range $n = 2^2$ to $n = 2^{16}$ are $6, 18, 54, 162, 486, 1458, 4374, 13122$ and 39366. The complexity of this implementation is $O(n^{\log_3 6})$ which is very similar to the complexity of two-way split TMVP method. We implement a *Hybrid* design of the three-way spilt TMVP improvising schoolbook method to increase the performance.

Table 4.5 represents the data from the implementation using TMVP and *Hybrid* designs of the TMVP with different values n as break-point. The table only shows large values of n because we are not much concerned about smaller polynomials for cryptographic purposes.

Size, n	TMVP (ms)	Hybrid(9) (ms)	Hybrid(27) (ms)	Mod-Hybrid(9) (ms)	Mod-Hybrid(27) (ms)
162	0.218	0.095	0.118	0.033	**0.021**
486	1.308	0.561	0.711	0.192	**0.132**
1458	7.906	3.368	4.305	1.235	**0.844**
4374	47.844	20.583	25.965	7.389	**5.054**
13122	286.626	122.407	154.149	45.814	**31.429**
39366	1725.190	734.938	1039.061	286.549	**190.180**

Table 4.5: Timing report for multiplication in ring $Z_p/(x^{2.3^h} + x^{3^h} + 1)$ using three-way split TMVP and *Hybrid(m)* with different *break-points* m

4.2.2 Multiplication of Polynomials in Ring $Z_p/(x^{2.2^i.3^j} - x^{2^i.3^j} + 1)$ using TMVP

$x^{2.2^i.3^j} - x^{2^i.3^j} + 1$ is a trinomial and we are performing polynomial multiplication considering the ring $Z_p/(x^{2.2^i.3^j} - x^{2^i.3^j} + 1)$ with the combination of two-way split and three-way split

TMVP approach. The sizes of polynomials are not limited to powers of 2 but $2.2^i.3^j$. So we can choose size n from a much bigger range of values. There is a huge range of sizes $\leq 2^{16}$ that can be achieved so we implement and collect data for all possible polynomials only in the range 2^{10} to 2^{11} and compare the performance. We also implement the basic *Hybrid* design to improve the timing results. The results are given in Table 4.6. The break-point m of each of the hybrid design is mentioned in braces as Hybrid(m).

Size, n	TMVP (ms)	Hybrid(9) (ms)	Hybrid(27) (ms)	Mod-Hybrid(9) (ms)	Mod-Hybrid(27) (ms)
1152	4.526	2.010	2.009	0.735	**0.734**
1296	5.835	2.528	3.169	0.888	**0.661**
1536	7.137	3.654	3.658	2.444	**2.443**
1728	8.855	3.919	4.815	1.389	**1.023**
1944	11.778	5.045	6.330	1.718	**1.296**

Table 4.6: Timing report for multiplication in ring $Z_p/(x^{2.2^i.3^h} - x^{2^i.3^h} + 1)$ using a combination of two-way split and three-way split TMVP and Hybrid(m) where m is the *break-point*

The higher the break-point, the slower is the performance for the *Hybrid* desgin. We can modify our *Hybrid* design as mentioned in this chapter and improve the performance for higher break-points. We can see some inconsistency in the results from the modified *Hybrid* designs. For example, the multiplication of polynomials with $n = 1728$ is faster than it is with $n = 1536$. If we beak the values down, $1536 = 2.2^8.3$ whereas $1728 = 2.2^5.3^3$. As discussed before in this chapter, the modified *Hybrid* is expected to give the better performance than plain TMVP and we are considering modified *Hybrid* for powers of three. $n = 1728$ has lower powers of two and higher powers of three than $n = 1536$ and performance of *Hybrid* part of the design is more prominant incase of $n = 1728$.

4.3 Hardware Implementation of Two-Way Split TMVP

The Xilinx board used for the experiments has a limitation of 600 input and output ports. The inputs are passed sequentially in groups of four 32-bit coefficients of each of the two input arrays every cycle and stored in registers. When all the coefficients have been stored, the multiplication component takes inputs and evaluates the output which is then stored in registers and passed as output in groups of 4 coefficeints each cycle. Hence, depending on the size of the polynomials, the number of clock cycles varies. The result of FPGA synthesis in Xilinx is represented in Table 4.7. The clock period, the number of registers and LUTs required are tabulated for different sizes of polynomials.

Size, n	Slice Registers	LUTs	% used	Clock Period(ns)
2^1	391	1183	1	17.689
2^2	797	4194	1	22.818
2^3	994	12987	4	29.241
2^4	3563	43486	14	37.214
2^5	7443	139047	45	52.526

Table 4.7: Xilinx synthesis report on implementation of polynomial multiplication using TMVP

Table 4.7 presents the data of implementing all the blocks combinationally, that is for each of the three recursive half-sized multiplication operation, three modules are created and implemented in parallel. The entire multiplication operation occurs in a single clock cycle. However, the operation does not work for $n \geq 2^6$ with this approach. The design platform VC707 runs out of LUTs. The implementation for $n = 2^6$ requires 150% of LUTs available making synthesis infeasible.

For multiplications of polynomials with sizes $\geq 2^6$, we reuse smaller multiplication module sequentially in different clock cycles instead of creating three copies of the module. The sequential implementation does not increase the % of LUTs required too significantly but it does increase the number of registers required to store the intermediate data. Table 4.8 shows the synthesis report for $n = 2^6$ and $n = 2^7$. Each of the critical path is the multiplication of 32-bit coefficient polynomials with 32 coefficients.

Size, n	Slice Registers	% used	LUTs	% used	Clock cycles	Clock Period(ns)
2^6	19752	3	134879	44	3	54.258
2^7	45561	7	173194	57	9	58.063

Table 4.8: Xilinx synthesis report for polynomials of size 2^6 and 2^7 reusing the block for 2^5

The design can be modified by replacing the TMVP multiplication procedure for smaller values of n with schoolbook method as we did in software to speed up the multiplication in smaller modules. We have a limited number IOB ports limiting us to output only 4 coefficients per clock cycle. This increases the number of clock latencies.

4.4 Summary

Multiplication of polynomials of size $n = 2^h$ are implemented in software using schoolbook method, the *Karatsuba* algorithm, TMVP method and NTT in the ring $Z_p/(x^n + 1)$. One of the goals of the thesis is to make the multiplication using TMVP somewhat as efficient as it is with NTT for larger polynomial. Different *Hybrid* versions of TMVP method are implemented to improve the performance in software. Another reason to consider the TMVP method is to allow multiplication of polynomials with sizes other than powers of two. Here we have implemented multiplication using two-way split and three-way split TMVP methods in two different rings that allow a wider range of polynomials. Similar hybrid algorithms are considered for these multiplications to enhance the performance. Simple two-way split TMVP method for multiplication in the ring $Z_p/(x^n+1)$ is simulated and synthesized in hardware for sizes $n = 2$ to $n = 2^7$. All implementations are done in modulo p where p is a 30-bit integer.

Chapter 5

Concluding Remarks

5.1 Contribution Summary

In this thesis, different approaches have been analyzed for the multiplication of polynomials in *power-of-two* cyclotomics, i.e., cyclotomic $\Phi_k = x^n + 1$ where $n = 2^h$. Performance of NTT is significantly better than any of the other methods for very large values of n because of its quasi-linear complexity. Schoolbook method gives the best performance for $n \leq 2^5$ and becomes the slowest for larger polynomials as it has a complexity of $O(n^2)$. Performances of Karatsuba algorithm and TMVP with subquadratic complexities are better than schoolbook method for $n \geq 2^8$. Multiplication in ring $Z_p/(x^n + 1)$ using a $Hybrid$ version of TMVP is implemented and it shows performance almost as good as NTT for $n \leq 2^{10}$ in software. Using TMVP method, multiplication of polynomials in other cyclotomic rings $Z_p/\Phi_k(x)$ has been implemented successfully with a subquadratic complexity. Trinomials $\Phi_k(x) = x^{2.3^i} + x^{3^i} + 1$ and $\Phi_k(x) = x^{2.2^h.3^i} + x^{2^h.3^i} + 1$ have been considered. Hybrid designs of TMVP and also the modified versions of Hybrid with unrolled *for-loop* have been implemented which result in performaces twice as good. Performance of this implementation for $n \leq 2^9$ is comparable to the performance of NTT for $n \leq 2^9$ in $Z_p/(x^n + 1)$.

The hardware implementation of two-way split TMVP is executed combinationally and synthesized in FPGA for $n \leq 2^5$. For larger polynomials, implementation is synthesized reusing the block for $n = 2^5$ in sequence.

5.2 Future Work

All implementations are carried out considering one of the 291 moduli from the RNS utilized in NFLlib, which is an open source library for lattice-based cryptography. The polynomials can actually be represented as a k-tuple with the 30-bit residues in each of the moduli and processed independently. NFLlib is an open source library that utilizes optimized version of NTT for arithmetic operations in certain HE. An optimized and parallelized version of our *Hybrid*-TMVP method can be analyzed for its performance in NFLlib in place of the default NTT. The Hybrid design can be implemented in existing schemes that involves multiplication of polynomials of size less than 2^{10} and its performance can be compared with respect to the existing implementation.

All implementations are carried out only in software except the two-way split TMVP method for multiplication in ring quotiented in $x^n + 1$ which is synthesized in FPGA using Xilinx. However, all the hybrid implementations can be repeated in hardware and synthesized in ASIC and FPGA. Area and throughput optimized implementation in hardware can be aimed using the hybrid design. We can have a comparison in terms of space complexity among all the implementations in Hardware.

References

[1] Jithra Adikari, M Anwar Hasan, and Christophe Negre. Towards faster and greener cryptoprocessor for η pairing on supersingular elliptic curve over $\mathbb{F}_{2^{1223}}$. In *International Conference on Selected Areas in Cryptography*, pages 166–183. Springer, 2012.

[2] Carlos Aguilar-Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrede Lepoint. NFLlib: NTT-based fast lattice library. In *Cryptographers Track at the RSA Conference*, pages 341–356. Springer, 2016.

[3] Sedat Akleyleky and Ferruh Özbudak. Multiplication in a Galois ring. In *Signal Design and its Applications in Communications (IWSDA), 2015 Seventh International Workshop on*, pages 28–32. IEEE, 2015.

[4] Eric Bach and Jeffrey Shallit. Factoring with cyclotomic polynomials. *Mathematics of Computation*, 52(185):201–219, 1989.

[5] Jean-Claude Bajard, Julien Eynard, Anwar Hasan, and Vincent Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *Selected Areas in Cryptography-SAC*, 2016.

[6] Antoon Bosselaers, René Govaerts, and Joos Vandewalle. Comparison of three modular reduction functions. In *Annual International Cryptology Conference*, pages 175–186. Springer, 1993.

[7] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO*, volume 7417, pages 868–886. Springer, 2012.

[8] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Annual cryptology conference*, pages 505–524. Springer, 2011.

[9] Jorge Castiñeira Moreira and Patrick Guy Farrell. Appendix B: Galois Fields GF (q). *Essentials of Error-Control Coding*, pages 339–349.

[10] D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. C. Cheung, D. Pao, and I. Verbauwhede. High-Speed Polynomial Multiplication Architecture for Ring-LWE and SHE Cryptosystems. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(1):157–166, Jan 2015.

[11] William T Cochran, James W Cooley, David L Favin, Howard D Helms, Reginald A Kaenel, William W Lang, GC Maling, David E Nelson, Charles M Rader, and Peter D Welch. What is the fast fourier transform? *Proceedings of the IEEE*, 55(10):1664–1674, 1967.

[12] Keith Conrad. On the origin of representation theory. *ENSEIGNEMENT MATHEMATIQUE*, 44:361–392, 1998.

[13] Haining Fan and M Anwar Hasan. A new approach to subquadratic space complexity parallel multipliers for extended binary fields. *IEEE Transactions on Computers*, 56(2):224–233, 2007.

[14] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[15] Xianjin Fang and Longshu Li. On karatsuba multiplication algorithm. In *Data, Privacy, and E-Commerce, 2007. ISDPE 2007. The First International Symposium on*, pages 274–276. IEEE, 2007.

[16] Harvey L Garner. The residue number system. In *Papers presented at the the March 3-5, 1959, western joint computer conference*, pages 146–153. ACM, 1959.

[17] Craig Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[18] Andy Greenberg. Hacker lexicon: What is homomorphic encryption?, Jun 2017.

[19] M Anwar Hasan, Nicolas Meloni, Ashkan H Namin, and Christophe Negre. Block recombination approach for subquadratic space complexity binary field multiplication based on Toeplitz matrix-vector product. *IEEE Transactions on Computers*, 61(2):151–163, 2012.

[20] David Lee. Fast multiplication of a recursive block Toeplitz matrix by a vector and its application. *Journal of Complexity*, 2(4):295–305, 1986.

[21] Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In *International Conference on Cryptology and Network Security*, pages 124–139. Springer, 2016.

[22] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[23] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 35–54. Springer, 2013.

[24] Edoardo Mastrovito. VLSI architectures for computations in Galois fields. *PhD thesis, Dept. of Electrical Eng, Linkoping Univ.*, 1991.

[25] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[26] Vincent Migliore, Maria Méndez Real, Vianney Lapotre, Arnaud Tisserand, Caroline Fontaine, and Guy Gogniat. Fast polynomial arithmetic for somewhat homomorphic encryption operations in hardware with karatsuba algorithm. In *Field-Programmable Technology (FPT), 2016 International Conference on*, pages 209–212. IEEE, 2016.

[27] Ahmad Habibizad Navin, Asghar Shahrzad Khashandarag, Amin Rahimi Oskuei, and Mirkamal Mirnia. A novel approach cryptography by using residue number system. In *Computer Sciences and Convergence Information Technology (ICCIT), 2011 6th International Conference on*, pages 636–639. IEEE, 2011.

[28] Harald Niederreiter. A survey of some applications of finite fields. *Designs, Codes and Cryptography*, 78(1):129–139, 2016.

[29] Christof Paar. A new architecture for a parallel finite field multiplier with low complexity based on composite fields. *IEEE Transactions on Computers*, 45(7):856–861, 1996.

[30] Thomas Pöppelmann and Tim Güneysu. Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. *LatinCrypt*, 7533:139–158, 2012.

[31] Karl C Posch and Reinhard Posch. Modulo reduction in residue number systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(5):449–454, 1995.

[32] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[33] CP Rentería-Mejía and J Velasco-Medina. Hardware design of an NTT-based polynomial multiplier. In *Programmable Logic (SPL), 2014 IX Southern Conference on*, pages 1–5. IEEE, 2014.

[34] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[35] Sujoy Sinha Roy, Kimmo Järvinen, Frederik Vercauteren, Vassil Dimitrov, and Ingrid Verbauwhede. Modular hardware architecture for somewhat homomorphic function evaluation. Cryptology ePrint Archive, Report 2015/337, 2015. https://eprint.iacr.org/2015/337.

[36] Sujoy Sinha Roy, Frederik Vercauteren, Nele Mentens, Donald Donglong Chen, and Ingrid Verbauwhede. Compact ring-LWE cryptoprocessor. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 371–391. Springer, 2014.

[37] Assaid Othman Sharoun. Residue number system (RNS). *Poznan University of Technology Academic Journals. Electrical Engineering*, 2013.

[38] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. *Advances in Cryptology-ASIACRYPT 2010*, pages 377–394, 2010.

[39] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *Asiacrypt*, volume 5912, pages 617–635. Springer, 2009.

[40] André Weimerskirch and Christof Paar. Generalizations of the Karatsuba algorithm for efficient implementations (2006). *URL: http://eprint. iacr. org/2006/224. Citations in this document*, 2.

APPENDICES

Appendix A

A.1 List of all the moduli in the RNS base used in this thesis

The results presented in this thesis are based on modular reduction in 1073479681. While representing the enormous cryptographic data in RNS, it is reduced in each of the following moduli which are member of the RNS base under consideration. Polynomial multiplication and all other operations are performed on each of the 291 numbers that are the reduced representation of the data in RNS base.
Below is the list of all the 291 moduli of the RNS base B.

1073479681, 1072496641, 1071513601, 1070727169, 1069219841, 1068564481, 1068433409, 1068236801, 1065811969, 1065484289, 1064697857, 1063452673, 1063321601, 1063059457, 1062862849, 1062535169, 1062469633, 1061093377, 1060765697, 1060700161, 1060175873, 1058209793, 1056440321, 1056178177, 1055260673, 1054212097, 1054015489, 1053818881, 1052835841, 1052508161, 1051721729, 1049100289, 1048772609, 1048707073, 1048379393, 1045430273, 1043464193, 1042415617, 1041694721, 1040908289, 1040842753, 1040056321, 1038745601, 1038155777, 1037303809, 1036779521, 1034813441, 1033961473, 1032650753, 1032257537, 1032192001, 1031667713, 1030619137, 1029308417, 1028456449, 1026686977, 1026490369, 1026162689, 1025703937, 1023148033, 1022164993, 1021444097, 1021247489, 1020592129, 1019805697, 1019609089, 1019478017, 1018429441, 1018101761, 1017839617, 1016922113, 1016463361, 1015283713, 1014366209, 1012989953, 1012924417, 1012596737, 1012334593, 1012006913, 1011023873, 1010761729, 1010565121, 1009975297, 1008795649, 1008271361, 1007681537, 1006108673, 1005649921, 1005518849, 1005060097, 1004535809, 1004339201, 1002766337, 1002373121, 1000800257, 1000210433, 999948289, 999424001, 999161857, 998572033, 998244353, 997261313, 996278273, 995622913, 995033089, 994902017, 994705409, 994246657, 993918977, 993329153, 993263617, 992083969, 991887361, 991363073, 991297537, 990576641, 989986817, 989003777, 988938241, 988610561, 986382337, 985661441, 985464833, 985006081, 984481793, 983826433, 982450177, 982056961, 981270529, 980746241, 980156417, 979107841, 978780161, 977993729, 977534977, 976355329, 976224257, 975831041, 975634433, 975175681, 974979073, 974258177, 973406209, 972029953, 971898881, 971243521, 970129409, 969146369, 967507969, 967180289, 966197249, 964558849, 962854913, 962592769, 962396161, 961085441, 959119361, 958922753, 958136321, 957939713, 957677569, 957546497, 957349889, 956366849, 955383809, 954531841, 954335233, 952434689, 952238081, 952041473, 951582721, 950468609, 950403073, 950009857, 949682177, 949616641, 949420033, 948699137, 948633601, 947847169, 946339841, 944570369, 944177153, 943718401, 942800897, 941424641, 940572673, 939655169, 938475521, 936312833, 935329793, 935264257, 933888001, 933101569, 932970497, 932904961, 932577281, 930742273, 930021377, 929955841, 929366017, 927662081, 927072257, 926220289, 925892609, 924844033, 924712961, 924450817, 924254209, 922877953, 922550273, 921894913, 921501697, 920518657, 920322049, 919601153, 919339009, 918552577, 918224897, 917569537, 917176321, 916389889, 915996673, 915013633, 914685953, 914096129, 913899521, 913309697, 913244161, 912130049, 911081473, 910491649, 909770753, 909377537, 909180929, 908328961, 908197889, 908132353, 907804673, 907542529, 907411457, 907214849, 907018241, 906362881, 904265729, 903282689, 903086081, 902627329, 902430721, 900923393, 900857857, 900464641, 899678209, 898301953, 897712129, 897581057, 897318913, 896991233, 896729089, 896204801, 896008193, 894959617, 893255681, 893059073, 892403713, 891617281, 890437633, 889454593, 889323521, 889126913, 889061377, 888668161, 887488513, 885719041, 885522433, 883949569, 882245633, 881983489, 881590273, 880869377, 880803841, 880214017, 879230977, 876675073, 875298817, 874708993, 874315777, 873725953, 873332737, 873136129, 873005057

A.2 Modular Reduction using Barrett's Reduction Algorithm

Modular Reduction is generally a slow process as it depends on repetitive use of long divisions. Barrett's reduction algorithm limits the need for numerous long divisions and replaces divisions with shifts, multiplications and subtractions.

The algorithm explaining all the steps required for Barrett's reduction is given below.

Algorithm 7 Barrett's Reduction

Input: a, q

1: **Precompute:** k such that $2^k > q, u$ where $u = \left\lfloor \frac{4^k}{n} \right\rfloor$

2: $\acute{a} \leftarrow a - \left\lfloor \frac{au}{4^k} \right\rfloor q$

3: **if** $\acute{a} \geq q$ **then**

4: $\quad \acute{a} \leftarrow \acute{a} - q$

5: $\quad$ **return** $\acute{a}$