2016

Efficient Multiplication Architectures for Truncated Polynomial Ring

Ruiqing Dong
University of Windsor

Efficient Multiplication Architectures for Truncated Polynomial Ring

by

Ruiqing Dong

A Thesis

Submitted to the Faculty of Graduate Studies

through Electrical and Computer Engineering

in Partial Fulfilment of the Requirements for

the Degree of Master of Applied Science

at the University of Windsor

Windsor, Ontario, Canada

2016

Efficient Multiplication Architectures for Truncated Polynomial Ring

by

Ruiqing Dong

APPROVED BY:

Dr. L. Rueda

School of Computer Science

Dr. C. Chen

Department of Electrical and Computer Engineering

Dr. H. Wu, Advisor

Department of Electrical and Computer Engineering

September 21, 2016

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyones copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

In this thesis, four efficient multiplication architectures, named as Multipliers I, II, III, and IV, respectively, for truncated polynomial ring are proposed. Their FPGA implementation results are presented. All of the four proposed multipliers can be used for implementation of NTRUEncrypt public key system.

All new multiplication architectures are based on certain extensions to Linear Feedback Shift Register (LFSR). Multiplier I uses x^2-net structure for LFSR, which scans two consecutive coefficients in the control input polynomial $r(x)$ during one clock cycle. In Multiplier II, three consecutive zeros in the control input polynomial $r(x)$ can be processed during one clock cycle. Multiplier III takes advantage of consecutive zeros in the control input polynomial $r(x)$. Multiplier IV is resistant to certain side-channel attacks through controlling the operations for each clock cycle.

An FPGA complexity comparison among the proposed multipliers and the existing similar works is made, including number of adaptive logic modules (ALMs), number of registers, number of cycles, maximum operating frequency (FMax) and latency.

The FPGA comparison results are given as follows. Multiplier I has smaller latency than any existing works when the first set of parameters from every security level is used ($ees401ep1$, $ees449ep1$, $ees677ep1$, $ees1087ep2$).

Multiplier II is the second best in speed compared to existing works, but has better area-latency product compared to the fastest existing work for the first set of parameters at security level 112-bit, 128-bit and 192-bit.

As an enhanced version of Multiplier II, Multiplier III is faster than any existing works in comparison for all IEEE recommended parameter sets.

Multiplier IV, designed to be resistant to side channel attacks, also has high speed property that it outperforms all the existing works in terms of latency for all three parameter sets to which it is applicable.

DEDICATION

To my loving parents:

Father: Shouqiang Dong

Mother: Liping Chen

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to everyone who helped me make this thesis possible. First of all, I am deeply indebted to my supervisor Dr. Huapeng Wu, who gave me a big help during my study for the master degree. He guided me throughout the writing of this thesis. His broad knowledge and logical thinking have been of great value. Without his detailed and constructive comments on my research, none of this thesis would be possible. In addition, I would like to show my gratitude to my committee members, Dr. Chunhong Chen and Dr. Luis Rueda, for their attendance and advice in my seminar and defense. Additionally, I would like to appreciate my loving parents. Without their support and encouragement, it is impossible for me to achieve such accomplishment. I also feel grateful to my colleagues and friends, Bingxin Liu, Siyu Zhang, Yaqiong Xu, Huifeng Dong and Yu Shen for their support and help to my research and thesis. Finally, I wish to extend my gratitude to everyone at UWindsor's Faculty of Electrical and Computer Engineering for their efforts during my study in the M.A.Sc. Program. Moreover, I gratefully thank for the financial support from University of Windsor and my supervisor Dr. Huapeng Wu.

Ruiqing Dong

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ACRONYMS

ALM	Adaptive logic module
CCSSG	Cloud Computing Standard Study Group
CRT	Chinese Remainder Theorem
CSA	Cloud Security Alliance
ECC	Elliptic Curve Cryptosystem
FMax	Maximum operating frequency
IaaS	Infrastructure as a Service
IT	Information Technology
LFSR	Linear Feedback Shift Register
LUT	Look-up table
LWE	Learning with errors
NIST	National Institute of Standards and Technology
NTRU	Nth Degree Truncated Polynomial Ring Unit
PaaS	Platform as a Service
SaaS	Software as a Service

1 INTRODUCTION

1.1 Motivation

The Internet technology has an enormous impact on many aspects of our daily life. One of the biggest advantages of the Internet is its easy access to information and services. Rather than searching in libraries, users can get access to vast amounts of information from their personal computers. In addition, the Internet offers different kinds of services, sometimes for free. Such advantages make our life easier and cheaper.

Since the Internet is accessible to all users, including the hackers, security breach becomes a practical issue. Cyber security is a technology suite that provides protection for data from theft or damage as well as from disruption or unauthorized access over cyber space including the Internet. Among a variety of technologies to provide cyber security, cryptography is probably the most important and effective one.

Cryptography, as the core technology for cyber security, can provide many essential and unique security services, such like confidentiality, authentication, data integrity and non-repudiation. For example, confidentiality is a security service that can prevent third parties from obtaining users' private information during communication.

The modern cryptographic technology can be divided into two parts: symmetric-key cryptography and public-key cryptography. A symmetrical-key cryptosystem requires the same shared keys for both encryption and decryption. Since the shared keys should be kept only by involved parties and not accessible to third parties, such system requires a secure communication channel for the key exchange between senders and receivers. This strict requirement becomes the main issue for symmetrical-key cryptography.

Stream ciphers and block ciphers are two major types of symmetric-key cryptosystem. Stream ciphers take plaintext input in the form of a stream of bits or digits and have simple encryption and decryption operations. In addition, key size in stream ciphers is very large and considered much larger than that in block ciphers. In block ciphers, plaintext is encrypted in a multiple of blocks while the encryption/decryption operations are more complicated. Examples for popular symmetric-key cryptosystems can be Trivium for stream cipher and AES for block cipher.

Public-key cryptography, also called asymmetric-key cryptography, is a class of cryptographic algorithms that use two different keys for encryption and decryption respectively. A public-key cryptosystem has two keys: a public key and a private key. The public key can be known to anyone including those who would like to send confidential messages to the cryptosystem owner (the receiver) and the private key is only known to the receiver. Public-key cryptography is proposed to address two main issues: key distribution, which is used to exchange keys without having to trust a third party, and digital signature, which is used to verify the intactness of a message from a claimed sender. There are several public-key cryptosystems which are frequently used currently, such as RSA and Elliptic Curve Cryptosystem (ECC). Compared with symmetric-key cryptosystems, public-key cryptosystems usually have higher computational complexity while provide unique security services in key management and digital signature, which makes them indispensable in cyber security.

However, with the emerging of quantum computing technology, some existing public-key cryptosystems, such as RSA and ECC, have been shown to be insecure [4]. Quantum computers are totally different from traditional electronic computers in that the former use quantum-mechanical phenomena to perform operations on data and have super faster computational speed. As quantum computers are expected to be practically available in the near future, it is necessary for us to study on the cryptographic technologies that can remain secure in the age of quantum computing. This research field is called post-quantum cryptography, which includes cryptographic algorithms that are considered to be secure under the attacks launched from quantum computers [5].

One of the most popularly researched areas of post-quantum cryptography is lattice-based cryptography, which depends on the worst-case hardness of lattice problems. Nth Degree Truncated Polynomial Ring Unit (NTRU), as one of the lattice-based systems, is probably the most practical and widely researched existing post-quantum cryptosystem.

NTRU was first proposed by J. Hoffstein et al. in 1996 [6]. In 2009, NTRU was regulated in IEEE P1363.1 standard [1]. NTRU consists of two algorithms: NTRUEncrypt and NTRUSign. NTRUEncrypt is used for encryption and decryption while NTRUSign is applied to digital signature. In addition, due to its high speed and low memory use, NTRU can be used in variety of applications such as mobile devices and smart cards. Its efficiency, as well as its resistance to quantum computer based attacks, makes NTRU and its efficient implementations a hot research topic in public-key cryptography.

Recently some new requirements for cloud computing are proposed in which the most critical one is that users prefer to delegate computations to untrusted computers. In specific, users would like to give untrusted computers only an encrypted version of data to process the computations. Considering confidentiality of the data, it is necessary for untrusted computers to perform the computations on encrypted data without knowing any part of the plaintext. The computation with untrusted computers yields results in encrypted form, which is then sent to the user for decryption. For the correctness of the whole computation, the decrypted result has to be equivalent to the intended value performed on the original data. This property is called homomorphism and such encryption schemes are referred to as homomorphic encryption.

Generally speaking, there are two main types of homomorphic encryption: partial homomorphic encryption and fully homomorphic encryption. Partial homomorphic encryption allows either addition or multiplication operation while fully homomorphic encryption supports arbitrary computations on ciphertext. Partial homomorphic encryption, such as RSA and ElGamal, took the lead in study of homomorphic encryption until 2009. In 2009, C. Gentry proposed the first fully homomorphic encryption scheme which showed a new path for the research of homomorphic encryption [7]. D. Stehlé et al. proposed a revised NTRUEncrypt scheme which can provide higher level of security in 2011 [8]. One year later, A. López-Alt et al. found the fully homomorphic property in the revised NTRUEncrypt scheme and proposed a multi-key fully homomorphic encryption scheme called the LTV homomorphic encryption scheme [9]. The application of NTRU to cloud computing security makes it one of the most interested research topics in cryptography.

1.2 Main Contributions

In this thesis, several efficient truncated polynomial ring multiplication architectures are proposed. Their FPGA implementations are presented and compared with existing similar works. A summary of the proposed works is given below.

- Multiplier I, which is presented in Chapter 5 Section 1 (§5.1), is a high-speed polynomial multiplier. The FPGA results show that latency of the proposed work is only 63.52%, 60.62%, 61.20%, and 93.53%, compared to the fastest among the existing similar works for IEEE recommended parameter sets $ees401ep1$, $ees449ep1$, $ees677ep1$, and $ees1087ep2$, respectively. Note that the above-mentioned parameters sets are corresponding to different security levels, such like 112-bit, 128-bit, 192-bit, and 256-bit.

- Multiplier II is presented in Chapter 5 Section 2 (§5.2). The architecture takes advantage of three consecutive zeros in polynomial coefficients by re-coding the polynomial $r(x)$. It is the second best in speed compared to existing works, but has better area-latency-product compared to the fastest existing work for the first set of parameters $ees401ep1$, $ees449ep1$, $ees677ep1$, at security level 112-bit, 128-bit and 192-bit, respectively.

- Multiplier III is presented in Chapter 5 Section 3 (§5.3). The architecture takes advantage of consecutive zeros in polynomial coefficients by re-coding the polynomial $r(x)$. Its FPGA results show that it is faster than any existing works in comparison for all IEEE recommended parameter sets.

- A new efficient polynomial multiplier resistant to certain side channel attacks, which we call it Multiplier IV, is presented in Chapter 6. It outperforms all the existing works in terms of latency for all three parameter sets $ees401ep1$, $ees449ep1$ and $ees677ep1$.

Note that a comprehensive list of comparisons of FPGA results of proposed multipliers with existing similar works is provided in Appendix A.

1.3 Thesis Organization

An organization of the rest of this thesis is as follows. Chapter 2 presents mathematical background of NTRUEncrypt system. A brief overview of existing works is given in Chapter 3. Chapter 4 introduces a brief overview of cloud computing and homomorphic encryption, especially the LTV homomorphic encryption scheme. Chapter 5 proposes three new truncated polynomial multiplier architectures and their FPGA results. In addition, a comparison of the proposed works with several existing works is given. Chapter 6 proposes a highly efficient multiplication architecture, which is also resistant to certain side channel attacks. Its FPGA simulation results are presented and a complexity comparison is made in the same chapter. In Chapter 7, a few conclusional remarks are given and possible future works are discussed.

2 MATHEMATICAL PRIMITIVES

This chapter first introduces truncated polynomial ring, as the mathematical background of NTRUEncrypt system. Then an overview of NTRUEncrypt system is given, which includes key generation, encryption, and decryption. The parameter selection for NTRUEncrypt cryptosystem is also discussed.

2.1 Truncated Polynomial Ring

NTRUEncrypt system is defined over a special algebraic structure, the truncated polynomial ring, which is denoted by $R = \mathbb{Z}[x]/(x^N - 1)$, where N is a prime number. In this notation, $\mathbb{Z}[x]$ denotes the set of polynomials of arbitrary degree with integer coefficients and $x^N - 1$ is the modulo polynomial. It can be seen that the set of R contains all the polynomials with integer coefficients of degree up to $N - 1$.

Let $f(x)$ be a polynomial belonging to R. Then it can be represented as

$$f(x) = f_{N-1}x^{N-1} + f_{N-2}x^{N-2} + ... + f_1 x + f_0,\ f_i \in \mathrm{Z} \text{ for i} = 0, 1, ..., \mathrm{N} - 1 \tag{1}$$

Arithmetic operations on $R = \mathbb{Z}[x]/(x^N - 1)$ includes addition and multiplication which are discussed in the following two subsections.

2.1.1 Addition

Let $a(x), b(x) \in R$ be given as

$$a(x) = a_{N-1}x^{N-1} + a_{N-2}x^{N-2} + ... + a_0 \tag{2}$$

$$b(x) = b_{N-1}x^{N-1} + b_{N-2}x^{N-2} + ... + b_0 \tag{3}$$

Then addition operation between $a(x)$ and $b(x)$ is given by

$$a(x) + b(x) = (a_{N-1} + b_{N-1})x^{N-1} + (a_{N-2} + b_{N-2})x^{N-2} + ... + (a_0 + b_0) \tag{4}$$

2.1.2 Multiplication

Let $a(x), b(x) \in R$ be given as

$$a(x) = a_{N-1}x^{N-1} + a_{N-2}x^{N-2} + ... + a_0 \tag{5}$$

$$b(x) = b_{N-1}x^{N-1} + b_{N-2}x^{N-2} + ... + b_0 \tag{6}$$

Let $c(x)$ be the product polynomial of $a(x)$ and $b(x)$. Then

$$a(x) \times b(x) = c(x) = c_{N-1}x^{N-1} + c_{N-2}x^{N-2} + ... + c_0 \tag{7}$$

where

$$c_k = \sum_{\substack{i,j=0,1,...,N-1 \\ i+j=k \mod N}} a_i b_j, k = 0, 1, ..., N-1 \tag{8}$$

Besides the limitation on the degree of polynomials, NTRUEncrypt system requires a certain restriction on polynomial coefficients. The operations in NTRUEncrypt system are performed in the truncated polynomial ring R modulo a fixed number q, which indicates that all polynomial coefficients should be from integer set $\{0, 1, 2, ..., q-1\}$.

Therefore, NTRUEncrypt cryptosystem works in the truncated polynomial ring R with modulo q, which is denoted by $R_q = \mathbb{Z}_q[x]/(x^N - 1)$.

2.2 NTRUEncrypt Cryptosystem

NTRUEncrypt cryptosystem includes totally four parts: parameter selection, key generation, encryption and decryption. Each part is discussed in this section in detail.

2.2.1 Parameter Selection

Basically, the NTRUEncrypt is a parameterized cryptosystem where the truncated polynomial ring R is determined by three integer parameters N, p and q. N is a prime number which is used to determine the degree of the truncated polynomials in R. p and q define two moduli where $p << q$. Additionally, p and q must be relatively prime. Besides these three integer parameters, three more integers d_f, d_g and d_r are selected to determine three corresponding polynomial sets L_f, L_g and L_r for NTRUEncrypt system. All these three polynomial sets are denoted in the form of $L(d_1, d_2)$, which means that a polynomial in $L(d_1, d_2)$ has d_1 coefficients equal to 1, d_2 coefficients equal to -1 and the rest coefficients equal to 0. In specific, we have

$$L_f = L(d_f, d_f - 1),\ L_g = L(d_g, d_g),\ L_r = L(d_r, d_r)$$

In 2009, a parameter generation function for NTRUEncrypt system were proposed in [3], which is shown in Algorithm 2.1. In this algorithm, security level, which represents the security strength, is taken as the input and the output is the parameter set of N and d_f. Note that there are two functions used in this algorithm. One is $hybridSecurityEstimate(n, d_f)$, which is used to estimate the minimal security over all attack strategies on certain parameter set while the other one is $decryptionFailureProb(n, d_f)$, which is used to estimate the chance of a decryption failure on certain parameter set.

Algorithm 2.1 Parameter Generation Function for NTRUEncrypt [3]

```
Input: Security Level, $k$
Output: Parameter Set $(n, d_f)$
1: $i \leftarrow 1$ {The variable $i$ is used to index the set of acceptable primes $\mathcal{P}$}
2: $i* \leftarrow 0$ {This will become the first index which can achieve the required security}
3: repeat
4:    $n \leftarrow \mathcal{P}_i$
5:    $d_f \leftarrow [n/3]$ {We will try each $d_f$ from $[n/3]$ down to 1}
6:    repeat
7:       $k_1 \leftarrow$ hybridSecurityEstimate($n$,$d_f$)
8:       $k_2 \leftarrow \log_2$(decryptionFailureProb($n$,$d_f$))
9:       if ($k_1 \geq k$ and $k_2 < -k$) then
10:         $(i^*, d_f^*) \leftarrow (i, d_f)$ {Record the first acceptable index $i$ and the value of $d_f$}
11:      end if
12:      $d_f \leftarrow d_f - 1$
13:   until $i^* > 0$ or $d_f < 1$
14:   $i \leftarrow i + 1$
15: until $i^* > 0$
16: $c^* \leftarrow cost(\mathcal{P}_{i^*}, d_f^*)$
17: while an increase in N can potentially lower the cost do
18:   $n \leftarrow \mathcal{P}_i$
19:   $d_f \leftarrow d_f^*$ {Note that when $n$ increases the cost must be worse for all $d_f \geq d_f^*$,and
      that the decryption failure probability is decreased both by an increase in $n$
      and a decrease in $d_f$}
20:   repeat
21:      $k_1 \leftarrow$ hybridSecurityEstimate($n$,$d_f$)
22:      $c \leftarrow cost(n, d_f)$
23:      if ($k_1 \geq k$ and $c < c^*$) then
24:         $(c^*, i^*, d_f^*) \leftarrow (c, i, d_f)$ {Record the the improvement in cost and the corre-
            sponding $i$, $d_f$}
25:      end if
26:      $d_f \leftarrow d_f - 1$
27:   until $d_f < 0$
28:   $i \leftarrow i + 1$
29: end while
30: return $(\mathcal{P}_{i^*}, d_f^*)$
```

A table of recommended parameters for NTRUEncrypt system can be generated with Algorithm 2.1, which is given in Table 2.1. From the table, it can be seen that there are totally four security levels for NTRUEncrypt system and in each security level, three parameter sets are presented. Note that the parameter q is assigned 2048 for all the parameter sets. It is in the form of 2^n to allow highly efficient computation of the modular operation. Since the parameter p and the parameter q are coprime and p has to be much smaller than q, p is selected as 3, which is the smallest odd prime number. It is also worthy to note that d_f and d_r are assigned to the same value. The security level of NTRUEncrypt cryptosystem is determined by the parameters N, d_f, d_g and d_r.

Table 2.1: Recommended Parameters for NTRUEncrypt [1]

Security Level	Parameter set	n	p	q	d_f	d_g	d_r
112	$ees401ep1$	401	3	2048	113	133	113
	$ees541ep1$	541	3	2048	49	180	49
	$ees659ep1$	659	3	2048	38	219	38
128	$ees449ep1$	449	3	2048	134	149	134
	$ees613ep1$	613	3	2048	55	204	55
	$ees761ep1$	761	3	2048	42	253	42
192	$ees677ep1$	677	3	2048	157	225	157
	$ees887ep1$	887	3	2048	81	295	81
	$ees1087ep1$	1087	3	2048	63	362	63
256	$ees1087ep2$	1087	3	2048	120	367	120
	$ees1171ep1$	1171	3	2048	106	390	106
	$ees1499ep1$	1499	3	2048	79	499	79

2.2.2 Key Generation

As a public-key cryptosystem, NTRUEncrypt system requires the generation of a public key and a private key. The public key is known to the public and the private key is only known by the receiver. The detailed key generation process is given as follows.

Step 1: Randomly choose a polynomial $f(x)$, which is invertible in R_q and R_p, from the polynomial set L_f.

Step 2: Randomly choose a polynomial $g(x)$ from the polynomial set L_g.

Step 3: Calculate $f_q(x)$ and $f_p(x)$ which are the inverse of polynomial $f(x) \mod q$ and $f(x) \mod p$.

Step 4: Compute $h(x) = p \cdot f_q(x) \times g(x) \mod q$.

After the above four steps, the public key $h(x)$ and the corresponding private key pair $(f(x), f_p(x))$ can be obtained.

2.2.3 Encryption

In order to prepare a confidential message m intended to be sent to the receiver, the sender is supposed to follow the encryption steps which are listed below.

Step 1: Encode the message m into a polynomial $m(x)$ with coefficients from $\{-1, 0, 1\}$.

Step 2: Randomly choose a polynomials $r(x)$ from the polynomial set L_r.

Step 3: Encrypt message by performing $e(x) = h(x) \times r(x) + m(x) \mod q$

After the above three steps, $e(x)$ is the ciphertext to be sent to the receiver.

2.2.4 Decryption

After receiving the encrypted message $e(x)$ from the sender, the receiver is supposed to follow the decryption steps to obtain the original plaintext.

Step 1: Compute $a(x) = f(x) \times e(x) \mod q$.

Step 2: Shift coefficients of $a(x)$ to the range $\left[-\frac{q}{2}, \frac{q}{2}\right]$

Step 3: Compute $m(x) = f_p(x) \times a(x) \mod p$

After the completion of the three steps, $m(x)$ is the plaintext sent from the sender.

3 AN OVERVIEW OF EXISTING WORKS

Since NTRU cryptosystem was invented in 1996 by J. Hoffstein et al., it has been a popularly researched area in the past twenty years. Compared with other popular public-key cryptosystems, such as RSA and ECC, NTRUEncrypt system has been shown more efficient in speed and memory usage [6]. In the last ten years, optimizations have been made on NTRU cryptosystem in order to improve either area efficiency or speed efficiency [10] [11] [12]. Meanwhile, several architectures focus on the reduction of power consumption [13] [14]. In this chapter, we will give an overview of the existing works on hardware architectures for NTRU system and their implementations.

The first hardware based implementation of NTRU system was reported in [10]. In this work, NTRU was implemented in software running on several different constrained devices including the Palm Computing Platform, Advanced RISC Machines ARM7TDMI and the Research in Motion Pager. In addition, it was implemented in FPGA, which belongs to hardware implementation. In this implementation, the parameter set $(N, p, q) = (251, X + 2, 128)$ is used. The whole encryption process is operated within the following steps. First of all, the operands $h(x)$, $r(x)$ and $m(x)$ are loaded serially. Then, the system begins to scan the bits of each coefficient of $r(x)$ consecutively and adds $h(x)$ to the intermediate results for each non-zero coefficient. After this procedure is done or for those coefficients equal to zero, $h(x)$ is shifted to left by one coefficient. After all the coefficients of $r(x)$ are scanned, the engine adds the message polynomial $m(x)$ to the previous results to obtain the final encryption results. Without using full coefficient multiplication, this implementation utilizes repeated coefficient addition, which improves the efficiency of the whole system.

C. M. O'Rourke proposed three software and hardware designs for NTRU cryptosystem in his master thesis [11]. The first implementation is based on software, which shows the practicality of applying Chinese Remainder Theorem (CRT) to the convolution algorithm. Although the result is not as good as the expectation, it indicates the probability of applying CRT to the computation of inverse polynomials. The second implementation uses a scalable NTRU multiplier, which can be used in the procedures including key generation, encryption and decryption. The core technology of such multiplier is the parallelism in the polynomial multiplication. The most significant contribution of this design is that it provides a variety of time-area configurations. Either time or area consumption can be focused on in practical, depending on the certain applications. Additionally, the multiplier can support arbitrary key length to meet the requirements of different security level. Finally, another architecture is presented by exploiting the Montgomery multiplication algorithm. Three kinds of popular public-key cryptosystems, RSA, ECC and NTRU, are implemented. The results indicate that for the same security level, NTRU shows the best performance among three.

A new design of NTRUEncrypt system with FPGA implementation was presented in 2009 [12]. Specifically, this architecture uses the statistical properties of the distance between the non-zero elements in the polynomials in both encryption and decryption. In the decryption, two different methods are used for mod p operation. One is using Mersenne primes algorithm and the other is using look-up tables (LUTs). This architecture also takes advantage of large number of zero coefficients in certain polynomial. In order to speed up the whole system, it utilizes an s-bit shifter to skip the zero coefficients, where s is much smaller than the parameter N. Through modifying the value of s, this hardware implementation can offer different area-speed trade-offs for the NTRUEncrypt system.

An efficient hardware architecture and FPGA implementation for NTRUEncrypt was proposed in [15]. In this paper, a new breakthrough is made by using LFSR structure due to its compact circuitry and high speed. Firstly, a new truncated polynomial ring multiplier is designed, which is based on the LFSR structure. Then, in order to optimize the system, a new modular arithmetic unit is introduced, which takes advantage of the property of coefficients in polynomial $r(x)$. As a result, the multiplication operations can be replaced with only shifting and addition operationsn. Then, a new LFSR based NTRUEncrypt system is proposed. The FPGA implementation results indicate that such architecture offers modest area consumption and relatively high speed. Compared with several existing works, it shows the best performance in term of area-latency product.

B. Liu proposed another efficient multiplication architecture over truncated polynomial ring for NTRUEncrypt system [16]. The proposed architecture is based on an extended LFSR. The general idea of this architecture is that a considerable reduction in the number of clock cycles can be achieved if two consecutive zero coefficients in the polynomial $r(x)$ can be processed during one clock cycle. In order to increase the utilization of each clock cycle, a new modular arithmetic unit is designed, where the redundant state of the input from $r(x)$ is used. Then, an extended LFSR based NTRUEncrypt structure can be obtained. Although the implementation results show that this design utilizes slightly more area resource, compared with the LFSR based architecture, it has a higher speed. Moreover, the product of area and latency of this architecture is the lowest among all the similar existing works.

4 NTRU BASED HOMOMORPHIC ENCRYPTION AND ITS APPLICATIONS IN CLOUD COMPUTING

4.1 Cloud Computing

Currently cloud computing is a popularly researched field of Information Technology (IT). According to the National Institute of Standards and Technology (NIST), cloud computing is defined as a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources, such as networks, servers, storage, applications, and services, which can be rapidly provisioned and released with minimal management efforts or service provider interaction.

Compared with conventional computing models where end-user data and computing power are located in users' computer systems, cloud computing resources are provided in massive, abstracted infrastructures managed by professional service providers, which makes it simpler and more convenient for users to operate, store and maintain their data.

4.1.1 Development of Cloud Computing

The first generation of cloud computing, which is called Cloud 1.0, originated from the abstraction of TCP/IP layers, where network devices communicate with one another by complying with TCP/IP protocol specifications without knowing exactly where and who the other one is.

Cloud 2.0 came out from the abstraction of World Wide Web data, where documents can be published and retrieved by users from the web without knowing beforehand exactly where they are located or who published them.

The current version of cloud computing is named Cloud 3.0, which is the abstraction of infrastructure complexities of servers, applications, data, and heterogeneous platforms where infrastructure, servers or applications can be used without knowing their exact location.

4.1.2 Security Issues in Different Service Models

Cloud computing utilizes three delivery models by which different types of services are delivered to end users. The three delivery models are Software as a Service (SaaS), Infrastructure as a Service (IaaS) and Platform as a Service (PaaS), which provide software, infrastructure resources and application platforms to consumers respectively.

SaaS is a software deployment model where applications are remotely hosted by service providers. It is available to customers on demand over the Internet. SaaS offers numerous benefits to customers, such as operational efficiency and reduced costs. These advantages make it as a dominant delivery model in some companies. However, many enterprises still worry about its security levels and thus, hesitate to put it into practice mainly due to the opacity of its data storage and security. In SaaS, users have to rely on providers for proper security measures. Providers must prevent multiple users from gathering others' data. Therefore, it is difficult for users to ensure that right security measures are in place and applications are available when needed. A variety of security elements are supposed to be taken into consideration before SaaS could be accepted more widely, such as data security, network security, data locality, data integrity and data authentication.

Compared with SaaS, IaaS completely changes the way developers deploy their applications. Instead of spending big efforts building their own data centers or deploying services, users can get access to an IaaS provider, such as Amazon Web Services, to get access to a virtual server. Using the IaaS model, users no longer need to focus on the maintenance and the management of infrastructure. On the other hand, security responsibilities are divided into two parts between vendors and customers. The former is responsible for the hardware layers while the later are supposed to control the software security.

PaaS is a layer above IaaS. It offers developers a service which provides a complete software development lifecycle management to build their own applications. Such service is based on software platforms. Hence, the security below the platform should be considered seriously by providers. Vulnerabilities are supposed to appear in both web applications and machine-to-machine service-oriented architecture applications.

4.1.3 Security Challenges in Cloud Computing

Although cloud computing brings us great benefits, there are a variety of security problems in cloud computing that need to be addressed.

Some traditional security challenges are required to be reconsider by cloud providers. Firstly, working environment is required to be changed to fit the environment of cloud computing, especially in terms of authentication and authorization, since in cloud computing, users are unable to access to the system hardware physically. Additionally, availability of cloud services is a big issue since the disruption of cloud services is much more serious than that of traditional service models. Last but not least, as virtual machines may also contain vulnerabilities, security of virtual machines should be concerned about as well.

On the other hand, many new security challenges are springing out with the development of cloud computing. Since end-users of cloud computing are supposed to store their data in cloud providers' infrastructure, the most critical security challenge is data privacy and confidentiality. Cloud users would like to know where their information is stored and who is in charge of their information. Meanwhile, they would like to be guaranteed that their personnal information is unaccessible to others, even to the cloud providers. Secondly, establishing a trusted security monitoring system is another big issue. Compared with Internet monitoring management system, cloud computing monitoring system requires three factors to be considered: rapid identification, warning and protection of security attacks based on cloud computing, content monitoring of cloud computing and identification and protection of cryptographic crimes. In addition, cloud standards are required to be set up to achieve interoperability and stability. There are several groups working on cloud computing specifications, like IEEE Cloud Computing Standard Study Group (CCSSG) and Cloud Security Alliance (CSA). Yet, no common standard has been set up and accepted jointly.

4.1.4 Several Current Solutions

Regarding security requirements of cloud computing, data storage and sharing are two main aspects which are concerned by cloud users, especially those companies who require strict secure data processing in order to protect their business interests. Several mechanisms and techniques have been proposed to solve these two troublesome security issues.

The key point of secure data storage is that data stored on the cloud should be extremely private and no one can access to or control the data except the data owners, even the providers have no rights to deal with the data. [17] suggests resource isolation to ensure data security by isolating the processor caches in virtual machines and isolating those virtual caches from the hypervisor cache. Another solution which is stated to be widely used in UK business is to use in-house private clouds. A private cloud is a software-defined data center that combines essential hardware and other computing resources into a unified virtualized unit. A private cloud's hardware and networking abstraction layer, which is provided by software, enables companies to scale and provide resources more dynamically. Compared with a public cloud, a private cloud is hosted within a company's firewall. A private cloud allows business to save time on hardware-based deployments. It also provides greater flexibility, security and privacy.

Secure data sharing is another critical issue in cloud computing. Homomorphic encryption and incremental encryption can be utilized to ensure the confidentiality during the transmission of users' data. Homomorphic encryption is a form of encryption that allows computations to be carried out on ciphertext, thus generating an encrypted result which, when decrypted, matches the result of operations performed on the plaintext. Such encryption makes it possible for multiple parties to cooperatively generate a piece of ciphertext without knowing the plaintext that others are working on. The general idea of incremental encryption is that if we have already computed the function on a document, and this document is modified, then we update the function value based on the old value rather than re-computing it from scratch. In data sharing, the encrypted data is re-encrypted without being decrypted first. The re-encrypted data is accessible only to the authorized users. This kind of encryption has a great advantage that in sharing process, the data is always in its encrypted form, which means there is no single stage that the data is decrypted before it is delivered to the authorized users.

4.2 Homomorphic Encryption

Basically, the goal of encryption is to ensure confidentiality of the data in the process transmission and storage. However, some new requirements for cloud computing are proposed in which the most critical one is that users prefer to delegate computations to untrusted computers. Specifically, users would like to give untrusted computers only an encrypted version of data to process the computations. Considering confidentiality of the data, it is necessary for untrusted computers to perform the computations on encrypted data without knowing any part of the plaintext. The computation with untrusted computers yields results in encrypted form, which is then sent to the user for decryption. For the correctness of the whole computation, the decrypted result has to be equivalent to the intended value performed on the original data. This property is called homomorphism and such encryption schemes are referred to as homomorphic encryption.

4.2.1 Development of Homomorphic Encryption

The notion of homomorphic encryption was first proposed by R. L. Rivest et al. in 1978 [18]. They proposed the exponentiation function and the RSA function as additive and multiplicative privacy homomorphism respectively, which made the first generation of homomorphic encryption known to the world. However, the security level of these functions was so low that they cannot even prevent the chosen plaintext attack.

In 1982, S. Goldwasser et al. first proposed a homomorphic encryption scheme satisfying semantic security [19]. After that, a number of homomorphic encryption schemes were proposed, including ElGamal encryption scheme, Paillier encryption scheme and so forth. All these encryption schemes support either additive homomorphism or multiplicative homomorphism. Therefore, they are called partial homomorphic encryption.

A new question rose in 1991 when J. Feigenbaum et al. proposed an important idea that if there exists an encryption function E that is both additively and multiplicatively homomorphic [20]. Basically, such schemes are called fully homomorphic encryption.

C. Gentry proposed the first fully homomorphic encryption scheme that enables the computations of arbitrary functions on encrypted data and producing compact ciphertexts in 2009 [7]. It showed a new path for study on fully homomorphic encryption.

4.2.2 Two Types of Homomorphic Encryption

The definition of homomorphic encryption is as follows: an encryption scheme is said to be homomorphic if for any given encryption key k, the encryption function E satisfies $E_k(m_1 \triangle m_2) = E_k(m_1) \square E_k(m_2)$, where $\triangle$, $\square$ denote either addition or multiplication, and m_1, m_2 stand for two messages.

Partial Homomorphic Encryption

In the early age of homomorphic encryption, several public key encryption schemes were shown to meet the homomorphic properties in terms of either addition or multiplication. Such encryption schemes are called partial homomorphic encryption.

At first, RSA was immediately shown to satisfy the multiplicative homomorphism. In 1984, T. ElGamal proposed a new public-key cryptosystem, which was shown to be multiplicatively homomorphic as well [21].

Another well-known homomorphic encryption is Goldwasser-Micali whose homomorphism is based on XOR operation [19]. Goldwasser-Micali is an encryption algorithm based on the problem of quadratic residue. In addition, along with this encryption scheme, S. Goldwasser et al. proposed a rigorous definition of semantic security. According to their definition, a cryptosystem is semantically secure if the knowledge and the length of the ciphertext do not reveal any additional information on the message that can be feasibly extracted. Furthermore, S. Goldwasser et al. showed the equivalence of semantic security and indistinguishability under chosen plaintext attack, which is considered as a basic requirement for secure public key cryptosystems.

One of the most effective additive homomorphic encryption is Paillier. It was proposed by P. Paillier in 1999 [22]. This cryptosystem is based on decisional composite residuosity assumption. Furthermore, Paillier is shown to be semantically secure. It is one of the most efficient additively homomorphic encryption schemes.

Partial homomorphic encryption is well-known for its high efficiency. Some partial homomorphic encryption schemes are efficient enough for practical applications. However, some restrictions still exist. The main problem is the limited circuits that they support. Since partial homomorphic encryption schemes only support one type of operation, basically either addition or multiplication, circuits' evaluation is limited in the range of single operation.

Fully Homomorphic Encryption

Partial homomorphic encryption took the lead in certain research field until 2009. C. Gentry proposed the first fully homomorphic encryption scheme which can evaluate arbitrary additions and multiplications [7]. His fully homomorphic encryption is based on three components: a somewhat homomorphic encryption scheme that can evaluate a limited class of functions, a sufficiently powerful homomorphic encryption scheme called bootstrappable encryption scheme and finally, to connect the dots, a specialized method to turn the somewhat homomorphic scheme into a bootstrappable scheme. In his construction, he uses ideal lattices to construct somewhat homomorphic encryption scheme. Therefore, this scheme is also called a lattice-based cryptosystem. However, this scheme is considered as impractical in that ciphertext size and computation time increase rapidly with the growth of security level.

On the other hand, Gentry's work created a new path for the development of fully homomorphic encryption. Several optimizations on Gentry's work were proposed in [23], [24], [25] and [26]. Meanwhile, some more efficient schemes were proposed, such as [27], [28], [29] and [30], most of which are based on the hardness of the learning with errors (LWE) problem [31].

Compared with partial homomorphic encryption, fully homomorphic encryption is advantageous in the evaluation of circuits. A practical and efficient fully homomorphic encryption scheme can support both additive and multiplicative operations unlimitedly. Nevertheless, until nowadays, fully homomorphic encryption is still far away from practical applications due to its low computation speed and large ciphertext size.

4.3 The LTV Homomorphic Encryption Scheme

A. López-Alt et al. proposed a multikey fully homomorphic encryption scheme called the LTV homomorphic encryption scheme in 2012, which is based on NTRU public-key cryptosystem [9]. This scheme is capable of operating on inputs encrypted under multiple, unrelated keys. A ciphertext resulting from a multikey evaluation can be jointly decrypted using the secret keys of all the users involved in the computation. A somewhat homomorphic encryption scheme based on NTRU is proposed at first. Then such somewhat homomorphic encryption scheme is transformed to a fully homomorphic encryption scheme. This scheme is expected to be a leading candidate for a practical fully homomorphic encryption scheme.

For brevity, we focus on the single user case for the LTV scheme in the following presentation.

4.3.1 Somewhat Homomorphic Encryption Scheme

In this section, the primitives of the somewhat homomorphic LTV encryption scheme are introduced, including parameter selection, key generation, encryption and decryption. Besides, homomorphic properties of such scheme are discussed as well.

Parameter Selection

D. Cabarcas et al. proposed a method of parameter selection for NTRU based homomorphic encryption, which can be applied to the LTV homomorphic encryption scheme in 2014 [2].

The parameters required in the LTV homomorphic encryption are listed as follows.

- An integer n.
- A prime number q.
- A degree-n polynomial $\varphi(x) = x^n + 1$.
- An error distribution χ, which is the truncated discrete Gaussian distribution $D_{Z^n,r}$ for standard deviation $r > 0$.

It is pointed out that the message space for the LTV homomorphic encryption is $M = \{0, 1\}$ and all the operations are carried out in the ring $R_q = \mathbb{Z}_q[x]/(\varphi(x))$. However, in order to prevent a wrap-around error, all the coefficients are required to located in the range $[-q/2, q/2]$.

Some specifications are required in parameter selection, which are listed as follows.

- Set n to be a power of two.
- Set $q \in [dn^6 \ln(n), 2dn^6 \ln(n)]$, such that $q \equiv 1 \mod 2n$. For $d = 25830$, correctness of the scheme can be guaranteed.
- Set $r = \sqrt{2n/\pi}$.

Some parameter sets are listed in Table 4.1.

Table 4.1: Some Parameter Sets for the LTV Homomorphic Encryption [2]

Security Level	n	$\log_2 q$	r
38	1024	71.90	25.53
144	2048	77.28	36.11
138	4096	83.30	51.06

Key Generation

The key generation process is shown as follows.

Step 1: Sample polynomials f' and g from χ.

Step 2: Compute $f = 2f' + 1 \in R_q$.

Step 3: Compute $h = 2gf^{-1} \in R_q$.

After the above three steps, the public key h and the corresponding private key f can be obtained.

Encryption

The encryption process is given as follows.

Step 1: Sample polynomials s and e from χ.

Step 2: Compute $c = hs + 2e + m \in R_q$.

After the above two steps, polynomial c is generated as the ciphertext.

Decryption

The decryption process is proposed as follows.

Step 1: Compute $\mu = fc \in R_q$.

Step 2: Compute $m = \mu \mod 2$.

After the completion of the two steps, m is output as the decrypted message.

Homomorphic Properties

Let c_1 and c_2 be two ciphertexts, which are the encryptions of messages m_1 and m_2 with the same secret key f.

Addition:

Define the sum of two ciphertexts as

$$c_{add} = c_1 + c_2 \tag{9}$$

Then, we can have

$$\begin{aligned} f \cdot c_{add} \mod 2 &= f \cdot (c_1 + c_2) \mod 2 \\ &= f \cdot c_1 + f \cdot c_2 \mod 2 \\ &= \mu_1 + \mu_2 \mod 2 \\ &= m_1 + m_2 \mod 2 \\ &= m_1 + m_2 \end{aligned} \tag{10}$$

Hence, we can decrypt the sum of the two ciphertexts to recover the sum of the two messages.

Multiplication:

Define the product of two ciphertext as

$$c_{mult} = c_1 \cdot c_2 \tag{11}$$

Then, we can have

$$\begin{aligned} f^2 \cdot c_{mult} \mod 2 &= f^2 \cdot (c_1 \cdot c_2) \mod 2 \\ &= (f \cdot c_1) \cdot (f \cdot c_2) \mod 2 \\ &= \mu_1 \cdot \mu_2 \mod 2 \\ &= m_1 \cdot m_2 \mod 2 \\ &= m_1 \cdot m_2 \end{aligned} \tag{12}$$

Hence, we can decrypt the product of the two ciphertexts to recover the product of the two messages.

Combination of Addition and Multiplication

As mentioned before, since

$$f = 2f' + 1$$

We can have

$$f \equiv 1 \mod 2$$

Thus,

$$f^k \cdot m \mod 2 = m$$

Therefore, we can decrypt a combination of additions and multiplications through multiplying the result ciphertext by f^k, where f is the secret key and k is the depth of the longest chain of multiplications. This is one of the most significant improvements from the NTRUEncrypt scheme to the LTV scheme, which makes the LTV scheme somewhat homomorphic.

However, with the increasing of additions and multiplications, noise grows in result ciphertexts. Specifically, noise grows linearly for addition and exponentially for multiplication. The noise makes the result ciphertext indecipherable ultimately, which is a limitation for somewhat homomorphic encryption schemes.

4.3.2 Fully Homomorphic Encryption Scheme

In this section, we introduce how to convert the somewhat homomorphic LTV encryption scheme to a fully homomorphic one. Two techniques, relinearization and modulus reduction, are applied to the fully homomorphic LTV scheme.

Relinearization

As mentioned before, in order to decrypt a product of k ciphertexts, we are supposed to multiply the product ciphertext by the secret key k times. Thus, we have to keep track of the depth of the circuit. It turns out to be difficult when the circuit is complex.

Relinearization makes it possible for us to decrypt any combination of additions and multiplications of the ciphertext by multiplying the result ciphertext by the secret key only once. In order to achieve this, an evaluation key is used, which is introduced in Key Generation in Section 4.3.2.

Modulus Reduction

Modulus reduction is a kind of noise management technique which provides an exponential gain on the depth of the circuit that can be evaluated. It controls the noise by scaling the ciphertext after each operation. Applying it repeatedly can significantly improve the maximum depth of the circuit. It is discussed in Evaluation in Section 4.3.2 in detail.

Parameter Selection

The parameter selection for the fully homomorphic LTV scheme is the same as that for the somewhat version, which is referred to Parameter Selection in Section 4.3.1.

Key Generation

The key generation process is shown as follows.

Step 1: Choose a decreasing sequence of moduli $q_0 > q_1 > ... > q_{d_{dec}}$, where d_{dec} is the depth of the circuit to be evaluated.

Step 2: For every $i \in \{0, ..., d_{dec}\}$, sample $g^{(i)}$ and $u^{(i)}$ from χ.

Step 3: Compute $f^{(i)} = 2u^{(i)} + 1 \in R_{q_i}$.

Step 4: Compute $h^{(i)} = 2g^{(i)}(f^{(i)})^{-1} \in R_{q_i}$.

Step 5: For all $i \in [d_{dec}] = \{1, 2, ..., d_{dec}\}$ and $\tau \in \{0, ..., \lfloor \log q_i \rfloor\}$, sample $s_\tau^{(i)}$ and $e_\tau^{(i)}$ from χ.

Step 6: Compute $\gamma_\tau^{(i)} = h^{(i)}s_\tau^{(i)} + 2e_\tau^{(i)} + 2^\tau f^{(i-1)} \in R_{q_{i-1}}$ and $\zeta_\tau^{(i)} = h^{(i)}s_\tau^{(i)} + 2e_\tau^{(i)} + 2^\tau (f^{(i-1)})^2 \in R_{q_{i-1}}$.

After the above six steps, the public key is $h^{(0)} \in R_{q_0}$, the private key is $f^{(d_{dec})} \in R_{q_{d_{dec}}}$ and the evaluation key is $\left\{\gamma_\tau^{(i)}, \zeta_\tau^{(i)}\right\}_{i\in[d_{dec}],\tau\in\{0,\ldots,\lfloor\log q_i\rfloor\}}$.

Encryption

The encryption process is given as follows.

Step 1: Sample polynomials s and e from χ.

Step 2: Compute $c = h^{(0)}s + 2e + m \in R_{q_0}$.

After the above two steps, polynomial c is generated as the ciphertext.

Decryption

Given a ciphertext $c \in R_{q_{d_{dec}}}$, the decryption process is proposed as follows.

Step 1: Compute $\mu = f^{(d_{dec})}c \in R_{q_{d_{dec}}}$.

Step 2: Compute $m = \mu \mod 2$.

After the completion of the two steps, m is output as the decrypted message.

Evaluation

We show how to evaluate a t-input circuit C, where all inputs c_i are encrypted with the same public key/private key pair (pk, sk) having corresponding evaluation key ek, which is also publicly available.
Addition:
Given two ciphertexts $c_1, c_2 \in R_{q_i}$

Step 1: Compute $c = c_1 + c_2$.

Step 2: For $\tau \in \{0, ..., \lfloor \log q_i \rfloor\}$, define $\tilde{c}_\tau$ as $c = \sum_{\tau=0}^{\lfloor \log q_i \rfloor} 2^\tau \tilde{c}_\tau$.

Step 3: Define $\tilde{c} = \sum_{\tau=0}^{\lfloor \log q_i \rfloor} \tilde{c}_\tau \gamma_\tau^{(i)} \in R_{q_i}$.

Step 4: Compute $c_{add} = \left[\left(\frac{q_{i+1}}{q_i} \right) \cdot \tilde{c} \right] \mod 2$.

The output c_{add} is the encryption of the sum of the corresponding messages.

Multiplication:

Given two ciphertexts $c_1, c_2 \in R_{q_i}$

Step 1: Compute $c = c_1 \cdot c_2$.

Step 2: For $\tau \in \{0, ..., \lfloor \log q_i \rfloor\}$, define $\tilde{c}_\tau$ as $c = \sum_{\tau=0}^{\lfloor \log q_i \rfloor} 2^\tau \tilde{c}_\tau$.

Step 3: Define $\tilde{c} = \sum_{\tau=0}^{\lfloor \log q_i \rfloor} \tilde{c}_\tau \zeta_\tau^{(i+1)} \in R_{q_i}$.

Step 4: Compute $c_{mult} = \left[\left(\frac{q_{i+1}}{q_i} \right) \cdot \tilde{c} \right] \mod 2$.

The output c_{mult} is the encryption of the product of the corresponding messages.

5 THREE PROPOSED MULTIPLIERS FOR NTRU-ENCRYPT

5.1 Proposed Multiplier I

In this section, a new truncated polynomial ring multiplication architecture is proposed for NTRUEncrypt cryptosystem. Its FPGA simulation results are obtained. Firstly, basic concepts about linear feedback shift register (LFSR) are briefly introduced. Secondly, the existing similar works are reviewed in detail. Then, we propose a new x^2-net architecture to speed up the system. Next, an optimized design for arithmetic unit in the x^2-net architecture is presented. After that, a high speed multiplier architecture is proposed, which is called Multiplier I. Finally, the FPGA implementation results and a comparison of the proposed work with several existing works are given.

5.1.1 Linear Feedback Shift Register

Basically, an LFSR is a shift register whose input bit is a linear function of its previous state. The initial value of an LFSR is called a seed. Since the operation of the register is deterministic, the stream of values produced by the register is determined by its previous state. An LFSR can generate periodic sequence if it starts with a non-zero state and the maximal-length of an LFSR is $2^n - 1$, where n is the highest degree of the characteristic polynomial. LFSR has a wide range of applications, including generation of pseudo-random numbers, fast digital counters and cryptography.

An LFSR structure is given in Fig.5.1. It is determined by a characteristic polynomial $f(x)$, which can be expressed as

$$f(x) = x^n + f_{n-1}x^{n-1} + ... + f_1 x + 1 \tag{13}$$

In Fig.5.1, $\oplus$ refers to an adder and $\otimes$ refers to a multiplier. Besides, $\square$ represents a register. If we suppose that the registers are loaded with the coefficients of a polynomial $a(x) = (a_{n-1}, ..., a_0)$, a shift-to-right operation equals to perform $a(x) \times x \mod f(x)$, where x is the root of its characteristic polynomial $f(x)$.

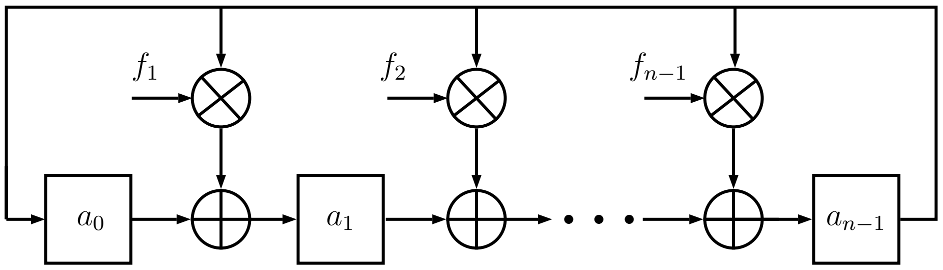


Fig. 5.1: Linear Feedback Shift Register

5.1.2 Truncated Polynomial Ring Multiplier

A truncated polynomial ring multiplier was proposed in [15]. Some modification is made to the LFSR structure to realize this function.

Assume that a register content e_k $(k = 0, ..., N-1)$ at clock cycle j is denoted by $e_k^{(j)}$. The detailed steps to implement truncated polynomial multiplication are given in Algorithm 5.1. In this algorithm, two inputs are polynomial $h(x)$ and $r(x)$, each with N coefficients and the output is the result polynomial $e(x)$ with N coefficients.

Algorithm 5.1 Multiplication in Truncated Polynomial Ring

```
Input: $h = h_{N-1}, ..., h_0$, $r = r_{N-1}, ..., r_0$
Output: $e = hr = e_{N-1}, ..., e_0$
1: $e^{(0)} := 0$
2: for $j := 1$ to $N$ do
3:     for $i := 0$ to $N-1$ do
4:         $e_i^{(j)} := e_{i+1 \bmod N}^{(j-1)} + h_{i+1 \bmod N} \times r_{j-1} \mod q$
5:     end for
6: end for
7: return $e := e^{(N)}$
```

An architecture for this multiplier is shown in Fig.5.2. In this multiplier, N adders, N multipliers and N registers are required. Note that the parameter q is in the form of 2^n, modular q operation can be easily achieved by truncating the result to n bits and all the operations including addition and multiplication are performed without any carry. The registers $e = (e_{N-1}, ..., e_0)$ are initially loaded with 0. The coefficients of the polynomial $h(x)$ are input to each multiplier in parallel while the coefficients of the polynomial $r(x)$ are input to all multipliers in a serial fashion. After N clock cycles, the multiplication result of $h(x) \times r(x)$ will be stored in the registers $e = (e_{N-1}, ..., e_0)$.

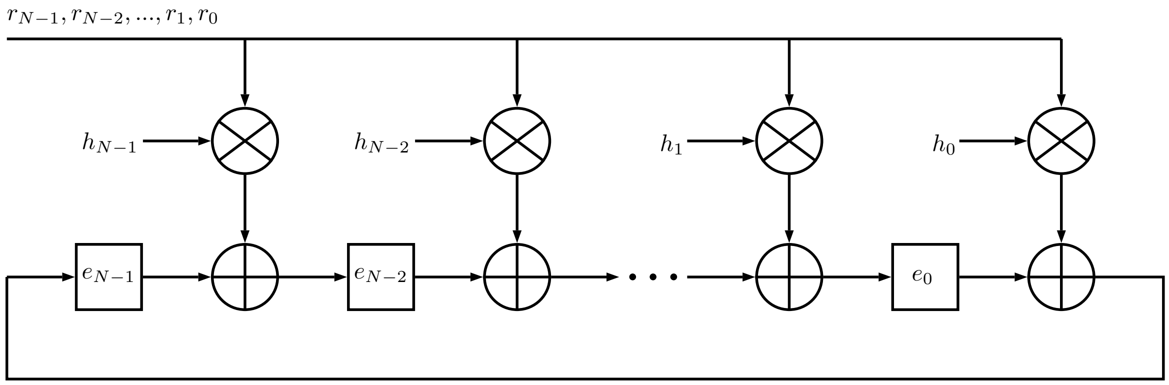


Fig. 5.2: Truncated Polynomial Ring Multiplier

5.1.3 Proposed x^2-net Architecture

The core architecture in the truncated polynomial ring multiplier is called x-net architecture, which is presented in Fig.5.3. In x-net architecture, a register content $e_k\,(k = 0, ..., N-1)$ at clock cycle j is denoted by $e_k^{(j)}$. $(h_{N-1}, ..., h_0)$ are the corresponding coefficients of the polynomial $h(x)$ and r_i stands for the ith coefficient of the polynomial $r(x)$, which acts as a control input at clock cycle $i + 1$. We treat one adder and one multiplier as an arithmetic unit and totally there are N arithmetic units required. In each arithmetic unit, there are three inputs and one output.

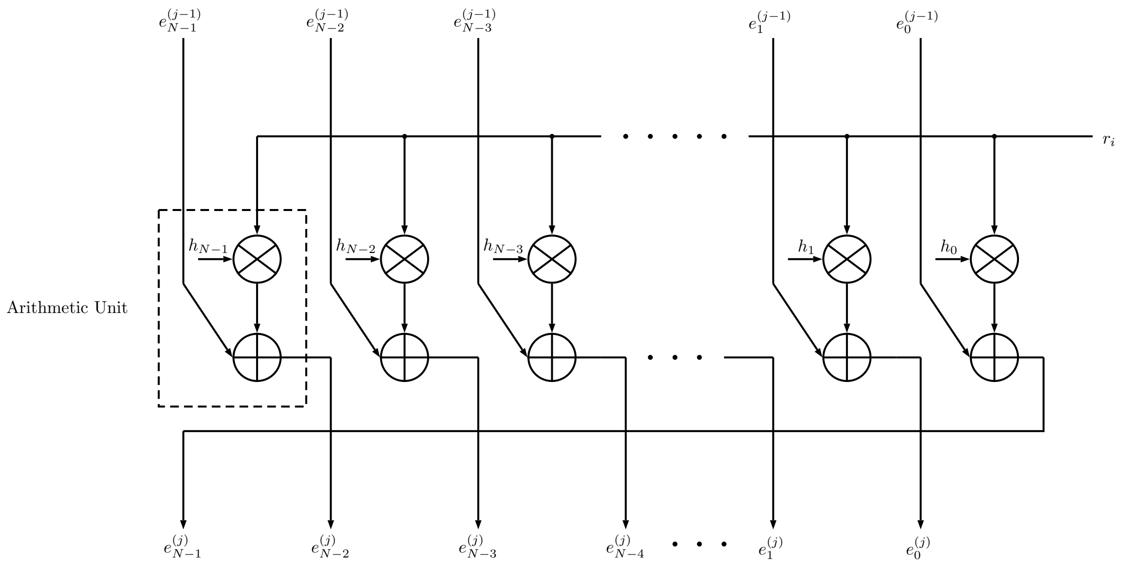


Fig. 5.3: x-net Architecture

In order to optimize x-net architecture, we propose a new design called x^2-net architecture, which is given in Fig.5.4. In x^2-net architecture, a register content $e_k\,(k = 0, .., N-1)$ at clock cycle j is denoted by $e_k^{(j)}$ and $(h_{N-1}, ..., h_0)$ are the corresponding coefficients of the polynomial $h(x)$. One difference is that this new design keeps the input r_i but also requires a new input r_{i-1}, which is the $(i-1)$th coefficient of the polynomial $r(x)$. Moreover, the size of one arithmetic unit is doubled that two adders and two multipliers are required. However, the total number of arithmetic units remains N. In each arithmetic unit, totally five inputs are required, two more than those in x-net architecture while the output is kept the same. Another modification is the shift-to-right operation. x-net architecture multiplies with x in one shift-to-right operation while the x^2-net architecture multiplies with x^2 in one shift-to-right operation. Two coefficients of the polynomial $r(x)$ can be processed during one clock cycle, thus, about half clock cycles can be saved.

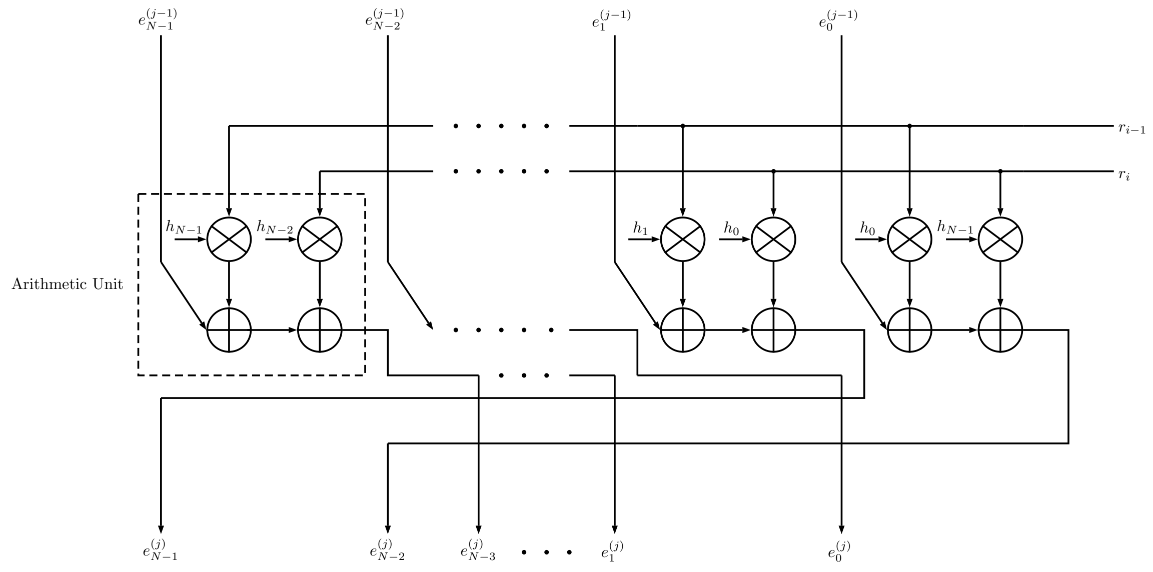


Fig. 5.4: x^2-net Architecture

5.1.4 Proposed Arithmetic Unit

In the encryption, since the coefficients of the polynomial $r(x)$ are selected from the set $\{-1, 0, 1\}$, the multiplication $h(x) \times r(x)$ can be evaluated without any integer multiplication. More specific, these multiplication operations can be replaced with only addition and subtraction operations. Meanwhile, as mentioned before, since the parameter q is in the form of 2^n, modular q operation can be achieved by truncating the result to n bits. An optimized design for the arithmetic unit in our x^2-net architecture is proposed in Fig.5.5.

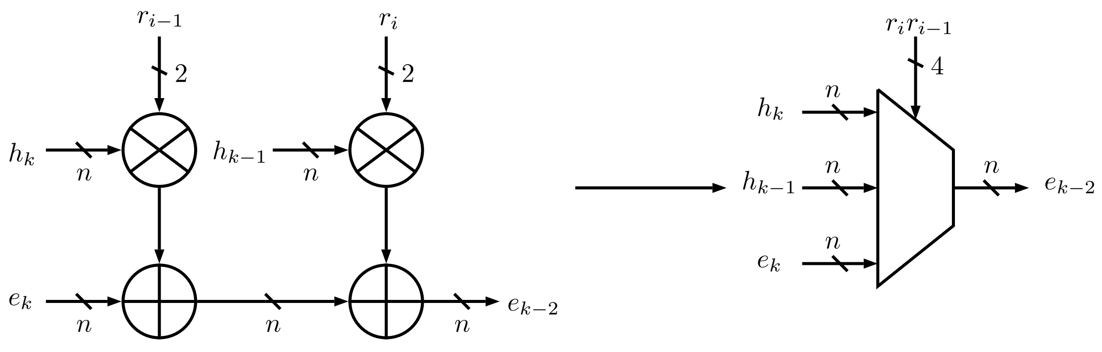


Fig. 5.5: Proposed Arithmetic Unit

In this new design of arithmetic unit, the number of inputs are reduced to four.

h_k, h_{k-1} and e_k are encoded in $n = [\log_2 q]$ bits. r_i and r_{i-1} are combined and a new input $r_i r_{i-1}$ is obtained, which is encoded in four bits and acts as the control input. Binary representation '11', '00' and '01' are used to represent the coefficients '-1', '0' and '1' in $r(x)$, respectively. The output e_{k-2} is also encoded in $n = [\log_2 q]$ bits. Table 5.1 shows the operations supported with this new arithmetic unit.

Table 5.1: Operations Supported with the Arithmetic Unit

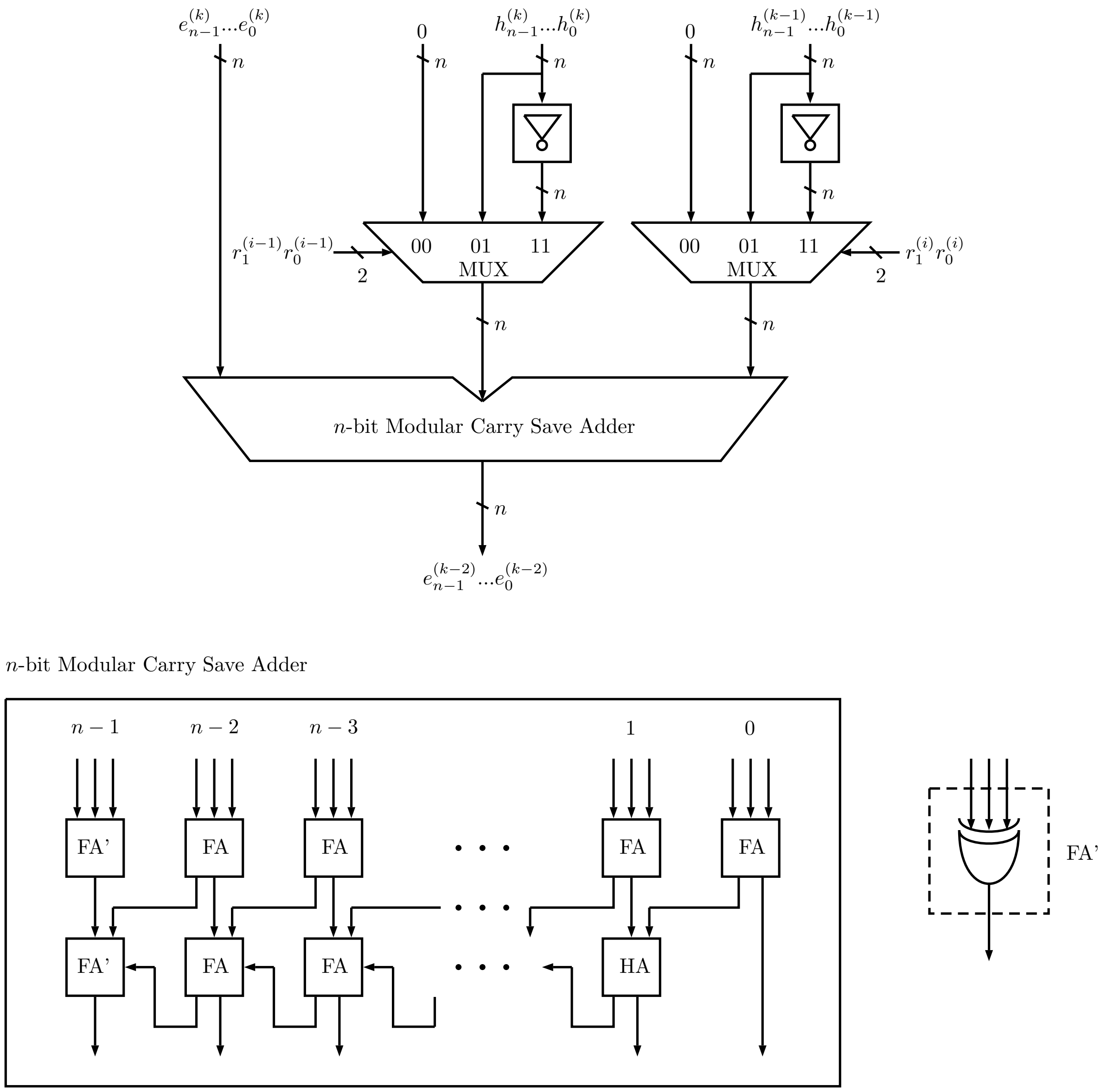
Input $r_i r_{i-1}$ $(r_1^{(i)} r_0^{(i)} r_1^{(i-1)} r_0^{(i-1)})$	Output e_{k-2}
0000	e_k
0001	$e_k + h_k \mod q$
0011	$e_k - h_k \mod q$
0100	$e_k + h_{k-1} \mod q$
0101	$e_k + h_k + h_{k-1} \mod q$
0111	$e_k - h_k + h_{k-1} \mod q$
1100	$e_k - h_{k-1} \mod q$
1101	$e_k + h_k - h_{k-1} \mod q$
1111	$e_k - h_k - h_{k-1} \mod q$

An algorithm that performs each step of this arithmetic unit is proposed in Algorithm 5.2.

Algorithm 5.2 Proposed Arithmetic Unit

```
Input: $e_k = (e_{n-1}^{(k)}...e_0^{(k)})_2$; $h_k = (h_{n-1}^{(k)}...h_0^{(k)})_2$; $h_{k-1} = (h_{n-1}^{(k-1)}...h_0^{(k-1)})_2$; $r_i r_{i-1} = (r_1^{(i)} r_0^{(i)} r_1^{(i-1)} r_0^{(i-1)})_2$;
Output: $e_{k-2} = (e_{n-1}^{(k-2)}...e_0^{(k-2)})_2$
 1: if $r_1^{(i-1)} r_0^{(i-1)} = 00$ then
 2:     $(h_{n-1}^{(k)}...h_0^{(k)}) := 0$
 3: else if $r_1^{(i-1)} r_0^{(i-1)} = 11$ then
 4:     $(h_{n-1}^{(k)}...h_0^{(k)}) := -(h_{n-1}^{(k)}...h_0^{(k)})$
 5: end if
 6: if $r_1^{(i)} r_0^{(i)} = 00$ then
 7:     $(h_{n-1}^{(k-1)}...h_0^{(k-1)}) := 0$
 8: else if $r_1^{(i)} r_0^{(i)} = 11$ then
 9:     $(h_{n-1}^{(k-1)}...h_0^{(k-1)}) := -(h_{n-1}^{(k-1)}...h_0^{(k-1)})$
10: end if
11: $(e_{n-1}^{(k-2)}...e_0^{(k-2)}) := (e_{n-1}^{(k)}...e_0^{(k)}) + (h_{n-1}^{(k)}...h_0^{(k)}) + (h_{n-1}^{(k-1)}...h_0^{(k-1)})$
```

The corresponding architecture for our proposed arithmetic unit is given in Fig.5.6.



$e^{(k)}_{n-1}...e^{(k)}_0$
0
$h^{(k)}_{n-1}...h^{(k)}_0$
0
$h^{(k-1)}_{n-1}...h^{(k-1)}_0$
n
$r^{(i-1)}_1 r^{(i-1)}_0$
2
00 01 11
MUX
$r^{(i)}_1 r^{(i)}_0$
n-bit Modular Carry Save Adder
$e^{(k-2)}_{n-1}...e^{(k-2)}_0$
n-bit Modular Carry Save Adder
$n-1$
$n-2$
$n-3$
1
0
FA'
FA
HA
FA'

Fig. 5.6: Proposed Arithmetic Unit Architecture

5.1.5 Proposed Multiplier I for Encryption

Based on the proposed arithmetic unit, a new multiplier for encryption in NTRU-Encrypt system can be proposed. In encryption, the ciphertext is computed by $e(x) = h(x) \times r(x) + m(x) \mod q$, where $h(x)$ is the public key, $r(x)$ is a randomly chosen polynomial whose coefficients are selected from the set $\{-1, 0, 1\}$ and $m(x)$ is the message polynomial.

A detailed algorithm for Multiplier I based encryption is shown in Algorithm 5.3. The inputs are three polynomials $h(x)$, $r(x)$ and $m(x)$, each with N coefficients and the output is an encrypted polynomial $e(x)$ with N coefficients. The corresponding structure is given in Fig.5.7.

Algorithm 5.3 Encryption in NTRUEncrypt

```
Input: $m = m_{N-1}, ..., m_0$; $h = h_{N-1}, ..., h_0$; $r = r_{N-1}, ..., r_0$;
Output: $e = e_{N-1}, ..., e_0 = hr + m \mod q$;
1: $e^{(0)} := m$
2: for $j := 1$ to $\left(\frac{N+1}{2}\right)$ do
3:   for $i := 0$ to $N-1$ do
4:     if $r_{2j-1} = 00$ then
5:       if $r_{2j-2} = 00$ then
6:         $e_i^{(j)} := e_{i+2 \mod N}^{(j-1)}$
7:       else if $r_{2j-2} = 01$ then
8:         $e_i^{(j)} := e_{i+2 \mod N}^{(j-1)} + h_{i+2 \mod N} \mod q$
9:       else
10:        $e_i^{(j)} := e_{i+2 \mod N}^{(j-1)} - h_{i+2 \mod N} \mod q$
11:      end if
12:    else if $r_{2j-1} = 01$ then
13:      if $r_{2j-2} = 00$ then
14:        $e_i^{(j)} := e_{i+2 \mod N}^{(j-1)} + h_{i+1 \mod N} \mod q$
15:      else if $r_{2j-2} = 01$ then
16:        $e_i^{(j)} := e_{i+2 \mod N}^{(j-1)} + h_{i+2 \mod N} + h_{i+1 \mod N} \mod q$
17:      else
18:        $e_i^{(j)} := e_{i+2 \mod N}^{(j-1)} - h_{i+2 \mod N} + h_{i+1 \mod N} \mod q$
19:      end if
20:    else
21:      if $r_{2j-2} = 00$ then
22:        $e_i^{(j)} := e_{i+2 \mod N}^{(j-1)} - h_{i+1 \mod N} \mod q$
23:      else if $r_{2j-2} = 01$ then
24:        $e_i^{(j)} := e_{i+2 \mod N}^{(j-1)} + h_{i+2 \mod N} - h_{i+1 \mod N} \mod q$
25:      else
26:        $e_i^{(j)} := e_{i+2 \mod N}^{(j-1)} - h_{i+2 \mod N} - h_{i+1 \mod N} \mod q$
27:      end if
28:    end if
29:  end for
30: end for
31: return $e := e^{\left(\frac{N+1}{2}\right)}$
```

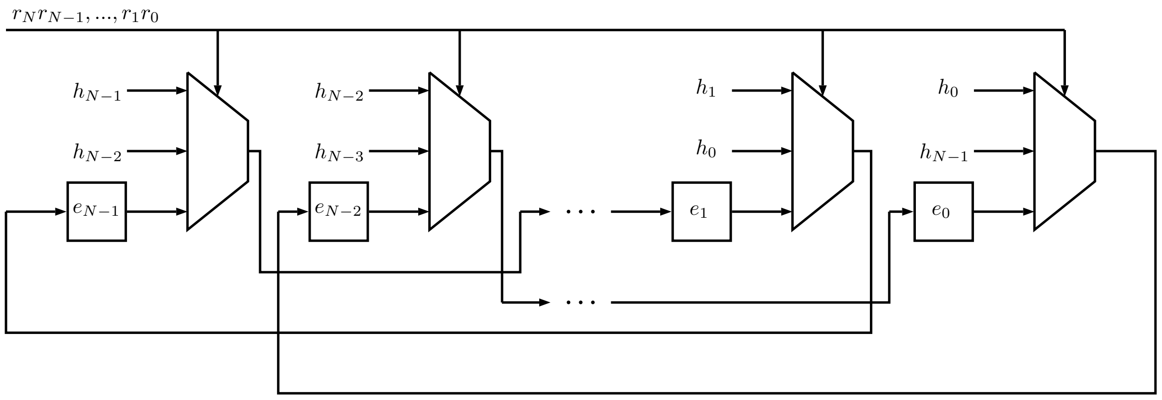


Fig. 5.7: Multiplier I Based NTRUEncrypt

This architecture contains N registers and N arithmetic units in total. The registers $e = (e_{N-1}, ..., e_0)$ are initially loaded with $m = (m_{N-1}, ..., m_0)$. When the operation starts, two consecutive coefficients of the polynomial $r(x)$ are scanned during one clock cycle. Since there are total N coefficients in the polynomial $r(x)$, we assume $r_N = 0$ and encode it as '00'. After $\frac{N+1}{2}$ clock cycles, the encryption result will be stored in the registers $e = (e_{N-2}, ..., e_0, e_{N-1})$.

5.1.6 Implementation of Multiplier I

FPGA is used to implement the proposed Multiplier I architecture. Basically, FPGA is an integrated circuit designed to be configured by a customer or a designer after manufacturing. It consists of an array of programmable logic blocks, which can be configured to perform either simple logic gates or complex combinational functions. In addition, it contains reconfigurable interconnects that allow the blocks to be wired together.

FPGA is widely used in different fields, such as digital signal processing and cryptography. It can also be used to solve computable problems, as it is much faster in some applications for their parallel nature and optimality in terms of the number of gates used for a certain process. Meanwhile, another significant usage of FPGA is hardware acceleration, especially the acceleration on certain parts of an algorithm.

In order to define the behavior of FPGA, a hardware description language is provided to specify the FPGA configuration. The most common used hardware description language is VHDL and Verilog HDL. In our implementation, we use Verilog HDL as our design language.

In specific, the following tools are used for our implementation.

- Quartus II v14.1 (64-bit) Software
- ModelSim-Altera Software

Our target device is Arria V 5AGXFB3H4F35I3. It belongs to Arria V FPGA family that offer the highest bandwidth and deliver the lowest total power for midrange applications.

Implementation Results

The simulation results are shown in Table 5.2.

Table 5.2: Simulation Results for Different Parameter Sets

Security Level	Parameter set	#ALM	#Register	#Cycles	FMax	Latency
112	$ees401ep1$	11,861	8,826	201	103.01 MHz	1.95 μs
	$ees541ep1$	16,091	11,906	271	102.33 MHz	2.65 μs
	$ees659ep1$	19,573	14,502	330	105.96 MHz	3.11 μs
128	$ees449ep1$	13,296	9,882	225	105.24 MHz	2.14 μs
	$ees613ep1$	18,201	13,490	307	104.60 MHz	2.93 μs
	$ees761ep1$	22,459	16,746	381	103.33 MHz	3.69 μs
192	$ees677ep1$	20,042	14,898	339	106.90 MHz	3.17 μs
	$ees887ep1$	26,295	19,518	444	103.68 MHz	4.28 μs
	$ees1087ep1$	32,241	23,918	544	96.43 MHz	5.64 μs
256	$ees1087ep2$	32,241	23,918	544	96.43 MHz	5.64 μs
	$ees1171ep1$	34,648	25,766	586	99.50 MHz	5.89 μs
	$ees1499ep1$	48,719	32,982	750	82.73 MHz	9.07 μs

For each parameter set, we mainly focus on five aspects of our proposed architecture.

- #ALM, number of adaptive logic modules.
- #Register, number of registers.
- #Cycles, number of cycles.
- FMax, maximum operating frequency.
- Latency, required encryption time.

ALM is the basic building block of Arria V. In our simulation, the maximal number of ALMs on our target device Arria V 5AGXFB3H4F35I3 is 136880. Number of registers indicates how many registers are required for the architecture. Area consumption of an architecture is denoted by the combination of number of ALMs and number of registers. Number of cycles shows how many clock cycles are required for the computation. FMax means the maximum frequency that can be achieved during the operation. It usually depends on the design as well as the chip. Latency reflects the time required for the encryption. It is calculated by number of cycles divided by FMax.

Comparison

For each security level, we choose one parameter set for comparison.

The comparison results for security level 112 are shown in Table 5.3.

Table 5.3: Security Level 112

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees401ep1$	4,052	9,638	401	62.95 MHz	6.37 μs
[11]	$ees401ep1$	837	1,165	3,617	67.69 MHz	53.43 μs
[12]	$ees401ep1$	15,662	8,838	227	55.00 MHz	4.13 μs
[15]	$ees401ep1$	4,636	8,826	401	121.62 MHz	3.30 μs
[16]	$ees401ep1$	9,044	8,826	349	113.67 MHz	3.07 μs
Multiplier I	$ees401ep1$	11,861	8,826	201	103.01 MHz	1.95 μs

It can be seen from the table that number of cycles of Multiplier I is the least, compared to all the existing works. In addition, Multiplier I has the highest speed. Latency of Multiplier I is only 63.52%, compared to the fastest among the existing similar works.

The comparison results for security level 128 are shown in Table 5.4.

Table 5.4: Security Level 128

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees449ep1$	4,523	10,793	449	56.99 MHz	7.88 μs
[11]	$ees449ep1$	837	1,165	4,049	67.69 MHz	59.82 μs
[12]	$ees449ep1$	17,527	9,884	269	53.95 MHz	4.99 μs
[15]	$ees449ep1$	5,188	9,882	449	121.69 MHz	3.69 μs
[16]	$ees449ep1$	10,124	9,882	398	112.90 MHz	3.53 μs
Multiplier I	$ees449ep1$	13,296	9,882	225	105.24 MHz	2.14 μs

It can be seen from the table that number of cycles of Multiplier I is the least, compared to all the existing works. Additionally, Multiplier I has the highest speed. Latency of Multiplier I is only 60.62%, compared to the fastest among the existing similar works.

The comparison results for security level 192 are shown in Table 5.5.

Table 5.5: Security Level 192

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees677ep1$	6,740	16,266	677	60.24 MHz	11.24 μs
[11]	$ees677ep1$	837	1,165	9,486	67.69 MHz	140.14 μs
[12]	$ees677ep1$	26,423	14,900	317	49.74 MHz	6.37 μs
[15]	$ees677ep1$	7,810	14,898	677	120.00 MHz	5.64 μs
[16]	$ees677ep1$	15,254	14,898	551	106.30 MHz	5.18 μs
Multiplier I	$ees677ep1$	20,042	14,898	339	106.90 MHz	3.17 μs

It can be seen from the table that Multiplier I has the highest speed. Latency of Multiplier I is only 61.20%, compared to the fastest among the existing similar works.

The comparison results for security level 256 are shown in Table 5.6.

Table 5.6: Security Level 256

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees1087ep2$	10,748	26,105	1,087	55.53 MHz	19.58 μs
[11]	$ees1087ep2$	837	1,165	23,922	67.69 MHz	353.41 μs
[12]	$ees1087ep2$	42,427	23,930	276	45.75 MHz	6.03 μs
[15]	$ees1087ep2$	12,526	23,918	1,087	104.06 MHz	10.45 μs
[16]	$ees1087ep2$	24,480	23,918	717	94.07 MHz	7.62 μs
Multiplier I	$ees1087ep2$	32,241	23,918	544	96.43 MHz	5.64 μs

It can be seen from the table that Multiplier I has the highest speed. Latency of Multiplier I is only 93.53%, compared to the fastest among the existing similar works.

5.2 Proposed Multiplier II

In this section, a truncated polynomial ring multiplication architecture is proposed for NTRUEncrypt. Its FPGA simulation results are obtained. At first, we introduce some basic idea for our proposed design. Next, we propose a new arithmetic unit. After that, a multiplier architecture is proposed, which is called Multiplier II. Finally, the FPGA implementation results and a comparison of the proposed work with several existing works are given.

5.2.1 General Idea

In LFSR based architecture proposed in [15], the control input r_i is encoded in two bits, thus, it has three states, '11', '00' and '01'. Thus, the state '10' is considered as a redundant state in such design. The architecture is supposed to be more efficient if we can make use of this redundant state.

According to the parameter selection of NTRUEncrypt system, as $r(x) \in L(d_r, d_r)$ and d_r is much smaller than the parameter N, there are a large number of coefficients equal to 0 in $r(x)$.

Basically, our proposed architecture is based on the idea that a considerable reduction in number of cycles can be achieved if ‘0, 0, 0’ coefficient pairs in $r(x)$ can be processed during one clock cycle. Therefore, we are required to find out how many ‘0, 0, 0’ coefficient pairs appear consecutively in $r(x)$.

The number of ‘1’ and ‘-1’ coefficients in $r(x)$ is denoted by

$$n_1 = n_{-1} = d_r \tag{14}$$

So the number of ‘0’ coefficients in $r(x)$ can be calculated as

$$n_0 = N - 2d_r \tag{15}$$

The maximum number of ‘0, 0, 0’ coefficient pairs in $r(x)$ is calculated as

$$n_{000_{max}} = (N - 2d_r)/3. \tag{16}$$

The minimum number of ‘0, 0, 0’ coefficient pairs in $r(x)$ is calculated as

$$n_{000_{min}} = \begin{cases} 0 & n_1 + n_{-1} > \dfrac{n_0}{2} - 1 \\ (N - 6d_r)/3 & n_1 + n_{-1} \leq \dfrac{n_0}{2} - 1 \end{cases} \tag{17}$$

The average number of ‘0, 0, 0’ coefficient pairs in $r(x)$ can be calculated as

$$n_{000_{avg}} = \sum_{k=n_{000_{min}}}^{n_{000_{max}}} \left(k \cdot \frac{{}^{2m}C_{k+2m} \cdot \left(\sum_{r=0}^{\lfloor \frac{n-2m-3k}{3} \rfloor} (-1)^r \cdot {}^{r}C_{2m+1} \cdot {}^{n-2m-3k-3r}C_{n-3k-3r} \right)}{{}^{n-2m}C_{n}} \right) \tag{18}$$

Table 5.7 shows the number of ‘0, 0, 0’ coefficient pairs in different parameter sets. For each parameter set, we first calculate the total number of ‘0’ coefficients which is listed in the column # '0'. In the column # '0, 0, 0', three sub-columns are separated, where Min, Max and Avg represent the minimum number of '0, 0, 0' coefficient pairs, the maximum number of '0, 0, 0' coefficient pairs and the average number of '0, 0, 0' coefficient pairs in $r(x)$, respectively.

Table 5.7: Number of '0, 0, 0' Pairs in Different Parameter Sets

Security Level	Parameter set	N	#"0"	#'0, 0, 0'		
				Min	Max	Avg
112	$ees401ep1$	401	175	0	58	20.30
	$ees541ep1$	541	443	82	147	119.03
	$ees659ep1$	659	583	143	194	170.76
128	$ees449ep1$	449	181	0	60	18.62
	$ees613ep1$	613	503	94	167	135.51
	$ees761ep1$	761	677	169	225	199.54
192	$ees677ep1$	677	363	0	121	57.01
	$ees887ep1$	887	725	133	241	194.59
	$ees1087ep1$	1087	961	236	320	281.47
256	$ees1087ep2$	1087	847	122	282	215.22
	$ees1171ep1$	1171	959	178	319	258.05
	$ees1499ep1$	1499	1341	341	447	397.93

Hence, the average number of cycles required for different parameter sets can be calculated and the results are given in Table 5.8.

Table 5.8: Average Number of Cycles for Different Parameter Sets

Security Level	Parameter set	N	p	q	d_r	#Cycles
112	ees401ep1	401	3	2048	113	381
	ees541ep1	541	3	2048	49	422
	ees659ep1	659	3	2048	38	489
128	ees449ep1	449	3	2048	134	431
	ees613ep1	613	3	2048	55	478
	ees761ep1	761	3	2048	42	562
192	ees677ep1	677	3	2048	157	620
	ees887ep1	887	3	2048	81	693
	ees1087ep1	1087	3	2048	63	806
256	ees1087ep2	1087	3	2048	120	872
	ees1171ep1	1171	3	2048	106	913
	ees1499ep1	1499	3	2048	79	1102

5.2.2 Proposed Arithmetic Unit

In this new arithmetic unit, the inputs h_k and e_k are both encoded in $n = [\log_2 q]$ bits. Meanwhile, t_i is the control input which is encoded in two bits. The output e_{k-1} is also encoded in $n = [\log_2 q]$ bits. The main difference is that we add e_{k+2} as a new input in our arithmetic unit, which is encoded in $n = [\log_2 q]$ bits as well. The proposed arithmetic unit is given in Fig.5.8 and the operation table is shown in Table 5.9.

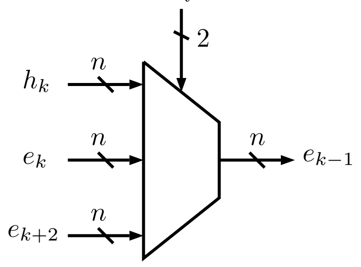


Fig. 5.8: Proposed Arithmetic Unit

Table 5.9: Operations Supported with the Arithmetic Unit

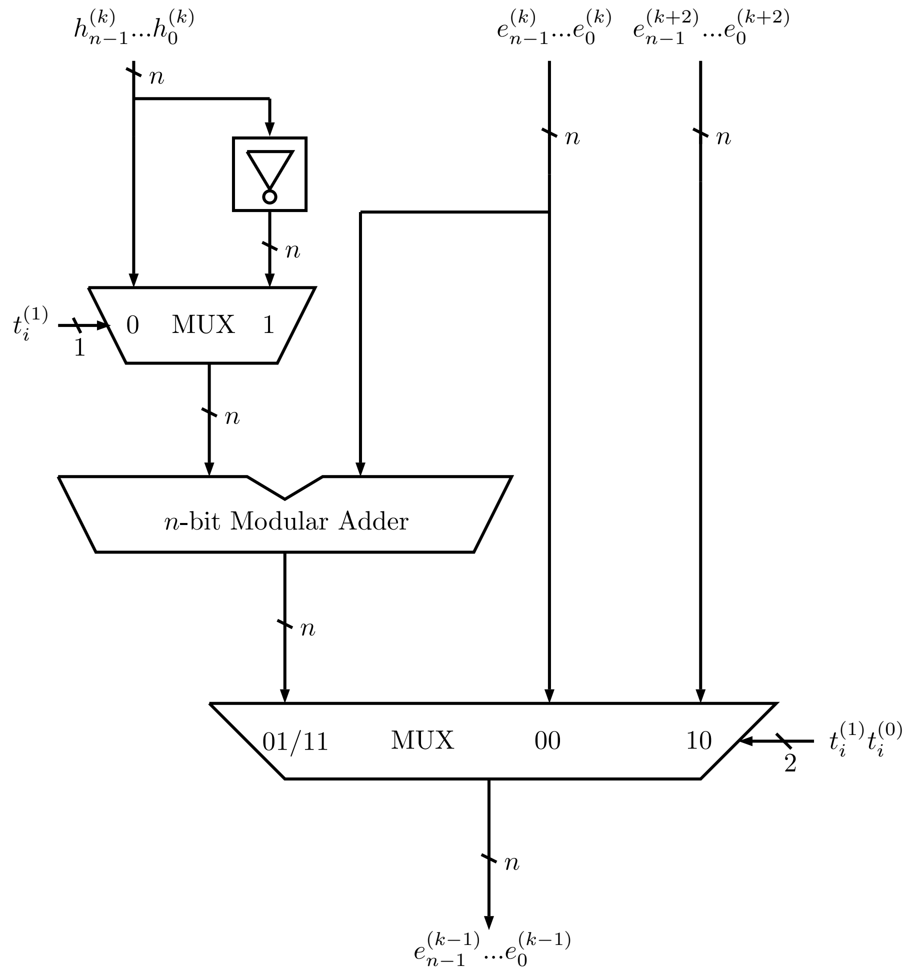
r_j, r_{j+1}, r_{j+2}	Input t_i ($t_1^{(i)} t_0^{(i)}$)	Output e_{k-1}
$0, \times, \times$	00	e_k
$1, \times, \times$	01	$e_k + h_k \mod q$
$0, 0, 0$	10	e_{k+2}
$-1, \times, \times$	11	$e_k - h_k \mod q$

An algorithm that performs each step of this arithmetic unit is shown in Algorithm 5.4.

Algorithm 5.4 Proposed Arithmetic Unit

```
Input: $e_k = (e^{(k)}_{n-1}...e^{(k)}_0)_2$; $e_{k+2} = (e^{(k+2)}_{n-1}...e^{(k+2)}_0)_2$; $h_k = (h^{(k)}_{n-1}...h^{(k)}_0)_2$; $t_i = (t^{(i)}_1 t^{(i)}_0)_2$;
Output: $e_{k-1} = (e^{(k-1)}_{n-1}...e^{(k-1)}_0)_2$;
1: if $t^{(i)}_1 t^{(i)}_0 = 00$ then
2:     $(e^{(k-1)}_{n-1}...e^{(k-1)}_0) := (e^{(k)}_{n-1}...e^{(k)}_0)$
3: else if $t^{(i)}_1 t^{(i)}_0 = 01$ then
4:     $(e^{(k-1)}_{n-1}...e^{(k-1)}_0) := (e^{(k)}_{n-1}...e^{(k)}_0) + (h^{(k)}_{n-1}...h^{(k)}_0)$
5: else if $t^{(i)}_1 t^{(i)}_0 = 10$ then
6:     $(e^{(k-1)}_{n-1}...e^{(k-1)}_0) := (e^{(k+2)}_{n-1}...e^{(k+2)}_0)$
7: else
8:     $(e^{(k-1)}_{n-1}...e^{(k-1)}_0) := (e^{(k)}_{n-1}...e^{(k)}_0) - (h^{(k)}_{n-1}...h^{(k)}_0)$
9: end if
```

The corresponding architecture for our proposed arithmetic unit is given in Fig.5.9.



$h^{(k)}_{n-1}...h^{(k)}_0$
$e^{(k)}_{n-1}...e^{(k)}_0$
$e^{(k+2)}_{n-1}...e^{(k+2)}_0$
n
$t^{(1)}_i$
1
0 MUX 1
n-bit Modular Adder
01/11 MUX 00 10
$t^{(1)}_i t^{(0)}_i$
2
$e^{(k-1)}_{n-1}...e^{(k-1)}_0$

Fig. 5.9: Proposed Arithmetic Unit Architecture

5.2.3 Proposed Multiplier II for Encryption

Based on the new arithmetic unit, we propose a multiplier architecture for NTRUEncrypt. Note that this new architecture can process three consecutive zero coefficients in $r(x)$ during one clock cycle, before the computation, we are supposed to encode $r(x) = (r_{N-1}, ..., r_0)$ to obtain a new control input sequence $(t_{u-1}, ..., t_0)$, where each t_i has two bits. If three consecutive zero coefficients occur in $r(x)$, we encode them as '10'. For the other coefficients '-1', '0' and '1', we encode them as '11', '00' and '01' correspondingly.

Algorithm 5.5 shows the encryption for our proposed architecture and the corresponding architecture is given in Fig.5.10. There are totally three inputs in this architecture, including the public key $h(x)$, the message polynomial $m(x)$ and the generated sequence $(t_{u-1}, ..., t_0)$. The output is an encrypted polynomial $e(x)$ with N coefficients.

Algorithm 5.5 Encryption in NTRUEncrypt

```
Input: $m = m_{N-1}, ..., m_0$; $h = h_{N-1}, ..., h_0$; $(t_{u-1}, ..., t_0)$;
Output: $e = e_{N-1}, ..., e_0 = hr + m \mod q$;
 1: $e^{(0)} := m$
 2: for $j := 1$ to $u$ do
 3:   for $i := 0$ to $N - 1$ do
 4:     if $t_{j-1} = 00$ then
 5:       $e_i^{(j)} := e_{i+1 \mod N}^{(j-1)}$
 6:     else if $t_{j-1} = 10$ then
 7:       $e_i^{(j)} := e_{i+3 \mod N}^{(j-1)}$
 8:     else if $t_{j-1} = 01$ then
 9:       $e_i^{(j)} := e_{i+1 \mod N}^{(j-1)} + h_{i+1 \mod N} \mod q$
10:     else
11:       $e_i^{(j)} := e_{i+1 \mod N}^{(j-1)} - h_{i+1 \mod N} \mod q$
12:     end if
13:   end for
14: end for
15: return $e := e^{(u)}$
```

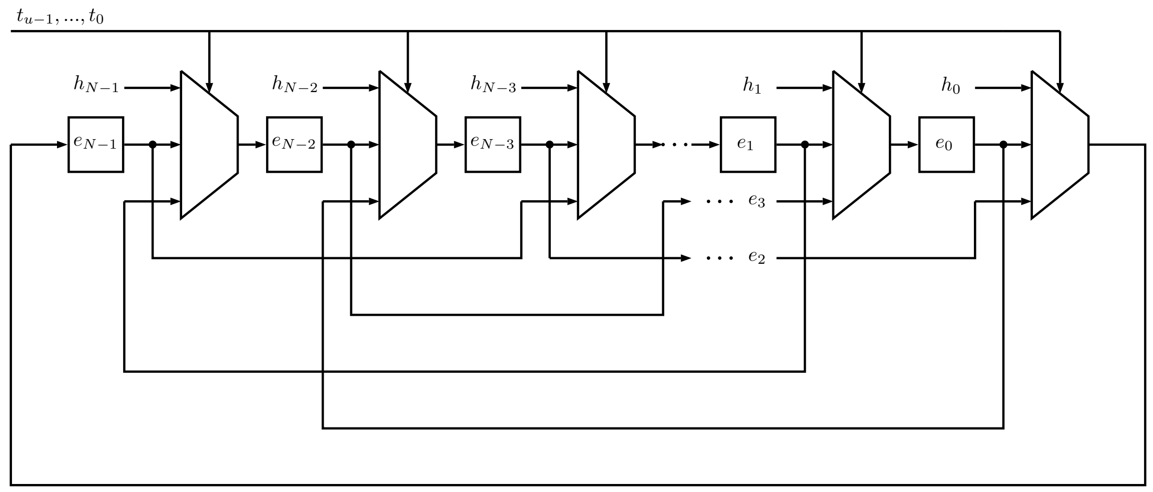


Fig. 5.10: Multiplier II Based NTRUEncrypt

This multiplier contains N registers and N arithmetic units in total. The registers $e = (e_{N-1}, ..., e_0)$ are initially loaded with $m = (m_{N-1}, ..., m_0)$. When the operation starts, each t_i is scanned during one clock cycle. After u clock cycles, the encryption result will be stored in the registers $e = (e_{N-1}, ..., e_0)$.

5.2.4 Implementation of Multiplier II

In our FPGA implementation, the following tools are used.

- Quartus II v14.1 (64-bit) Software
- ModelSim-Altera Software

Our target device is Arria V 5AGXFB3H4F35I3 and we use Verilog HDL as our design language.

Implementation Results

The implementation results are shown in Table 5.10.

Table 5.10: Simulation Results for Different Parameter Sets

Security Level	Parameter set	#ALM	#Register	#Cycles	FMax	Latency
112	$ees401ep1$	6,888	8,826	381	121.25 MHz	3.14 μs
	$ees541ep1$	9,256	11,906	422	121.76 MHz	3.47 μs
	$ees659ep1$	11,267	14,502	489	113.57 MHz	4.31 μs
128	$ees449ep1$	7,731	9,882	431	120.12 MHz	3.59 μs
	$ees613ep1$	10,469	13,490	478	113.09 MHz	4.23 μs
	$ees761ep1$	13,023	16,746	562	108.70 MHz	5.17 μs
192	$ees677ep1$	11,551	14,898	620	118.38 MHz	5.24 μs
	$ees887ep1$	15,163	19,518	693	102.30 MHz	6.77 μs
	$ees1087ep1$	18,613	23,918	806	103.01 MHz	7.82 μs
256	$ees1087ep2$	18,613	23,918	872	103.01 MHz	8.47 μs
	$ees1171ep1$	19,964	25,766	913	103.79 MHz	8.80 μs
	$ees1499ep1$	25,543	32,982	1,102	97.73 MHz	11.28 μs

Comparison

For each security level, we choose one parameter set for comparison.

The comparison results for security level 112 are shown in Table 5.11.

Table 5.11: Security Level 112

Work	Parameter Set	#ALM(S_1)	#Register(S_2)	Latency(T)	$(S_1 + S_2) \times T$
[10]	$ees401ep1$	4,052	9,638	6.37 μs	176.74%
[11]	$ees401ep1$	837	1,165	53.43 μs	216.79%
[12]	$ees401ep1$	15,662	8,838	4.13 μs	205.07%
[15]	$ees401ep1$	4,636	8,826	3.30 μs	90.03%
[16]	$ees401ep1$	9,044	8,826	3.07 μs	111.19%
Multiplier II	$ees401ep1$	6,888	8,826	3.14 μs	100%

It can be seen from the table that latency of Multiplier II is the second lowest in comparison. Although [16] performs better than Multiplier II in terms of latency, Multiplier II has better area-latency-product. Area-latency-product of Multiplier II is only 89.94%, compared to [16].

The comparison results for security level 128 are shown in Table 5.12.

Table 5.12: Security Level 128

Work	Parameter Set	#ALM(S_1)	#Register(S_2)	Latency(T)	$(S_1 + S_2) \times T$
[10]	$ees449ep1$	4,523	10,793	7.88 μs	190.87%
[11]	$ees449ep1$	837	1,165	59.82 μs	189.40%
[12]	$ees449ep1$	17,527	9,884	4.99 μs	216.32%
[15]	$ees449ep1$	5,188	9,882	3.69 μs	87.95%
[16]	$ees449ep1$	10,124	9,882	3.53 μs	111.69%
Multiplier II	$ees449ep1$	7,731	9,882	3.59 μs	100%

It can be seen from the table that latency of Multiplier II is the second lowest in comparison. Though [16] outperforms Multiplier II in terms of latency, Multiplier II has better area-latency-product. Area-latency-product of Multiplier II is only 89.53%, compared to [16].

The comparison results for security level 192 are shown in Table 5.13.

Table 5.13: Security Level 192

Work	Parameter Set	#ALM(S_1)	#Register(S_2)	Latency(T)	$(S_1 + S_2) \times T$
[10]	$ees677ep1$	6,740	16,266	11.24 μs	186.58%
[11]	$ees677ep1$	837	1,165	140.14 μs	202.44%
[12]	$ees677ep1$	26,423	14,900	6.37 μs	189.93%
[15]	$ees677ep1$	7,810	14,898	5.64 μs	92.41%
[16]	$ees677ep1$	15,254	14,898	5.18 μs	112.70%
Multiplier II	$ees677ep1$	11,551	14,898	5.24 μs	100%

It can be seen from the table that latency of Multiplier II is the second lowest in comparison. Although [16] performs better than Multiplier II in terms of latency, Multiplier II has better area-latency-product. Area-latency-product of Multiplier II is only 88.73%, compared to [16].

The comparison results for security level 256 are shown in Table 5.14.

Table 5.14: Security Level 256

Work	Parameter Set	#ALM(S_1)	#Register(S_2)	Latency(T)	$(S_1 + S_2) \times T$
[10]	$ees1087ep2$	10,748	26,105	19.58 μs	200.31%
[11]	$ees1087ep2$	837	1,165	353.41 μs	196.41%
[12]	$ees1087ep2$	42,427	23,930	6.03 μs	111.07%
[15]	$ees1087ep2$	12,526	23,918	10.45 μs	105.72%
[16]	$ees1087ep2$	24,480	23,918	7.62 μs	102.37%
Multiplier II	$ees1087ep2$	18,613	23,918	8.47 μs	100%

It can be seen from the table that latency of Multiplier II is the third lowest in comparison. Though [12] and [16] outperform Multiplier II in terms of latency, Multiplier II has better area-latency-product. Area-latency-product of Multiplier II is only 90.03% and 97.68%, compared to [12] and [16], respectively.

5.3 Proposed Multiplier III

In this section, a truncated polynomial ring multiplication architecture is proposed for NTRUEncrypt. Firstly, we introduce some basic idea for our proposed architecture. Then, we propose an arithmetic unit. Next, a multiplier architecture is proposed, which is called Multiplier III. Finally, the FPGA implementation results and a comparison of the proposed work with several existing works are given.

5.3.1 General Idea

In Multiplier II, we use two bits to encode the control input t_i and four encoding states of two bits are fully used.

In this new design, we extend the control input t_i from two bits to three bits, thus, we can obtain eight encoding states in total.

Basically, our proposed architecture is based on the idea that a considerable reduction in number of cycles can be achieved if different coefficient pairs in $r(x)$ can be processed during one clock cycle. Hence, we are required to find out how many coefficient pairs meet the certain requirement. Specifically, we are supposed to compute the number of cycles for different parameter sets.

The detailed encoding of t_i is shown in Table 5.16. In such encoding, r_j represents the jth coefficient in $r(x)$ and we are supposed to scan each coefficient in $r(x)$ when we encode t_i. Note that such encoding does not contain coefficients of one zero or two consecutive zeros in $r(x)$, we are required to calculate a phase-shift value during the last scan. There are totally three circumstances. If the last scan is one zero, no clock cycle will be utilized and the phase-shift value will be one. If the last scan is two consecutive zeros, no clock cycle will be utilized and the phase-shift value will be two. For the rest, number of cycles will be added by one and the phase-shift value will be zero. The phase-shift value will be used when the final result is read out and it will decide the location of the most significant coefficient of the encryption result polynomial.

According to Table 5.16, we can compute the number of cycles for each parameter set. Several steps are supposed to be followed. First, we randomly generate one hundred sets of polynomial $r(x)$ for a certain parameter set. Then, for each generated polynomial $r(x)$, we start to encode $r(x) = (r_{N-1}, ..., r_0)$ to obtain $(t_{u-1}, ..., t_0)$. Next, we count the number of t_i for each polynomial set. At last, we compute the average of the number of t_i, which is the number of cycles we are looking for.

Applying these steps to each parameter set, we can obtain the number of cycles for all parameter sets, which is given in Table 5.15.

Table 5.15: Average Number of Cycles for Different Parameter Sets

Security Level	Parameter set	N	p	q	d_r	#Cycles
112	ees401ep1	401	3	2048	113	246
	ees541ep1	541	3	2048	49	196
	ees659ep1	659	3	2048	38	213
128	ees449ep1	449	3	2048	134	286
	ees613ep1	613	3	2048	55	222
	ees761ep1	761	3	2048	42	243
192	ees677ep1	677	3	2048	157	367
	ees887ep1	887	3	2048	81	323
	ees1087ep1	1087	3	2048	63	350
256	ees1087ep2	1087	3	2048	120	420
	ees1171ep1	1171	3	2048	106	424
	ees1499ep1	1499	3	2048	79	473

5.3.2 Proposed Arithmetic Unit

In this new design of arithmetic unit, the inputs e_{k+1}, e_{k+2}, e_{k+3}, e_{k+4} and h_{k+1} are all encoded in $n = [\log_2 q]$ bits while the control input t_i is encoded in three bits. The output e_k is also encoded in $n = [\log_2 q]$ bits. The proposed arithmetic unit is shown in Fig.5.11 and the operation table is given in Table 5.16. There is no redundant state in such design, as eight states of three bits are fully used.

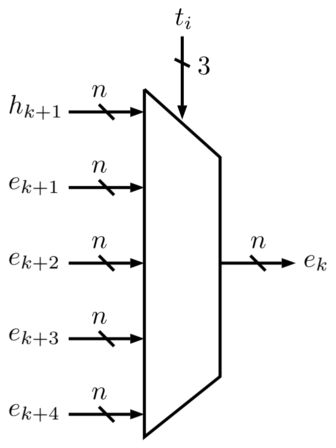


Fig. 5.11: Proposed Arithmetic Unit

Table 5.16: Operations Supported with the Arithmetic Unit

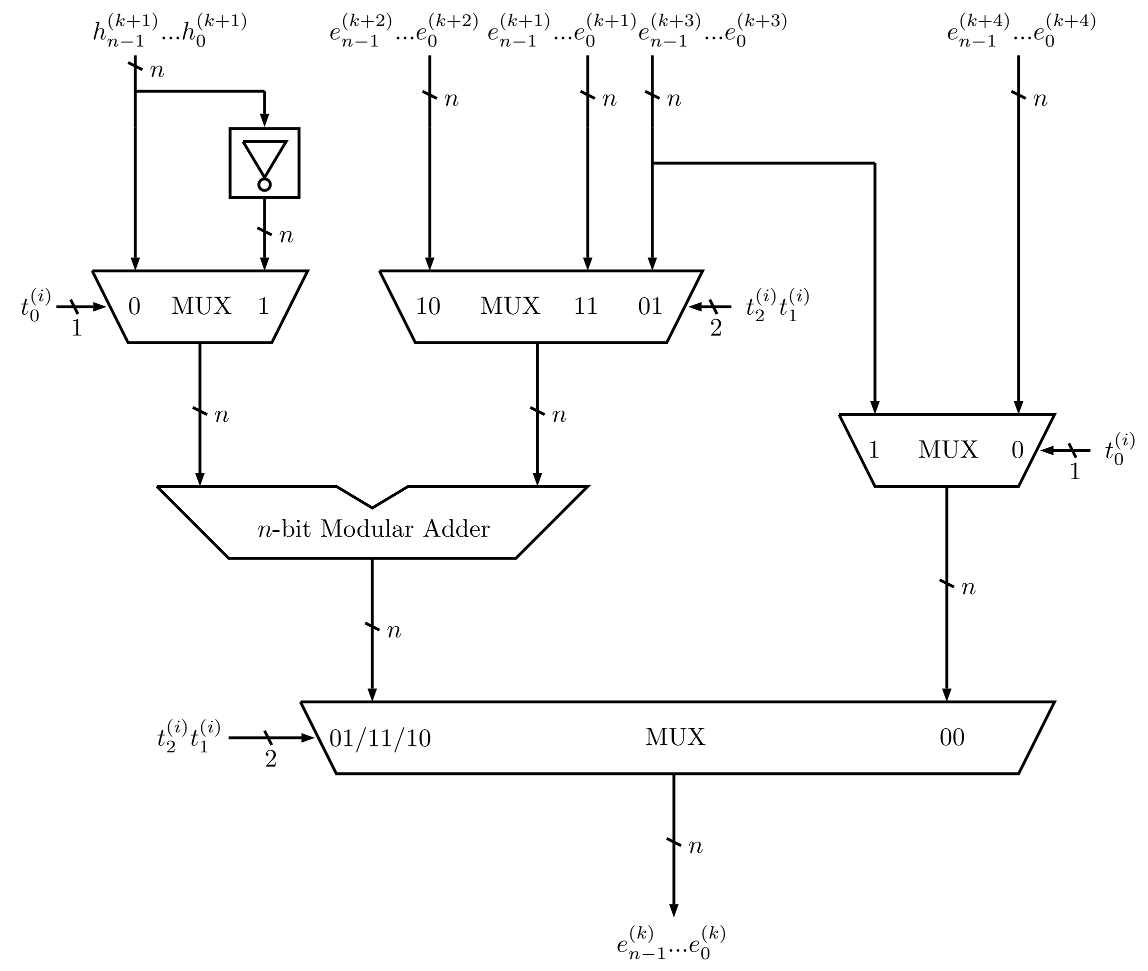
$r_j, r_{j+1}, r_{j+2}, r_{j+3}$	Input t_i $(t_2^{(i)} t_1^{(i)} t_0^{(i)})$	Output e_k
$0, 0, 0, 0$	000	e_{k+4}
$0, 0, 0, \pm 1$	001	e_{k+3}
$0, 0, 1, \times$	010	$e_{k+3} + h_{k+1} \mod q$
$0, 0, -1, \times$	011	$e_{k+3} - h_{k+1} \mod q$
$0, 1, \times, \times$	100	$e_{k+2} + h_{k+1} \mod q$
$0, -1, \times, \times$	101	$e_{k+2} - h_{k+1} \mod q$
$1, \times, \times, \times$	110	$e_{k+1} + h_{k+1} \mod q$
$-1, \times, \times, \times$	111	$e_{k+1} - h_{k+1} \mod q$

The corresponding algorithm for this arithmetic unit is proposed in Algorithm 5.6.

Algorithm 5.6 Proposed Arithmetic Unit

```
Input: $e_l = (e_{n-1}^{(l)} ... e_0^{(l)})_2$ $(l = k+1, k+2, k+3, k+4)$; $h_{k+1} = (h_{n-1}^{(k+1)} ... h_0^{(k+1)})_2$;
    $t_i = (t_2^{(i)} t_1^{(i)} t_0^{(i)})_2$;
Output: $e_k = (e_{n-1}^{(k)} ... e_0^{(k)})_2$;
 1: if $t_2^{(i)} t_1^{(i)} = 00$ then
 2:     if $t_0^{(i)} = 0$ then
 3:         $(e_{n-1}^{(k)} ... e_0^{(k)}) := (e_{n-1}^{(k+4)} ... e_0^{(k+4)})$
 4:     else
 5:         $(e_{n-1}^{(k)} ... e_0^{(k)}) := (e_{n-1}^{(k+3)} ... e_0^{(k+3)})$
 6:     end if
 7: else
 8:     if $t_0^{(i)} = 1$ then
 9:         $(h_{n-1}^{(k+1)} ... h_0^{(k+1)}) := -(h_{n-1}^{(k+1)} ... h_0^{(k+1)})$
10:     end if
11:     if $t_2^{(i)} t_1^{(i)} = 01$ then
12:         $(e_{n-1}^{(k)} ... e_0^{(k)}) := (e_{n-1}^{(k+3)} ... e_0^{(k+3)}) + (h_{n-1}^{(k+1)} ... h_0^{(k+1)})$
13:     else if $t_2^{(i)} t_1^{(i)} = 11$ then
14:         $(e_{n-1}^{(k)} ... e_0^{(k)}) := (e_{n-1}^{(k+1)} ... e_0^{(k+1)}) + (h_{n-1}^{(k+1)} ... h_0^{(k+1)})$
15:     else
16:         $(e_{n-1}^{(k)} ... e_0^{(k)}) := (e_{n-1}^{(k+2)} ... e_0^{(k+2)}) + (h_{n-1}^{(k+1)} ... h_0^{(k+1)})$
17:     end if
18: end if
```

The corresponding architecture for this arithmetic unit is shown in Fig.5.12.



$h_{n-1}^{(k+1)}...h_0^{(k+1)}$
$e_{n-1}^{(k+2)}...e_0^{(k+2)}$
$e_{n-1}^{(k+1)}...e_0^{(k+1)}$
$e_{n-1}^{(k+3)}...e_0^{(k+3)}$
$e_{n-1}^{(k+4)}...e_0^{(k+4)}$
n
$t_0^{(i)}$
0 MUX 1
10 MUX 11 01
$t_2^{(i)}t_1^{(i)}$
1 MUX 0
n-bit Modular Adder
01/11/10 MUX 00
$e_{n-1}^{(k)}...e_0^{(k)}$

Fig. 5.12: Proposed Arithmetic Unit Architecture

5.3.3 Proposed Multiplier III for Encryption

Based on our proposed arithmetic unit, we propose a multiplier architecture to implement NTRUEncrypt. Before the computation, we are required to encode $r(x) = (r_{N-1}, ..., r_0)$ to obtain $(t_{u-1}, ..., t_0)$, where each t_i is encoded in three bits. Then, each t_i is treated as a control input for one clock cycle.

The detailed encryption algorithm is shown in Algorithm 5.7. Three inputs, including the public key $h(x)$, the message polynomial $m(x)$ and the generated sequence $(t_{u-1}, ..., t_0)$, are required to obtain the encryption result $e(x)$ with N coefficients. The corresponding architecture is shown in Fig.5.13.

Algorithm 5.7 Encryption in NTRUEncrypt

```
Input: $m = m_{N-1}, ..., m_0$; $h = h_{N-1}, ..., h_0$; $(t_{u-1}, ..., t_0)$;
Output: $e = e_{N-1}, ..., e_0 = hr + m \mod q$;
1: $e^{(0)} := m$
2: for $j := 1$ to $u$ do
3:     for $i := 0$ to $N-1$ do
4:         if $t_{j-1} = 000$ then
5:             $e_i^{(j)} := e_{i+4 \mod N}^{(j-1)}$
6:         else if $t_{j-1} = 001$ then
7:             $e_i^{(j)} := e_{i+3 \mod N}^{(j-1)}$
8:         else if $t_{j-1} = 010$ then
9:             $e_i^{(j)} := e_{i+3 \mod N}^{(j-1)} + h_{i+1 \mod N} \mod q$
10:        else if $t_{j-1} = 011$ then
11:            $e_i^{(j)} := e_{i+3 \mod N}^{(j-1)} - h_{i+1 \mod N} \mod q$
12:        else if $t_{j-1} = 100$ then
13:            $e_i^{(j)} := e_{i+2 \mod N}^{(j-1)} + h_{i+1 \mod N} \mod q$
14:        else if $t_{j-1} = 101$ then
15:            $e_i^{(j)} := e_{i+2 \mod N}^{(j-1)} - h_{i+1 \mod N} \mod q$
16:        else if $t_{j-1} = 110$ then
17:            $e_i^{(j)} := e_{i+1 \mod N}^{(j-1)} + h_{i+1 \mod N} \mod q$
18:        else
19:            $e_i^{(j)} := e_{i+1 \mod N}^{(j-1)} - h_{i+1 \mod N} \mod q$
20:        end if
21:    end for
22: end for
23: return $e := e^{(u)}$
```

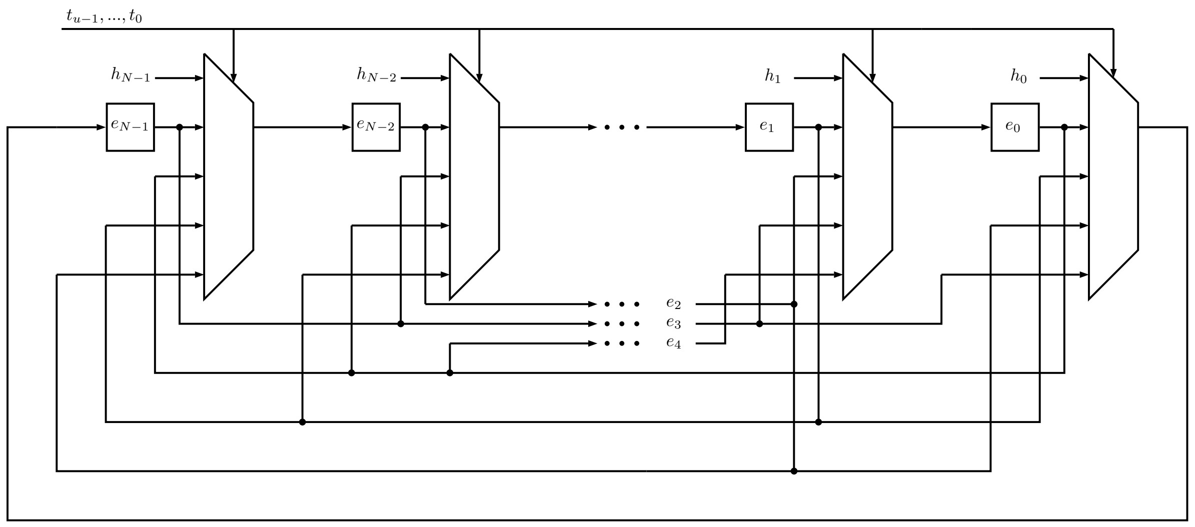


Fig. 5.13: Multiplier III Based NTRUEncrypt

Basically, this design requires N registers and N arithmetic units. The registers $e = (e_{N-1}, ..., e_0)$ are initially loaded with $m = (m_{N-1}, ..., m_0)$. When the operation starts, each t_i is scanned during one clock cycle. After u clock cycles, the phase-shift value is used to decide the location of the most significant coefficient of the encryption result. If the phase-shift value is one, the encryption result will be stored in the registers $e = (e_0, e_{N-1}, ..., e_1)$. If the phase-shift value is two, the encryption result will be stored in the registers $e = (e_1, e_0, e_{N-1}, ..., e_2)$. If the phase-shift value is zero, the encryption result will be stored in the registers $e = (e_{N-1}, ..., e_0)$.

5.3.4 Implementation of Multiplier III

In our FPGA implementation, the following tools are used.

- Quartus II v14.1 (64-bit) Software
- ModelSim-Altera Software

Our target device is Arria V 5AGXFB3H4F35I3 and we use Verilog HDL as our design language.

Implementation Results

The implementation results are shown in Table 5.17.

Table 5.17: Simulation Results for Different Parameter Sets

Security Level	Parameter set	#ALM	#Register	#Cycles	FMax	Latency
112	$ees401ep1$	9,029	8,826	246	110.53 MHz	2.23 μs
	$ees541ep1$	12,185	11,906	196	106.04 MHz	1.85 μs
	$ees659ep1$	14,880	14,502	213	110.47 MHz	1.93 μs
128	$ees449ep1$	10,155	9,882	286	111.26 MHz	2.57 μs
	$ees613ep1$	13,799	13,490	222	106.18 MHz	2.09 μs
	$ees761ep1$	17,147	16,746	243	105.54 MHz	2.30 μs
192	$ees677ep1$	15,343	14,898	367	108.93 MHz	3.37 μs
	$ees887ep1$	19,982	19,518	323	100.50 MHz	3.21 μs
	$ees1087ep1$	24,469	23,918	350	103.04 MHz	3.40 μs
256	$ees1087ep2$	24,469	23,918	420	103.04 MHz	4.08 μs
	$ees1171ep1$	26,363	25,766	424	101.65 MHz	4.17 μs
	$ees1499ep1$	33,749	32,982	473	98.98 MHz	4.78 μs

Comparison

For each security level, we choose one parameter set for comparison.

The comparison results for security level 112 are shown in Table 5.18.

Table 5.18: Security Level 112

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees401ep1$	4,052	9,638	401	62.95 MHz	6.37 μs
[11]	$ees401ep1$	837	1,165	3,617	67.69 MHz	53.43 μs
[12]	$ees401ep1$	15,662	8,838	227	55.00 MHz	4.13 μs
[15]	$ees401ep1$	4,636	8,826	401	121.62 MHz	3.30 μs
[16]	$ees401ep1$	9,044	8,826	349	113.67 MHz	3.07 μs
Multiplier III	$ees401ep1$	9,029	8,826	246	110.53 MHz	2.23 μs

It can be seen from the table that latency of Multiplier III is the lowest among all these similar works. Latency of Multiplier III is only 72.64%, compared to the fastest among the existing similar works.

The comparison results for security level 128 are shown in Table 5.19.

Table 5.19: Security Level 128

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees449ep1$	4,523	10,793	449	56.99 MHz	7.88 μs
[11]	$ees449ep1$	837	1,165	4,049	67.69 MHz	59.82 μs
[12]	$ees449ep1$	17,527	9,884	269	53.95 MHz	4.99 μs
[15]	$ees449ep1$	5,188	9,882	449	121.69 MHz	3.69 μs
[16]	$ees449ep1$	10,124	9,882	398	112.90 MHz	3.53 μs
Multiplier III	$ees449ep1$	10,155	9,882	286	111.26 MHz	2.57 μs

It can be seen from the table that latency of Multiplier III is the lowest among all these similar works. Latency of Multiplier III is only 72.80%, compared to the fastest among the existing similar works.

The comparison results for security level 192 are shown in Table 5.20.

Table 5.20: Security Level 192

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees677ep1$	6,740	16,266	677	60.24 MHz	11.24 μs
[11]	$ees677ep1$	837	1,165	9,486	67.69 MHz	140.14 μs
[12]	$ees677ep1$	26,423	14,900	317	49.74 MHz	6.37 μs
[15]	$ees677ep1$	7,810	14,898	677	120.00 MHz	5.64 μs
[16]	$ees677ep1$	15,254	14,898	551	106.30 MHz	5.18 μs
Multiplier III	$ees677ep1$	15,343	14,898	367	108.93 MHz	3.37 μs

It can be seen from the table that latency of Multiplier III is the lowest among all these similar works. Latency of Multiplier III is only 65.06%, compared to the fastest among the existing similar works.

The comparison results for security level 256 are shown in Table 5.21.

Table 5.21: Security Level 256

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees1087ep2$	10,748	26,105	1,087	55.53 MHz	19.58 μs
[11]	$ees1087ep2$	837	1,165	23,922	67.69 MHz	353.41 μs
[12]	$ees1087ep2$	42,427	23,930	276	45.75 MHz	6.03 μs
[15]	$ees1087ep2$	12,526	23,918	1,087	104.06 MHz	10.45 μs
[16]	$ees1087ep2$	24,480	23,918	717	94.07 MHz	7.62 μs
Multiplier III	$ees1087ep2$	24,469	23,918	420	103.04 MHz	4.08 μs

It can be seen from the table that latency of Multiplier III is the lowest among all these similar works. Latency of Multiplier III is only 67.66%, compared to the fastest among the existing similar works.

6 PROPOSED MULTIPLIER IV FOR NTRUENCRYPT

In this chapter, a truncated polynomial ring multiplication architecture is proposed for NTRUEncrypt. Firstly, we give a brief introduction to side channel attacks. Then, some basic idea for our proposed architecture is discussed. Next, we propose an arithmetic unit for our design. After that, a multiplier architecture is proposed, which is called Multiplier IV. Finally, the FPGA implementation results and a comparison of the proposed work with several existing works are given.

6.1 Side Channel Attacks

With the development of modern cryptography, cryptosystems are facing unprecedented challenges in terms of security. Security attacks are one of the most crucial challenges of all. There are mainly three types of attacks to NTRU based systems: brute force key search, mathematical attacks and side channel attacks.

In cryptography, a side channel attack is any attack based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithms. Side channel attacks consist of several classes:

- Timing attack. It is an attack that measuring how much time various computations take to perform.
- Power monitoring attack. It is an attack that measuring power consumption by the hardware during computation.
- Electromagnetic attack. It is an attack that measuring leaked electromagnetic radiation.
- Differential fault analysis. It is an attack that secrets are discovered by introducing faults in a computation.

Among all these side channel attacks, power monitoring attack is supposed to be the most common in hardware implementation of cryptosystems. Power monitoring attack, also called power analysis attack, is a form of side channel attack in which the attacker studies the power consumption of a cryptographic hardware device.

There are two kinds of power monitoring attacks: simple power analysis and differential power analysis. Simple power analysis involves visually interpreting power traces or graphs of electrical activity over time. Differential power analysis is a more advanced form of power analysis which can allow an attacker to compute the intermediate values within cryptographic computations by statistically analyzing data collected from multiple cryptographic operations.

6.2 General Idea

Since our proposed works are all based on hardware implementation, the power monitoring attack is considered as the most threatening attack to our proposed architectures. In specific, in each clock cycle of our previous proposed works, operations include shifting and addition (subtraction). Therefore, power consumption varies with different clock cycles, which makes our architectures vulnerable to power monitoring attacks.

In order to prevent architectures from power monitoring attacks, especially the simple power analysis, we propose a new structure, which is called Multiplier IV.

In this new design, three bits are used to encode the control input t_i and thus, eight encoding states can be obtained. The major difference is that in this architecture, computation in each clock cycle only contains one addition (subtraction) operation. Therefore, the power cost for each clock cycle is supposed to be the same and it is difficult for attackers to study the power consumption. Such design makes this proposed work resistant to power monitoring attacks.

The detailed encoding of t_i is shown in Table 6.2. In this scheme, we assume that

$r(x)$ does not contain more than three consecutive zeros. In such encoding, r_j represents the jth coefficient in $r(x)$ and we are supposed to scan each coefficient in $r(x)$ when we encode t_i. Note that such encoding does not contain coefficients of one zero, two consecutive zeros and three consecutive zeros in $r(x)$, we are required to calculate a phase-shift value during the last scan. There are totally four circumstances. If the last scan is one zero, no clock cycle will be utilized and the phase-shift value will be one. If the last scan is two consecutive zeros, no clock cycle will be utilized and the phase-shift value will be two. If the last scan is three consecutive zeros, no clock cycle will be utilized and the phase-shift value will be three. For the rest, clock cycles will be added by one and the phase-shift value will be zero. The phase-shift value will be used when the final result is read out and it will decide the location of the most significant coefficient of the encryption result polynomial.

Then, we are supposed to compute the number of cycles for different parameter sets. However, since we have a prerequisite for the number of consecutive zeros in $r(x)$, most of the parameter sets are not qualified due to their large number of zero coefficients. Specifically, only three of twelve can be used for our new design: $ees401ep1$, $ees449ep1$ and $ees677ep1$. According to Table 6.2, since each encoding is ended with a non-zero coefficient, the number of cycles is equivalent to the number of non-zero coefficients in $r(x)$. Hence, we can obtain the number of cycles for these three parameter sets, which is given in Table 6.1.

Table 6.1: Average Number of Cycles for Different Parameter Sets

Security Level	Parameter set	N	p	q	d_r	#Cycles
112	$ees401ep1$	401	3	2048	113	226
128	$ees449ep1$	449	3	2048	134	268
192	$ees677ep1$	677	3	2048	157	314

6.3 Proposed Arithmetic Unit

In our new arithmetic unit, the inputs e_{k+1}, e_{k+2}, e_{k+3}, e_{k+4} and h_{k+1} are all encoded in $n = [\log_2 q]$ bits while the control input t_i is encoded in three bits. The output e_k is also encoded in $n = [\log_2 q]$ bits. The proposed arithmetic unit is shown in Fig.6.1 and the operation table is given in Table 6.2. There is no redundant state in such design, as eight states of three bits are fully used.

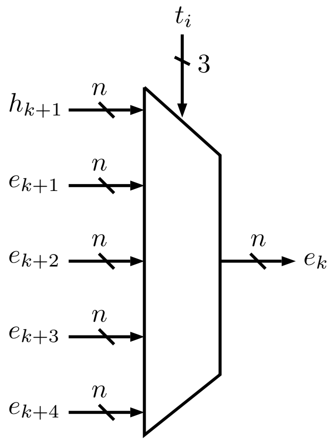


Fig. 6.1: Proposed Arithmetic Unit

Table 6.2: Operations Supported with the Arithmetic Unit

$r_j, r_{j+1}, r_{j+2}, r_{j+3}$	Input t_i $(t_2^{(i)}t_1^{(i)}t_0^{(i)})$	Output e_k
$0, 0, 0, 1$	000	$e_{k+4} + h_{k+1} \mod q$
$0, 0, 0, -1$	001	$e_{k+4} - h_{k+1} \mod q$
$0, 0, 1, \times$	010	$e_{k+3} + h_{k+1} \mod q$
$0, 0, -1, \times$	011	$e_{k+3} - h_{k+1} \mod q$
$0, 1, \times, \times$	100	$e_{k+2} + h_{k+1} \mod q$
$0, -1, \times, \times$	101	$e_{k+2} - h_{k+1} \mod q$
$1, \times, \times, \times$	110	$e_{k+1} + h_{k+1} \mod q$
$-1, \times, \times, \times$	111	$e_{k+1} - h_{k+1} \mod q$

The corresponding algorithm for this arithmetic unit is proposed in Algorithm 6.1.

Algorithm 6.1 Proposed Arithmetic Unit

```
Input: $e_l = (e_{n-1}^{(l)}...e_0^{(l)})_2$ $(l = k+1, k+2, k+3, k+4)$; $h_{k+1} = (h_{n-1}^{(k+1)}...h_0^{(k+1)})_2$;
    $t_i = (t_2^{(i)}t_1^{(i)}t_0^{(i)})_2$;
Output: $e_k = (e_{n-1}^{(k)}...e_0^{(k)})_2$;
1: if $t_0^{(i)} = 1$ then
2:     $(h_{n-1}^{(k+1)}...h_0^{(k+1)}) := -(h_{n-1}^{(k+1)}...h_0^{(k+1)})$
3: end if
4: if $t_2^{(i)}t_1^{(i)} = 00$ then
5:     $(e_{n-1}^{(k)}...e_0^{(k)}) := (e_{n-1}^{(k+4)}...e_0^{(k+4)}) + (h_{n-1}^{(k+1)}...h_0^{(k+1)})$
6: else if $t_2^{(i)}t_1^{(i)} = 01$ then
7:     $(e_{n-1}^{(k)}...e_0^{(k)}) := (e_{n-1}^{(k+3)}...e_0^{(k+3)}) + (h_{n-1}^{(k+1)}...h_0^{(k+1)})$
8: else if $t_2^{(i)}t_1^{(i)} = 11$ then
9:     $(e_{n-1}^{(k)}...e_0^{(k)}) := (e_{n-1}^{(k+1)}...e_0^{(k+1)}) + (h_{n-1}^{(k+1)}...h_0^{(k+1)})$
10: else
11:     $(e_{n-1}^{(k)}...e_0^{(k)}) := (e_{n-1}^{(k+2)}...e_0^{(k+2)}) + (h_{n-1}^{(k+1)}...h_0^{(k+1)})$
12: end if
```

The architecture for this arithmetic unit is shown in Fig.6.2.

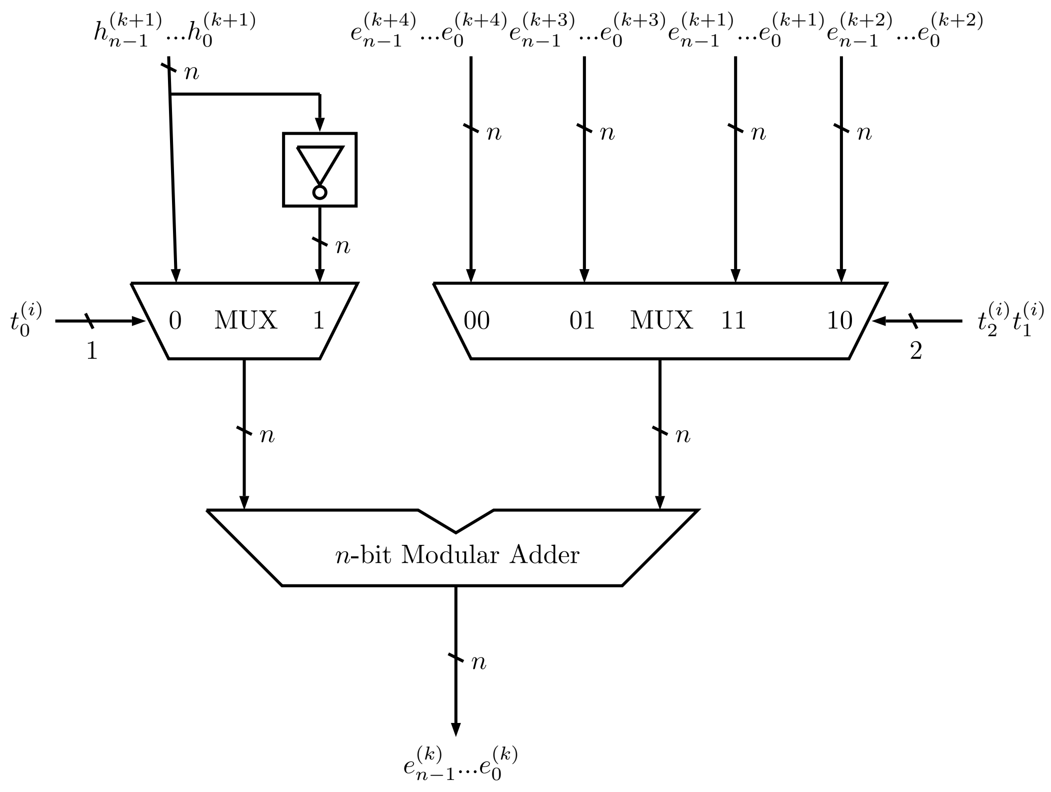

$h_{n-1}^{(k+1)}...h_0^{(k+1)}$
$e_{n-1}^{(k+4)}...e_0^{(k+4)}$
$e_{n-1}^{(k+3)}...e_0^{(k+3)}$
$e_{n-1}^{(k+1)}...e_0^{(k+1)}$
$e_{n-1}^{(k+2)}...e_0^{(k+2)}$
n
$t_0^{(i)}$
1
0 MUX 1
00 01 MUX 11 10
$t_2^{(i)}t_1^{(i)}$
2
n-bit Modular Adder
$e_{n-1}^{(k)}...e_0^{(k)}$

Fig. 6.2: Proposed Arithmetic Unit Architecture

6.4 Proposed Multiplier IV for Encryption

Based on our proposed arithmetic unit, a multiplier architecture is proposed to implement NTRUEncrypt. Before the computation, we are required to encode $r(x) = (r_{N-1}, ..., r_0)$ to obtain $(t_{u-1}, ..., t_0)$, where each t_i has three bits. Then, each t_i is treated as a control input for one clock cycle.

The detailed encryption algorithm is shown in 6.2. Three inputs, including the public key $h(x)$, the message polynomial $m(x)$ and the generated sequence $(t_{u-1}, ..., t_0)$, are required to obtain the encryption result $e(x)$ with N coefficients. The corresponding architecture is given in Fig.6.3.

Algorithm 6.2 Encryption in NTRUEncrypt

```
Input: $m = m_{N-1}, ..., m_0$; $h = h_{N-1}, ..., h_0$; $(t_{u-1}, ..., t_0)$;
Output: $e = e_{N-1}, ..., e_0 = hr + m \mod q$;
 1: $e^{(0)} := m$
 2: for $j := 1$ to $u$ do
 3:     for $i := 0$ to $N-1$ do
 4:         if $t_{j-1} = 000$ then
 5:             $e_i^{(j)} := e_{i+4 \mod N}^{(j-1)} + h_{i+1 \mod N} \mod q$
 6:         else if $t_{j-1} = 001$ then
 7:             $e_i^{(j)} := e_{i+4 \mod N}^{(j-1)} - h_{i+1 \mod N} \mod q$
 8:         else if $t_{j-1} = 010$ then
 9:             $e_i^{(j)} := e_{i+3 \mod N}^{(j-1)} + h_{i+1 \mod N} \mod q$
10:         else if $t_{j-1} = 011$ then
11:             $e_i^{(j)} := e_{i+3 \mod N}^{(j-1)} - h_{i+1 \mod N} \mod q$
12:         else if $t_{j-1} = 100$ then
13:             $e_i^{(j)} := e_{i+2 \mod N}^{(j-1)} + h_{i+1 \mod N} \mod q$
14:         else if $t_{j-1} = 101$ then
15:             $e_i^{(j)} := e_{i+2 \mod N}^{(j-1)} - h_{i+1 \mod N} \mod q$
16:         else if $t_{j-1} = 110$ then
17:             $e_i^{(j)} := e_{i+1 \mod N}^{(j-1)} + h_{i+1 \mod N} \mod q$
18:         else
19:             $e_i^{(j)} := e_{i+1 \mod N}^{(j-1)} - h_{i+1 \mod N} \mod q$
20:         end if
21:     end for
22: end for
23: return $e := e^{(u)}$
```

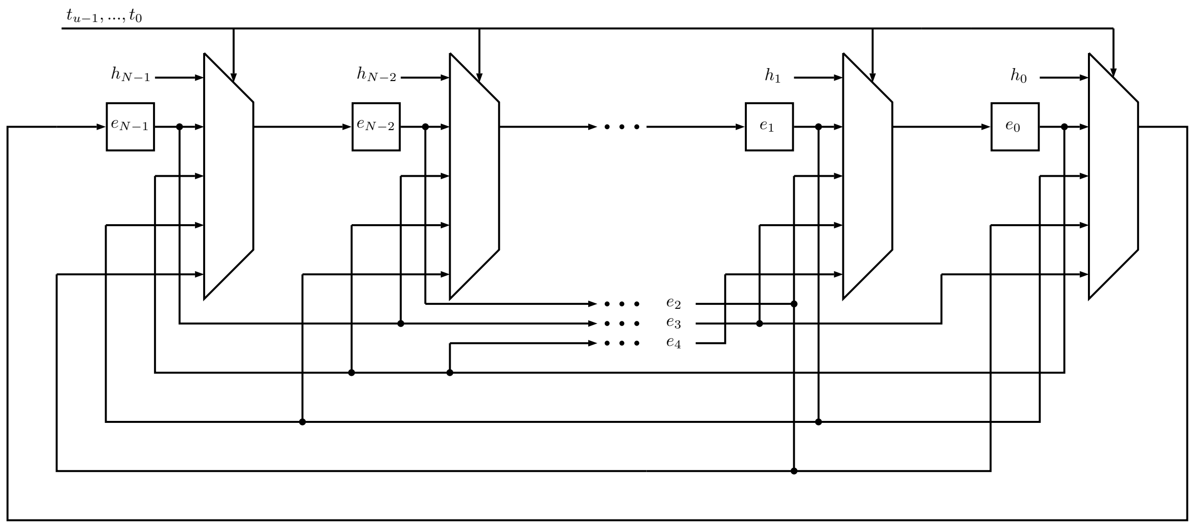


Fig. 6.3: Multiplier IV Based NTRUEncrypt

Our new design requires N registers and N arithmetic units totally. The registers $e = (e_{N-1}, ..., e_0)$ are initially loaded with $m = (m_{N-1}, ..., m_0)$. When the operation starts, each t_i is scanned during one clock cycle. After u clock cycles, the phase-shift value is used to decide the location of the most significant coefficient of the encryption result. If the phase-shift value is one, the encryption result will be stored in the registers $e = (e_0, e_{N-1}, ..., e_1)$. If the phase-shift value is two, the encryption result will be stored in the registers $e = (e_1, e_0, e_{N-1}, ..., e_2)$. If the phase-shift value is three, the encryption result will be stored in the registers $e = (e_2, e_1, e_0, e_{N-1}, ..., e_3)$. If the phase-shift value is zero, the encryption result will be stored in the registers $e = (e_{N-1}, ..., e_0)$.

6.5 Implementation of Multiplier IV

In our FPGA implementation, the following tools are used.

- Quartus II v14.1 (64-bit) Software
- ModelSim-Altera Software

Our target device is Arria V 5AGXFB3H4F35I3 and we use Verilog HDL as our design language.

Implementation Results

The implementation results are shown in Table 6.3.

Table 6.3: Simulation Results for Different Parameter Sets

Security Level	Parameter set	#ALM	#Register	#Cycles	FMax	Latency
112	$ees401ep1$	9,026	8,826	226	118.41 MHz	1.91 μs
128	$ees449ep1$	10,105	9,882	268	117.81 MHz	2.27 μs
192	$ees677ep1$	15,235	14,898	314	119.50 MHz	2.63 μs

Comparison

The comparison results for security level 112 are shown in Table 6.4.

Table 6.4: Security Level 112

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees401ep1$	4,052	9,638	401	62.95 MHz	6.37 μs
[11]	$ees401ep1$	837	1,165	3,617	67.69 MHz	53.43 μs
[12]	$ees401ep1$	15,662	8,838	227	55.00 MHz	4.13 μs
[15]	$ees401ep1$	4,636	8,826	401	121.62 MHz	3.30 μs
[16]	$ees401ep1$	9,044	8,826	349	113.67 MHz	3.07 μs
Multiplier IV	$ees401ep1$	9,026	8,826	226	118.41 MHz	1.91 μs

It can be seen from the table that compared to all the existing works, Multiplier IV has the least number of cycles. Furthermore, the highest speed can be achieved by Multiplier IV. Latency of Multiplier IV is only 62.21%, compared to the fastest of existing works.

The comparison results for security level 128 are given in Table 6.5.

Table 6.5: Security Level 128

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees449ep1$	4,523	10,793	449	56.99 MHz	7.88 μs
[11]	$ees449ep1$	837	1,165	4,049	67.69 MHz	59.82 μs
[12]	$ees449ep1$	17,527	9,884	269	53.95 MHz	4.99 μs
[15]	$ees449ep1$	5,188	9,882	449	121.69 MHz	3.69 μs
[16]	$ees449ep1$	10,124	9,882	398	112.90 MHz	3.53 μs
Multiplier IV	$ees449ep1$	10,105	9,882	268	117.81 MHz	2.27 μs

It can be seen from the table that compared to all the existing works, Multiplier IV has the least number of cycles. Moreover, the highest speed can be achieved. Latency of Multiplier IV is only 64.31%, compared to the fastest of existing works.

The comparison results for security level 192 are presented in Table 6.6.

Table 6.6: Security Level 192

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees677ep1$	6,740	16,266	677	60.24 MHz	11.24 μs
[11]	$ees677ep1$	837	1,165	9,486	67.69 MHz	140.14 μs
[12]	$ees677ep1$	26,423	14,900	317	49.74 MHz	6.37 μs
[15]	$ees677ep1$	7,810	14,898	677	120.00 MHz	5.64 μs
[16]	$ees677ep1$	15,254	14,898	551	106.30 MHz	5.18 μs
Multiplier IV	$ees677ep1$	15,235	14,898	314	119.50 MHz	2.63 μs

It can be seen from the table that compared to all the existing works, Multiplier IV has the least number of cycles. Moreover, the highest speed can be achieved. Latency of Multiplier IV is only 50.77%, compared to the fastest of existing works.

7 CONCLUSIONS AND FUTURE WORKS

7.1 Conclusions

In this thesis, four efficient multiplication architectures for truncated polynomial ring are proposed. All these architectures can be applied to NTRUEncrypt based systems.

First of all, Multiplier I is proposed to implement NTRUEncrypt. Multiplier I takes advantage of x^2-net architecture, which makes it possible to handle two consecutive coefficients in the control input polynomial during one clock cycle. Thus, number of cycles for this architecture is fixed. Another merit of this architecture is that we are not required to scan and encode the control input polynomial before the computation. The FPGA simulation results show that compared with all the existing works, Multiplier I has the best performance in terms of latency. For different security levels 112-bit, 128-bit, 192-bit, and 256-bit, the latency of the proposed multiplier is only 63.52%, 60.62%, 61.20%, and 93.53%, compared to the lowest latency among the existing similar works.

Then, we propose Multiplier II to implement NTRUEncrypt. It takes advantage of three consecutive zeros in polynomial coefficients by re-coding the polynomial $r(x)$. It can be seen from the FPGA simulation results that Multiplier II is the second best in speed compared to existing works, but has better area-latency-product compared to the fastest existing work for the first set of parameters $ees401ep1$, $ees449ep1$, $ees677ep1$, at security level 112-bit, 128-bit and 192-bit, respectively.

Next, Multiplier III is given to implement NTRUEncrypt. It takes advantage of consecutive zeros in polynomial coefficients by re-coding the polynomial $r(x)$. It can be seen from the FPGA simulation results that it is faster than any existing works in comparison for all IEEE recommended parameter sets.

Finally, we propose Multiplier IV to implement NTRUEncrypt. It has an advantage in resistance to side channel attacks. From the FPGA simulation results, it can be seen that Multiplier IV outperforms all the existing works in terms of latency for all three parameter sets $ees401ep1$, $ees449ep1$ and $ees677ep1$.

A comprehensive comparison list for all IEEE recommended parameter sets is given in Appendix A.

7.2 Possible Future Works

Based on the research works on truncated polynomial ring multipliers presented in this thesis, we propose the following research topics as possible future works.

- Multiplier IV can be applied to only three parameter sets due to some restrictions on the control input polynomial. Further improvement can be achieved if we can extend this architecture to a new one that can be applied to all parameter sets.

- Since the LTV homomorphic encryption scheme is regarded as a potential practical fully homomorphic encryption in cloud computing, it is possible for us to propose a hardware implementation for such scheme.

REFERENCES

[1] W. G. of the C/MM Committee, "IEEE p1363.1 standard specification for public-key cryptographic techniques based on hard problems over lattices," 2009.

[2] D. Cabarcas, P. Weiden, and J. Buchmann, "On the efficiency of provably secure NTRU," in *International Workshop on Post-Quantum Cryptography.* Springer International Publishing, 2014, pp. 22–39.

[3] P. S. Hirschhorn, J. Hoffstein, N. Howgrave-Graham, and W. Whyte, "Choosing NTRUEncrypt parameters in light of combined lattice reduction and mitm approaches," in *International Conference on Applied Cryptography and Network Security.* Springer Berlin Heidelberg, 2009, pp. 437–455.

[4] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Review*, vol. 41, no. 2, pp. 303–332, 1999.

[5] D. J. Bernstein, J. Buchmann, and E. Dahmen, *Post-quantum cryptography.* Springer Science & Business Media, 2009.

[6] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *International Algorithmic Number Theory Symposium.* Springer Berlin Heidelberg, 1998, pp. 267–288.

[7] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009.

[8] D. Stehlé and R. Steinfeld, "Making NTRU as secure as worst-case problems over ideal lattices," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer Berlin Heidelberg, 2011, pp. 27–47.

[9] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing.* ACM, 2012, pp. 1219–1234.

[10] D. V. Bailey, D. Coffin, A. Elbirt, J. H. Silverman, and A. D. Woodbury, "NTRU in constrained devices," in *International Workshop on Cryptographic Hardware and Embedded Systems.* Springer Berlin Heidelberg, 2001, pp. 262–272.

[11] C. M. ORourke, "Efficient NTRU implementations," Master's thesis, Worcester Polytechnic Institute, 2002.

[12] A. A. Kamal and A. M. Youssef, "An fpga implementation of the NTRUEncrypt cryptosystem," in *2009 International Conference on Microelectronics-ICM.* IEEE, 2009, pp. 209–212.

[13] J.-P. Kaps, "Cryptography for ultra-low power devices," Ph.D. dissertation, Worcester Polytechnic Institute, 2006.

[14] A. C. Atici, L. Batina, J. Fan, I. Verbauwhede, and S. B. O. Yalcin, "Low-cost implementations of NTRU for pervasive security," in *2008 International Conference on Application-Specific Systems, Architectures and Processors.* IEEE, 2008, pp. 79–84.

[15] B. Liu and H. Wu, "Efficient architecture and implementation for NTRUEncrypt system," in *2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS).* IEEE, 2015, pp. 1–4.

[16] B. Liu, "Efficient architecture and implementation for NTRU based systems," Master's thesis, University of Windsor, 2015.

[17] H. Raj, R. Nathuji, A. Singh, and P. England, “Resource management for isolation enhanced cloud services,” in *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM, 2009, pp. 77–84.

[18] R. L. Rivest, L. Adleman, and M. L. Dertouzos, “On data banks and privacy homomorphisms,” *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[19] S. Goldwasser and S. Micali, “Probabilistic encryption & how to play mental poker keeping secret all partial information,” in *Proceedings of the fourteenth annual ACM symposium on Theory of computing*. ACM, 1982, pp. 365–377.

[20] J. Feigenbaum and M. Merritt, “Open questions, talk abstracts, and summary of discussions,” 1991.

[21] T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” in *Workshop on the Theory and Application of Cryptographic Techniques*. Springer Berlin Heidelberg, 1984, pp. 10–18.

[22] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer Berlin Heidelberg, 1999, pp. 223–238.

[23] N. P. Smart and F. Vercauteren, “Fully homomorphic encryption with relatively small key and ciphertext sizes,” in *International Workshop on Public Key Cryptography*. Springer Berlin Heidelberg, 2010, pp. 420–443.

[24] C. Gentry and S. Halevi, “Fully homomorphic encryption without squashing using depth-3 arithmetic circuits,” in *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*. IEEE, 2011, pp. 107–109.

[25] C. Gentry and S. Halevi, “Implementing gentrys fully-homomorphic encryption scheme,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer Berlin Heidelberg, 2011, pp. 129–148.

[26] N. P. Smart and F. Vercauteren, “Fully homomorphic SIMD operations,” *Designs, codes and cryptography*, vol. 71, no. 1, pp. 57–81, 2014.

[27] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “(leveled) fully homomorphic encryption without bootstrapping,” in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ACM, 2012, pp. 309–325.

[28] Z. Brakerski, “Fully homomorphic encryption without modulus switching from classical gapsvp,” in *Advances in CryptologyCRYPTO 2012*. Springer Berlin Heidelberg, 2012, pp. 868–886.

[29] C. Gentry, A. Sahai, and B. Waters, “Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based,” in *Advances in CryptologyCRYPTO 2013*. Springer Berlin Heidelberg, 2013, pp. 75–92.

[30] Z. Brakerski and V. Vaikuntanathan, “Efficient fully homomorphic encryption from (standard) lwe,” *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.

[31] O. Regev, “On lattices, learning with errors, random linear codes, and cryptography,” *Journal of the ACM (JACM)*, vol. 56, no. 6, p. 34, 2009.

APPENDIX A

The complete comparison results among our proposed works and some existing works are listed as follows.

Table A.1: FPGA Results for $ees401ep1$, Security Level 112-bit

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees401ep1$	4,052	9,638	401	62.95 MHz	6.37 μs
[11]	$ees401ep1$	837	1,165	3,617	67.69 MHz	53.43 μs
[12]	$ees401ep1$	15,662	8,838	227	55.00 MHz	4.13 μs
[15]	$ees401ep1$	4,636	8,826	401	121.62 MHz	3.30 μs
[16]	$ees401ep1$	9,044	8,826	349	113.67 MHz	3.07 μs
Multiplier I	$ees401ep1$	11,861	8,826	201	103.01 MHz	1.95 μs
Multiplier II	$ees401ep1$	6,888	8,826	381	121.25 MHz	3.14 μs
Multiplier III	$ees401ep1$	9,029	8,826	246	110.53 MHz	2.23 μs
Multiplier IV	$ees401ep1$	9,026	8,826	226	118.41 MHz	1.91 μs

Table A.2: FPGA Results for $ees541ep1$, Security Level 112-bit

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees541ep1$	5,425	12,995	541	61.50 MHz	8.80 μs
[11]	$ees541ep1$	837	1,165	5,959	67.69 MHz	88.03 μs
[12]	$ees541ep1$	21,124	11,918	121	52.87 MHz	2.29 μs
[15]	$ees541ep1$	6,246	11,906	541	122.02 MHz	4.43 μs
[16]	$ees541ep1$	12,194	11,906	342	116.14 MHz	2.94 μs
Multiplier I	$ees541ep1$	16,091	11,906	271	102.33 MHz	2.65 μs
Multiplier II	$ees541ep1$	9,256	11,906	422	121.76 MHz	3.47 μs
Multiplier III	$ees541ep1$	12,185	11,906	196	106.04 MHz	1.85 μs

Table A.3: FPGA Results for $ees659ep1$, Security Level 112-bit

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees659ep1$	6,572	15,832	659	59.43 MHz	11.09 μs
[11]	$ees659ep1$	837	1,165	9,234	67.69 MHz	136.42 μs
[12]	$ees659ep1$	25,726	14,514	107	51.08 MHz	2.09 μs
[15]	$ees659ep1$	7,603	14,502	659	114.53 MHz	5.75 μs
[16]	$ees659ep1$	14,850	14,502	386	106.92 MHz	3.61 μs
Multiplier I	$ees659ep1$	19,573	14,502	330	105.96 MHz	3.11 μs
Multiplier II	$ees659ep1$	11,267	14,502	489	113.57 MHz	4.31 μs
Multiplier III	$ees659ep1$	14,880	14,502	213	110.47 MHz	1.93 μs

Table A.4: FPGA Results for $ees449ep1$, Security Level 128-bit

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees449ep1$	4,523	10,793	449	56.99 MHz	7.88 μs
[11]	$ees449ep1$	837	1,165	4,049	67.69 MHz	59.82 μs
[12]	$ees449ep1$	17,527	9,884	269	53.95 MHz	4.99 μs
[15]	$ees449ep1$	5,188	9,882	449	121.69 MHz	3.69 μs
[16]	$ees449ep1$	10,124	9,882	398	112.90 MHz	3.53 μs
Multiplier I	$ees449ep1$	13,296	9,882	225	105.24 MHz	2.14 μs
Multiplier II	$ees449ep1$	7,731	9,882	431	120.12 MHz	3.59 μs
Multiplier III	$ees449ep1$	10,155	9,882	286	111.26 MHz	2.57 μs
Multiplier IV	$ees449ep1$	10,105	9,882	268	117.81 MHz	2.27 μs

Table A.5: FPGA Results for $ees613ep1$, Security Level 128-bit

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees613ep1$	6,124	14,730	613	61.62 MHz	9.95 μs
[11]	$ees613ep1$	837	1,165	7,977	67.69 MHz	117.85 μs
[12]	$ees613ep1$	23,931	13,502	135	50.92 MHz	2.65 μs
[15]	$ees613ep1$	7,075	13,490	613	115.12 MHz	5.32 μs
[16]	$ees613ep1$	13,814	13,490	387	111.08 MHz	3.48 μs
Multiplier I	$ees613ep1$	18,201	13,490	307	104.60 MHz	2.93 μs
Multiplier II	$ees613ep1$	10,469	13,490	478	113.09 MHz	4.23 μs
Multiplier III	$ees613ep1$	13,799	13,490	222	106.18 MHz	2.09 μs

Table A.6: FPGA Results for $ees761ep1$, Security Level 128-bit

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees761ep1$	7,571	18,282	761	59.53 MHz	12.78 μs
[11]	$ees761ep1$	837	1,165	12,184	67.69 MHz	180.00 μs
[12]	$ees761ep1$	29,707	16,758	120	49.61 MHz	2.42 μs
[15]	$ees761ep1$	8,776	16,746	761	110.92 MHz	6.86 μs
[16]	$ees761ep1$	17,144	16,746	443	109.14 MHz	4.06 μs
Multiplier I	$ees761ep1$	22,459	16,746	381	103.33 MHz	3.69 μs
Multiplier II	$ees761ep1$	13,023	16,746	562	108.70 MHz	5.17 μs
Multiplier III	$ees761ep1$	17,147	16,746	243	105.54 MHz	2.30 μs

Table A.7: FPGA Results for $ees677ep1$, Security Level 192-bit

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees677ep1$	6,740	16,266	677	60.24 MHz	11.24 μs
[11]	$ees677ep1$	837	1,165	9,486	67.69 MHz	140.14 μs
[12]	$ees677ep1$	26,423	14,900	317	49.74 MHz	6.37 μs
[15]	$ees677ep1$	7,810	14,898	677	120.00 MHz	5.64 μs
[16]	$ees677ep1$	15,254	14,898	551	106.30 MHz	5.18 μs
Multiplier I	$ees677ep1$	20,042	14,898	339	106.90 MHz	3.17 μs
Multiplier II	$ees677ep1$	11,551	14,898	620	118.38 MHz	5.24 μs
Multiplier III	$ees677ep1$	15,343	14,898	367	108.93 MHz	3.37 μs
Multiplier IV	$ees677ep1$	15,235	14,898	314	119.50 MHz	2.63 μs

Table A.8: FPGA Results for $ees887ep1$, Security Level 192-bit

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees887ep1$	8,788	21,305	887	60.12 MHz	14.75 μs
[11]	$ees887ep1$	837	1,165	15,974	67.69 MHz	235.99 μs
[12]	$ees887ep1$	34,622	19,530	198	47.70 MHz	4.15 μs
[15]	$ees887ep1$	10,226	19,518	887	106.24 MHz	8.35 μs
[16]	$ees887ep1$	19,980	19,518	562	99.98 MHz	5.62 μs
Multiplier I	$ees887ep1$	26,295	19,518	444	103.68 MHz	4.28 μs
Multiplier II	$ees887ep1$	15,163	19,518	693	102.30 MHz	6.77 μs
Multiplier III	$ees887ep1$	19,982	19,518	323	100.50 MHz	3.21 μs

Table A.9: FPGA Results for $ees1087ep1$, Security Level 192-bit

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees1087ep1$	10,748	26,105	1,087	55.53 MHz	19.58 μs
[11]	$ees1087ep1$	837	1,165	23,922	67.69 MHz	353.41 μs
[12]	$ees1087ep1$	42,427	23,930	177	47.19 MHz	3.75 μs
[15]	$ees1087ep1$	12,526	23,918	1,087	104.06 MHz	10.45 μs
[16]	$ees1087ep1$	24,480	23,918	637	94.07 MHz	6.77 μs
Multiplier I	$ees1087ep1$	32,241	23,918	544	96.43 MHz	5.64 μs
Multiplier II	$ees1087ep1$	18,613	23,918	806	103.01 MHz	7.82 μs
Multiplier III	$ees1087ep1$	24,469	23,918	350	103.04 MHz	3.40 μs

Table A.10: FPGA Results for $ees1087ep2$, Security Level 256-bit

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees1087ep2$	10,748	26,105	1,087	55.53 MHz	19.58 μs
[11]	$ees1087ep2$	837	1,165	23,922	67.69 MHz	353.41 μs
[12]	$ees1087ep2$	42,427	23,930	276	45.75 MHz	6.03 μs
[15]	$ees1087ep2$	12,526	23,918	1,087	104.06 MHz	10.45 μs
[16]	$ees1087ep2$	24,480	23,918	717	94.07 MHz	7.62 μs
Multiplier I	$ees1087ep2$	32,241	23,918	544	96.43 MHz	5.64 μs
Multiplier II	$ees1087ep2$	18,613	23,918	872	103.01 MHz	8.47 μs
Multiplier III	$ees1087ep2$	24,469	23,918	420	103.04 MHz	4.08 μs

Table A.11: FPGA Results for $ees1171ep1$, Security Level 256-bit

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees1171ep1$	11,568	28,121	1,171	59.28 MHz	19.75 μs
[11]	$ees1171ep1$	837	1,165	28,112	67.69 MHz	415.31 μs
[12]	$ees1171ep1$	45,703	25,778	261	44.60 MHz	5.85 μs
[15]	$ees1171ep1$	13,491	25,766	1,171	108.31 MHz	10.81 μs
[16]	$ees1171ep1$	26,370	25,766	740	93.48 MHz	7.92 μs
Multiplier I	$ees1171ep1$	34,648	25,766	586	99.50 MHz	5.89 μs
Multiplier II	$ees1171ep1$	19,964	25,766	913	103.79 MHz	8.80 μs
Multiplier III	$ees1171ep1$	26,363	25,766	424	101.65 MHz	4.17 μs

Table A.12: FPGA Results for $ees1499ep1$, Security Level 256-bit

Work	Parameter Set	#ALM	#Register	#Cycles	FMax	Latency
[10]	$ees1499ep1$	14,755	35,989	1,499	51.87 MHz	28.90 μs
[11]	$ees1499ep1$	837	1,165	44,978	67.69 MHz	664.47 μs
[12]	$ees1499ep1$	58,507	32,994	228	42.95 MHz	5.31 μs
[15]	$ees1499ep1$	17,263	32,982	1,499	103.68 MHz	14.46 μs
[16]	$ees1499ep1$	33,750	32,982	866	96.99 MHz	8.93 μs
Multiplier I	$ees1499ep1$	48,719	32,982	750	82.73 MHz	9.07 μs
Multiplier II	$ees1499ep1$	25,543	32,982	1,102	97.73 MHz	11.28 μs
Multiplier III	$ees1499ep1$	33,749	32,982	473	98.98 MHz	4.78 μs

VITA AUCTORIS

NAME:	Ruiqing Dong
PLACE OF BIRTH:	Shanghai, China
YEAR OF BIRTH:	1990
EDUCATION:	East China Normal University, Shanghai, China Bachelor of Engineering, Software Engineering 2009-2013
	University of Windsor, Windsor, ON, Canada Master of Applied Science, Electrical and Computer Engineering 2014-2016