10-19-2015

Efficient Architecture and Implementation for NTRU Based Systems

Bingxin Liu
University of Windsor

Efficient Architecture and Implementation for NTRU Based Systems

by

Bingxin Liu

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

2015

Efficient Architecture and Implementation for NTRU Based Systems

by

Bingxin Liu

APPROVED BY:

Dr. Henry. Hu

Department of Mechanical, Automotive, & Materials Engineering

---

Dr. H. K. Kwan

Department of Electrical and Computer Engineering

Dr. H. Wu, Supervisor

Department of Electrical and Computer Engineering

August 28, 2015

DECLARATION OF

CO-AUTHORSHIP/PREVIOUS PUBLICATION

I DECLARATION OF CO-AUTHORSHIP

I hereby declare that this thesis presents the research outcome under the supervision of professor, Dr. Huapeng Wu. In all cases, the key ideas, primary contributions, experimental designs, data analysis and interpretation, were performed by the author, with the review and revision being provided by Prof. Wu. This joint research was submitted to the International Midwest Symposium on Circuits and Systems 2015, as specified in the declaration below.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

II DECLARATION OF PREVIOUS PUBLICATION

This thesis includes 1 original paper that has been previously submitted for publication in a peer reviewed conference, as follows:

Thesis Chapter/Section	Publication title/full citation	Publication status
Chapter 4	Efficient Architecture and Implementation for NTRUEncrypt System, IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS 2015), Aug 2-5, Fort Collins, Colorado, USA	Published

I certify that I have obtained a written permission from the copyright owner to include the above published material in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Windsor.

I declare 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. 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

NTRU (Nth degree Truncated polynomial Ring Units) is probably the only post quantum public key cryptosystem suitable for practical implementation. Recently, several NTRU based systems have also been shown having property of homomorphic encryption with important application in cloud computing security.

In this thesis, several efficient algorithms and architectures for NTRUEcrypt system and for NTRU based homomorphic encryption system are proposed. For NTRUEncrypt system, a new LFSR (linear feedback shift register) based architecture is firstly presented. A novel design of the modular arithmetic unit is proposed to reduce the critical path delay. The FPGA implementation results have shown that the proposed design outperforms all the existing works in terms of area-delay product. Secondly, a new architecture using extended LFSR is proposed for NTRUEncrypt system. It takes advantage of small polynomials with many zero coefficients, and thus significantly reduces the latency of the computation with modest increase of the complexity. Thirdly, a systolic array architecture is proposed for NTRUEncrypt. There is only one type of PE (process element) in the array and the PE was designed with optimized arithmetic. The systolic array yields all the output in N clock cycles.

Two new architectures are proposed for computation of NTRU based fully homomorphic encryption system. One architecture uses LFSR with a novel design of the modular multiplication unit, and the other proposed architecture is systolic array based which uses two types of PEs.

DEDICATION

To my loving parents:

Father: Bocai Liu

Mother: Qiurong Qin

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to everyone who helped make this thesis possible. I am deeply indebted to my supervisor Prof. Huapeng Wu, Professor of Electrical and Computer Engineering at University of Windsor, for guiding me throughout the writing of this thesis. As one of best teachers I have ever had, Professor Wu impressed upon me that a good teacher instructs students in matters far beyond those in textbooks. His broad knowledge and logical way of thinking have been of great value; without his detailed and constructive comments on my research, none of this thesis would be possible. I would also grateful to my colleagues and friends, Yiruo He, Zheng Gong and Wu Zheng and for their time and support. Finally, I with to extend my gratitude to everyone at UWindsors Faculty of ECE for their efforts during my study in the M.A.Sc. Program. I also gratefully acknowledge the financial support form University of Windsor and Professor Huapeng Wu.

Bingxin Liu

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ACRONYMS

ECC Elliptic Curve Cryptosystem

EMAU Extended Modular Arithmetic Unit

FHE Fully Homomorphic Encryption

HE Homomorphic Encryption

LFSR Linear Feedback Shift Register

MAU Modular Arithmetic Unit

MAU Modular Unit

NTRU Nth Degree Truncated Polynomial Ring Unit

1 INTRODUCTION

1.1 Motivation

The rapid development of internet and technology have provided vast areas of new opportunities and potential sources of efficiency for individuals and organizations of all sizes. Cybersecurity, which refers to the technologies and processes designed to protect computers, networks and data from unauthorized access, vulnerabilities and attacks, becomes a critical issue for people. Cryptography provides the core technology for cybersecurity. There are two types of cryptographic systems available, symmetrical key system and asymmetrical key system.

Asymmetrical key system (also called public key system) is a class of cryptographic algorithms which requires two separate keys: a public key to encrypt messages and a private key to decrypt messages. Asymmetrical key systems, unlike symmetric key systems, do not require a secure communication channel for the initial exchange of one (or more) secret keys between the parties and thus they can provide very important and unique security services such as key distribution and digital signature.

RSA and elliptic curve cryptosystem (ECC) are two currently popular public key systems in modern cryptography. All these asymmetrical key systems are based on some hard mathematical problems. In another word, breaking an asymmetrical system is mathematically equivalent to solving a certain hard problem. For instance, integer factorization and elliptic curve discrete logarithm are the underlying hard problems for RSA and elliptic curve cryptosystems, respectively. These cryptography systems are considered to be secure today since no efficient algorithm exists to solve their underlying hard problems.

However, this situation will change with emerging of quantum computing technology. Quantum computers are very different from electronics based computers and they make use of quantum-mechanical phenomena. Quantum computers are highly

efficient in solving many hard mathematical problems. For example, Shor [4] find efficient quantum algorithm for solving integer factorization and discrete logarithm, which indicates that both RSA and elliptic curve cryptosystems will not be secure anymore with availability of quantum computers. Therefore there is a need for cryptographic technologies that are still secure in the age of quantum computers. Studies of these cryptographic technologies are referred as post-quantum cryptography [5].

The current research in the area of post-quantum cryptography is mainly focused on four approaches. These approaches and some typical cryptography systems are listed as follows:

- Code-based cryptography
  - McEliece encryption [6] and Niederreiter signature [7]
- Hash-based cryptography
  - Lamport signature [8] and Merkle signature [9]
- Multivariate cryptography
  - Rainbow signature scheme [10]
- Lattice-based cryptography
  - Learning with Errors [11] and NTRU [12]

Among the four post-quantum cryptography sub-areas, lattice based cryptography relies on the worst-case hardness of lattice problems, it can provide strong security guarantees, efficient implementations and great simplicity. One well known example of lattice based systems is NTRU, which is probably the most practical system among all the existing post-quantum cryptosystems.

NTRU was firstly developed by Hoffstein, Pipher and Silverman from Brown University in 1996 [12]. It includes two well known schemes. NTRUEncrypt is used for

encryption and NTRUSign is used for digital signature. NTRU has been adopted by IEEE P1363.1 standards under the specifications for lattice-based public-key cryptography since 2009 [1]. In 2009, D. Stehlé, *et al.* introduced a revised NTRUEncrypt scheme [13] which is a provably secure variant NTRUEncrypt. The revised scheme is likely to be less efficient than NTRUEncrypt, but it can provide higher security since it is established under lattice problem in its worst case.

Recently, with the development of cloud computing, there is a requirement on security that allows computations to be carried out on encryption data, while preserving the privacy of the message. This encryption scheme is called homomorphic encryption(HE). Homomorphic encryption can be divided into two types, partially homomorphic encryption which allows only addition or multiplication operation and fully homomorphic encryption(FHE) which supports arbitrary computation on ciphertexts. The first FHE is proposed by Craig Gentry [14] [15] [16]. It is based on the worst-case problems over ideal lattices. Several optimizations and implementation were proposed in [17] [18] [19] [20] and [21]. But the large size of ciphertext and the efficiency bottleneck limit the develop of FHE. More efficient new schemes were developed. See for example [22] [23] [24] and [25]. Most of these schemes are based on the hardness of the learning with errors (LWE) problem [11] which is as hard to solve as several worst-case lattice problems. However, in 2012, López-Alt *et al.* [26] found some fully homomorphic properties in the revised NTRUEncrypt scheme and proposed a fully homomorphic scheme based on NTRU system. The scheme is appears to be efficient. More recently, a modify scheme is proposed to make the security of this NTRU based FHE depends only on standard lattice assumptions [27]. Several implementations on these NTRU based FHE were proposed in [28] [29] and [30]. Because of these new found properties, NTRU system attract the public attention again.

1.2 Our Contribution

In this thesis, we propose some efficient hardware architectures for both original NTRUEncrypt system and revised NTRUEncrypt system which can be extended to FHE. More specifically we

- proposed an linear feedback shift register (LFSR) architecture to implement both encryption and decryption of NTRUEncrypt system.
- proposed an extended LFSR architecture which take advantage of large number of zero coefficients of the input. A parameter selection optimization for hardware is also introduced.
- presented a systolic array architecture for efficient implementation of NTRU-Encrypt.
- propose two new architectures with one LFSR based and the other systolic array based for implementation of NTRU based fully homomorphic encryption system.

1.3 Dissertation Organization

An organization of the rest of this thesis is as follows. Chapter 2 presents some mathematical background and gives a brief introduction of NTRUEncrypt algorithm and revised NTRUEncrypt algorithm. A brief overview of previous works is shown in Chapter 3. Chapter 4 presents a new hardware architecture based on LFSR for NTRUEncrypt with FPGA implementation result. Comparison between our work and previous works is also given. Chapter 5 describes an extend LFSR structure improve to efficient of our design. An FPGA implementation result will be also presented. Chapter 6 proposed a new hardware architecture based on systolic array. Chapter 7 gives a brief overview of NTRU based fully homomorphic encryption system

and presents two architectures to implement the polynomial multiplication for FHE with NTRU based systems. Finally, the last chapter concludes the paper and discusses about the future work.

2 MATHEMATICAL PRIMITIVES

This chapter introduces relevant mathematical background related to the NTRU and NTRU based systems that include definition of lattices as well as the hard problems in lattice that has a significant place in cryptography, and the definition of truncated polynomial ring and its operations. The encryption and decryption algorithms of NTRU system and revised NTRU system will also be covered in this chapter.

2.1 Lattices

Lattices have been extensively studied recently by cryptographers for quite some time, in both the field of cryptanalysis and a source of hard problems to build encryption schemes. We begin with some definitions and brief discussion of lattices.

2.1.1 Definitions

Let the real vector space be denoted by $\mathbb{R}^n$. A lattice L is a discrete subset of $\mathbb{R}^n$. Every lattice in $\mathbb{R}^n$ can be generated from a basis for the vector space by forming all linear combinations with integer coefficients.

Let $v_1, v_2, \ldots, v_n$ be linearly independent vectors in $\mathbb{R}^n$. Then the vectors $v_1, v_2, \ldots, v_n$ can be viewed as basis of lattice L. The set of lattice L can be expressed as an integer linear combination of basis vectors as follows:

$$L = \{a_1v_1 + a_2v_2 + ... + a_nv_n | a_i \in \mathbb{Z}\}$$

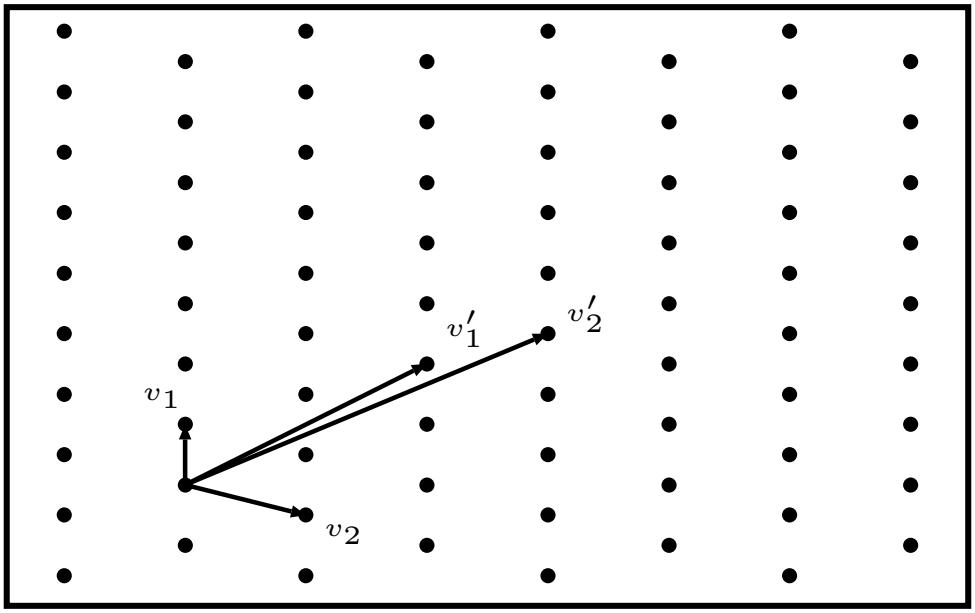


Fig. 2.1: A Two-dimensional Lattice

A lattice has many bases. Two possible bases, $\{v_1, v_2\}$ and $\{v_1', v_2'\}$, for a two dimensional lattice are shown in Fig.2.1. A lattice is integral if it is contained in n-dimensional integer vector space $\mathbb{Z}^n$ and it is called rational if it is contained in $\mathbb{Q}^n$, where $\mathbb{Q}$ denote the set of rational numbers [1]. $\mathbb{Z}^n$ is a simple example of a lattice in $\mathbb{R}^n$.

2.1.2 Ideal Lattice

Ideal lattice has important applications place in cryptography. It is a new class of lattice that includes cyclic lattices as a special case. In general terms, ideal lattices are lattices corresponding to ideals in rings of the form $\mathbb{Z}[x]/\langle f \rangle$ for some irreducible polynomial $f(x)$ of degree n [31]. Cyclic lattices are a special case of ideal lattices where $f(x) = x^n - 1$ [32].

2.1.3 Hard Problem in Lattices

Shortest vector problem(SVP) and closest vector problem (CVP) are two fundamental hard problems in lattices. The general CVP is known to be NP-hard[1] and the SVP is NP-hard under a randomized reduction hypothesis.

The CVP is to find a lattice point closest to the given target point with respect to a lattice basis. It is the underlying hard problem for NTRUEncrypt system. Encryption is a matter of selecting a target point. Decryption is a matter of mapping the target point back to the closest lattice point.

2.2 Truncated Polynomial Ring with Modular Arithmetic

Ntru system is based on the hardness of problems for lattices that can be represented as ideals in the ring $R = \mathbb{Z}[x]/(x^n - 1)$. That means the basic NTRU operations take place in the ring of convolution polynomials $R = \mathbb{Z}[x]/(x^n - 1)$, where n is prime. That is why it is called NTRU, short for Nth Degree Truncated Polynomial Ring Unit. The set of degree $n - 1$ truncated polynomial can be denoted by $R = \mathbb{Z}[x]/(x^n - 1)$, where $\mathbb{Z}[x]$ is the set of polynomials with integer coefficients and taken modulo $x^n - 1$. A polynomial a(x) $\in$ R can be expressed as

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

Let $b(x) = b_{n-1}x^{n-1} + ... + b_0$ be another truncated polynomial in R. Addition operation $a(x) + b(x)$ can be performed by simply adding their corresponding coefficients.

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

Multiplication is a little complex. The product of two polynomials is given by

[1]Class of problems which are at least as hard as the hardest problems in NP.

$$a(x) \times b(x) = c_{n-1}x^{n-1} + ... + c_1 x + c_0 \tag{3}$$

where

$$c_k = \sum_{\substack{i,j=0,1,...,n-1 \\ i+j=k \mod n}} a_i b_j, k = 0, 1, ..., n-1 \tag{4}$$

Note that NTRU works in the truncated polynomial ring modulo q, means every coefficients in the polynomial should modulo q. It is denoted by $R_q = \mathbb{Z}_q[x]/(x^n - 1)$.

2.3 NTRUEncrypt Public Key Cryptosystem

In this chapter, we give a brief overview of NTRUEncrypt, including parameter selection, key generation, encryption and decryption operation.

2.3.1 Parameter Selection

Integers n, p and q should be chosen to set up the NTRUEncrypt. n is a prime number to define the degree of truncated polynomial. q and p need to be relatively prime and q should be considerably larger than p. Integers d_f,d_g and d_r should be selected to determine three sets of $n-1$ degree polynomials L_f, L_g and L_r. We will use $L(d_1, d_2)$ to describe these sets, which means the polynomials in L has d_1 coefficients equal to 1, d_2 coefficients equal to -1 and the rest coefficients equal to 0. We set

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

A parameter set generation algorithm is presented in [3] and shown in Algorithm 2.1. It described a machine which takes as input a desired security level k and outputs a parameter set that gives k bits of security. The function $hybridSecurityEstimate(n, d_f)$ in Algorithm 2.1 is used to estimate the minimum security over all attack strategies

on that parameter set. The chance of a decryption failure is given by the function $decryptionFailureProb(n, d_f)$.

Algorithm 2.1 Parameter Generation Function for NTRUEncrypt [3]

Input: Security Level, k
Output: Parameter Set (n, d_f)

$i \leftarrow 1$ {The variable i is used to index the set of acceptable primes $\mathcal{P}$}
$i* \leftarrow 0$ {This will become the first index which can achieve the required security}
repeat
  $n \leftarrow \mathcal{P}_i$
  $d_f \leftarrow [n/3]$ {We will try each d_f from $[n/3]$ down to 1}
  repeat
    $k_1 \leftarrow$ hybridSecurityEstimate(n,d_f)
    $k_2 \leftarrow log_2$(decryptionFailureProb(n,d_f))
    if ($k_1 \geq k$ and $k_2 < -k$) **then**
      (i^*,d_f^*) $\leftarrow (i, d_f)$ {Record the first acceptable index i and the value of d_f}
    end if
    $d_f \leftarrow d_f - 1$
  until $i^* > 0$ or $d_f < 1$
  $i \leftarrow i + 1$
until $i^* > 0$
$c^* \leftarrow cost(\mathcal{P}_{i^*}, d_f^*)$
while an increase in N can potentially lower the cost **do**
  $n \leftarrow \mathcal{P}_i$
  $d_f \leftarrow d_f^*$ {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}
  repeat
    $k_1 \leftarrow$ hybridSecurityEstimate(n,d_f)
    $c \leftarrow cost(n, d_f)$
    if ($k_1 \geq k$ and $c < c^*$) **then**
      (c^*,i^*,d_f^*) $\leftarrow (c, i, d_f)$ {Record the the improvement in cost and the corresponding i, d_f}
    end if
    $d_f \leftarrow d_f - 1$
  until $d_f < 0$
  $i \leftarrow i + 1$
end while
return ($\mathcal{P}_{i^*}$,d_f^*)

Some parameter sets are listed in Table 2.1 provide by [3]. There are three parameter sets for each security levels using three different cost metrics, including optimized

size $cost_{size} = n \cdot log_2 q$, optimized trade-off $cost_{trade-off} = cost_{size}^2 \times cost_{speed}$ and optimized speed $cost_{speed} = n \cdot d_f$.

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

The parameter q is usually in the form of 2^n for its computational advantages. As q and p need to be relatively prime, 3 is the smallest number usually selected as the value of p. n is the parameter determining the security level of the system.

2.3.2 Key Generation

The key generation for NTRUEncrypt is given below:

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

Step 2: Randomly choose a polynomial $g(x)$ from 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 $h(x) = p \cdot f_q(x) \times g(x) \mod q$.

$h(x)$ is the public key, while the pair $(f(x),\ f_p(x))$ is the corresponding private key of the system.

2.3.3 Encryption

Step 1: Encode message to ternary polynomial $m(x)$.

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

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

2.3.4 Decryption

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

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

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

2.3.5 Why the Decryption of NTRUEncrypt Works

$$\begin{aligned} f(x) = a(x) &= f(x) \times e(x) \mod q \\ &= f(x) \times [r(x) \times pf_q(x) \times g(x) + m(x)] \mod q \\ &= p \times r(x) \times g(x) + f(x) \times m(x) \mod q \\ &= f_p(x) \times [p \times r(x) \times g(x) + f(x) \times m(x)] \mod q \\ &= m(x) \end{aligned} \tag{5}$$

2.3.6 Discussions on NTRUEncrypt Optimization

Invertibility of f: If $f(x) = 1 + p \cdot f'(x)$, $f(x)$ is always invertible modulo p, $f(x) \times f_p(x) = 1 \mod p$, so the step in decryption $m(x) = f_p(x) \times a(x) \mod p$ can be skipped.

Taking p to be a polynomial: We may take p to be a small polynomial so that it will be more natural to use binary polynomials.

Low Hamming Weight Product: There is an algorithm which can factorize the polynomial into two polynomials, like $f(x) = f_1(x) \times f_2(x)$ $(d_f < d_{f_1} \cdot d_{f_2})$. $f(x)$ can be further rewritten as $f(x) = 1 + p \cdot (f_1(x) \times f_2(x) + f_3(x))$ which can also avoid the short lattice vector attacks.

2.4 Revised NTRUEncrypt Public Key Cryptosystem

In [13], D.Stehlé, *et al.* introduced a revised NTRUEncrypt scheme which is a provably secure variant. Some modifications are made to the original NTRUEncrypt scheme in order to make it hard to solve a well established lattice problem in its worst case.

1. $R = \mathbb{Z}[x]/(x^n + 1)$ instead of $R = \mathbb{Z}[x]/(x^n - 1)$.
2. n choose as a power of 2, q is a prime integer and p select as 2.
3. A small error $s(x)$ is added during the encryption, $e(x) = h(x) \times r(x) + p \cdot e(x) + m(x) \mod q$.

In the follow sections, we give a brief overview of this revised NTRUEncrypt scheme, including parameter selection, key generation, encryption and decryption operation.

2.4.1 Parameter Selection

In [2], D.Cabarcas *et al.* showed how to select parameters that provide an expected security level. Other than n, p and q, σ and ψ should be also chosen to set up the revised NTRUEncrypt scheme. Some example sets of parameter value is shown in Table 2.2.

Step 1: Select n to be a power of two.

Step 2: Choose a prime $q \in [d \cdot n^6 \ln(n), 2d \cdot n^6 \ln(n)]$, such that $q \equiv 1 \mod 2n$ and $d = 25830$ guarantees correctness of the scheme.

Step 3: Set $p = 2$, so that the message can be encoded in binary form.

Step 4: Set $\sigma = 2n\sqrt{\ln(8n \cdot q) \cdot q}$, it is the standard deviation for $D_{\mathbb{Z}^n,\sigma}$.

Step 5: Set $\psi = \sqrt{2n \cdot \pi}$,it is the standard deviation for $D_{\mathbb{Z}^n,\psi}$.

Table 2.2: Parameters for Revised NTRUEncrypt [2]

Security Level	n	$\log q$	$\log \sigma$	ψ
38	1024	71.90	49.89	25.53
144	2048	77.28	53.63	36.11
338	4096	83.30	57.70	51.06

2.4.2 Key Generation

Step 1: Sample a polynomial $f'(x)$ from $D_{\mathbb{Z}^n,\sigma}$, and let $f(x) = p \cdot f'(x) + 1$ that is invertible in R_q.

Step 2: Sample a polynomial $g(x)$ from $D_{\mathbb{Z}^n,\sigma}$.

Step 3: Calculate $h(x) = p \cdot f_q(x) \times g(x) \mod q$.

$h(x)$ is the public key, while the pair $f(x)$ is the corresponding private key of the system.

2.4.3 Encryption

Step 1: Encode message to binary polynomial $m(x)$.

Step 2: Sample polynomials $r(x)$ and $s(x)$ from $D_{\mathbb{Z}^n,\psi}$.

Step 3: Encrypt message $e(x) = h(x) \times r(x) + p \cdot s(x) + m(x) \mod q$.

2.4.4 Decryption

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

Step 2: Compute $m(x) = a(x) \mod p$.

3 A REVIEW OF EXISTING WORK

NTRUEncrypt mostly has been implemented in software. A limited amount of literature has been published to provide hardware implementation of NTRUEncrypt systems. The existing NTRUEncrypt architcure can be grouped into two types. Some of them tried to reduce power consumption [33] [34] and some of them tried to improve efficiency [35] [36] [37]. Our work will mainly focus on providing an efficient hardware implementation, and we will review some existing works on efficient implementation of NTRUEncrypt system first.

The earliest published research that implemented NTRU system in hardware was in 2001 [35]. D. Bailey and five others implemented NTRUEncrypt in constrained devices using embedded system and FPGA technique. They designed a system to realize NTRU encrypt engine and applied it on Xilinx's Virtex 1000EFG860 FPGA. The parameters he used for NTRUEncrypt was $(N, p, q) = (251, x + 2, 128)$. The advantage of using these parameters is that the operands $r(x)$ and $m(x)$ are binary polynomials which can simplify NTRU's encryption procedure in hardware. While some extra steps are need in decryption which is difficult to implement in hardware. But his design showed that NTRU cryptosystem was exremely beneficial for hardware implementation due to its low complexity and parallel nature.

O'Rourke presented an algorithm of NTRU polynomial multiplication and a corresponding flexible architecture in her thesis [36]. The polynomial multiplier can be applied in the NTRU's key creation, encryption and decryption procedures. The design takes the advantage of the parallel nature of the partial product array. Since a partial product can be processed by a processing unit independently, arbitrary number of processing unit can be used to make the design scalable. The high flexibility of their work is the critical contribution to the study of this field.

A.Kamal and A.Youssef proposed a high-speed FPGA implementation of NTRUEncrypt in [37]. They took advantage of "small" polynomial which include large

number of zero elements in the polynomial. In order to skip the zero coefficient in the polynomial, an n-bit shifter is design by using multiplexors to implement a small number of shifts. They also use statistical properties of the distance between the non-zero elements to find a proper n to offer different area-speed trade off.

4 PROPOSED LFSR ARCHITECTURE TO IMPLEMENT NTRUENCRYPT

In this chapter, we will introduce an architecture to implement NTRUEncrypt algorithm with FPGA implementation and our work and the previous work. Since the generation of public key and private key can be precomputed, we will only focus on speeding up the encryption and decryption process in hardware implementation. Truncated polynomial multiplication is probably the most time consuming operation during encryption and decryption. The efficiency of the system is determined by the speed of the truncated polynomial multiplication.

4.1 Linear Feedback Shift Register

An LFSR is a shift register whose feedback value is a linear function of its previous state. It has well known applications in pseudo-random numbers, cyclic redundancy check and cryptography. An LFSR shown in Fig.4.1 is determined with its characteristic polynomial $f(x)$, which is in the form of

$$f(x) = x^n + f_{n-1}x_{n-1} + ... + f_1x + 1 \tag{6}$$

Note in Fig.4.1 that a $\oplus$ refers to an adder, a $\otimes$ refers to a multiplier and a $\square$ stands for a register. Assume that the registers are loaded with the coefficients of polynomial $A(x) = (a_0, a_1, ..., a_{n-1})$. A shift-to-right operation of LFSR is equivalent to performing $A(x) \times x \mod f(x)$, where x is the root of its characteristic polynomial $f(x)$.

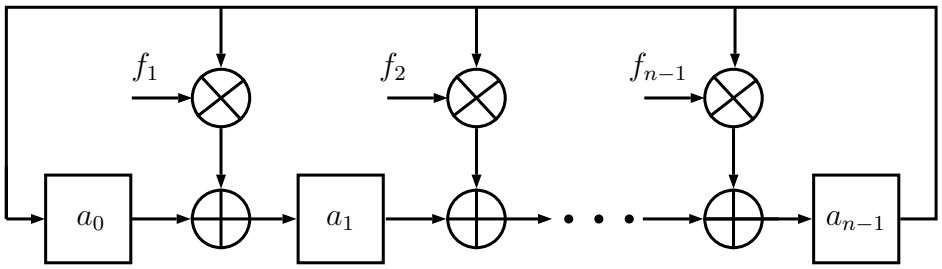


Fig. 4.1: Linear Feedback Shift Register

4.2 Proposed Truncated Polynomial Ring Multiplier

Since the polynomial in truncated polynomial ring requires to take modulo $x^n - 1$, LFSR can be used under truncated polynomial ring.

Let the content of registers $e_0, e_1, ..., e_{n-1}$ at clock cycle j be denoted as $e_0^{(j)}, e_1^{(j)}, ..., e_{n-1}^{(j)}$, respectively. The step to perform truncated polynomial ring multiplication in hardware is presented in Algorithm 4.1.

Algorithm 4.1 Multiplication in Truncated Polynomial Ring

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

We made some modifications to LFSR. An architecture diagram of the proposed multiplier in truncated polynomial ring is shown in Fig.4.2. It includes n multipliers, n adders and n registers. Each register can store $[log_2 q]$ bits. The operations in NTRUEncrypt are incorporate with modular arithmetic. Since q is in the form of 2^n, modulo q can be easily achieved by truncated the result to n bits. All the operations can be performed without carry.

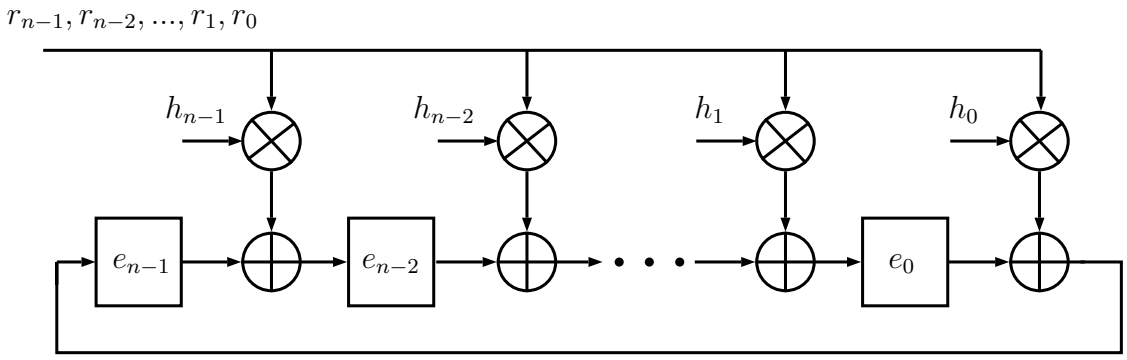


Fig. 4.2: Proposed Truncated Polynomial Ring Multiplier

Coefficients of operand $h(x) = (h_0, h_1, ..., h_{n-1})$ input to each multiplier in parallel, while coefficients of operand $r(x) = (r_0, r_1, ..., r_{n-1})$ input to all the multipliers in a serial fashion. The registers $e(x) = (e_0, e_1, ..., e_{n-1})$ are initially loaded with 0. The registers will store the product $h(x) \times r(x) \mod (x^n - 1)$ after n clock cycles.

4.3 Proposed Encryption Architecture

4.3.1 Proposed Modular Arithmetic Unit

During each clock cycle, multiplication is probably the most time consuming operation in our proposed architecture . As $r(x)$ is always chosen as a ternary polynomial, r_j always has value from $\{-1, 0, 1\}$. This operation can actually be evaluated without any multiplications. Furthermore, if $r_j = -1$, subtraction operation $e_i = e_{i+1} - h_{i+1} \mod q$ $(q = 2^n)$ can be evaluated by

$$e_i = e_{i+1} + \overline{h_{i+1}} + 1 \tag{7}$$

As a result, these operations can be simply calculated with only addition operation incorporate with modular arithmetic. A proposed Modular Arithmetic Unit (MAU) is designed to optimize our design (Fig.4.3).

Table 4.1: Operations Supported with the Modular Arithmetic Unit

Input r $(r_{(1)}r_{(0)})$	Output s
01	$e + h$
11	$e + \overline{h} + 1$
00	e

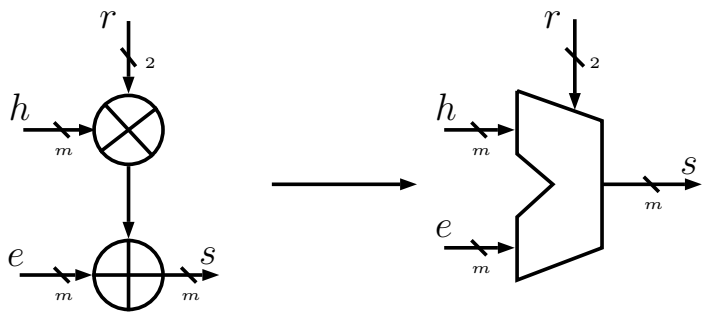


Fig. 4.3: Modular Arithmetic Unit

The modulus of MAU is q. The input h, e and output s are encoded in ternary with $m = [log_2 q]$ bits. The input r, encoded in two bits $r_{(1)}r_{(0)}$, acts as the control input. The step to perform operation in hardware is presented in Algorithm 4.2. The corresponding MAU architecture, which is responsible for performing the operation in Table 4.1, is shown in Fig.4.4.

Algorithm 4.2 Proposed Modular Arithmetic Unit

Input: $e : e_{(m-1)}, ..., e_{(1)}, e_{(0)}$ $h : h_{(m-1)}, ..., h_{(1)}, h_{(0)}$ $r : r_{(1)}, r_{(0)}$
Output: $s : s_{(m-1)}, ..., s_{(1)}, s_{(0)}$
1: **if** $r_{(0)} = 0$ **then**
2: $(s_{(m-1)}, ..., s_{(1)}, s_{(0)}) = (e_{(m-1)}, ..., e_{(1)}, e_{(0)})$
3: **else** $\{r_{(0)} = 1\}$
4: $(h_{(m-1)}, ..., h_{(1)}, h_{(0)}) = (h_{(m-1)} \oplus r_{(1)}), ..., (h_{(1)} \oplus r_{(1)}), (h_{(0)} \oplus r_{(1)})$;
5: $(s_{(m-1)}, ..., s_{(1)}, s_{(0)}) = (e_{(m-1)}, ..., e_{(1)}, e_{(0)}) + (h_{m-1)}, ...h_{(1)}, h_{(0)}) + r_{(1)}$;
6: **end if**
$^*x_{(i)}$ represtent ith bit of x;

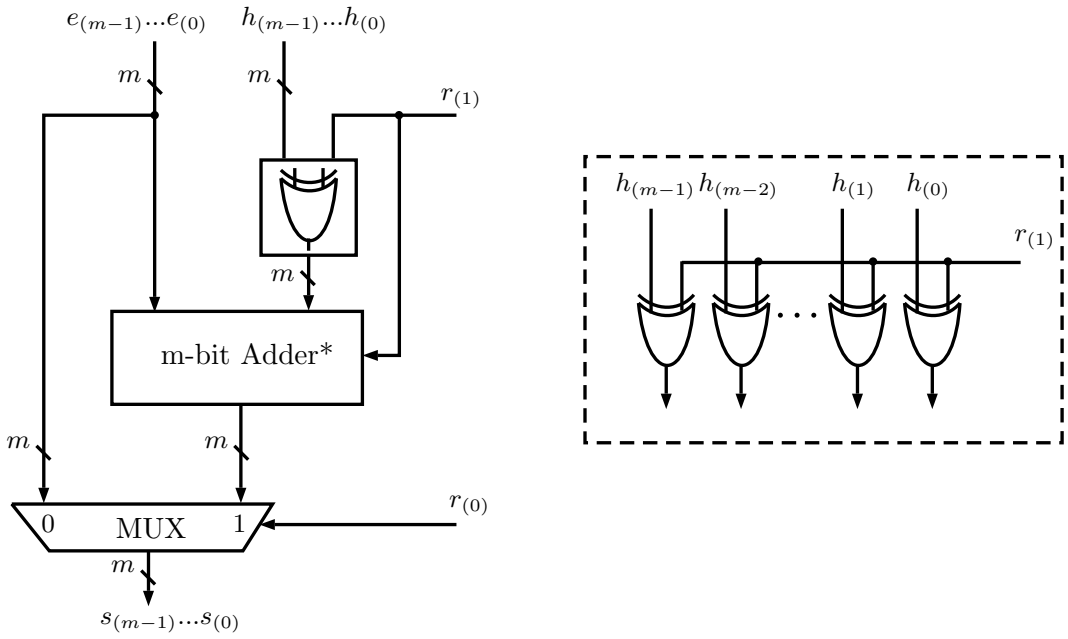


Fig. 4.4: Proposed Modular Arithmetic Unit

4.3.2 Proposed Encryption Architecture

Operations in encryption:

$$e(x) = h(x) \times r(x) + m(x) \mod q \tag{8}$$

Since the random polynomial $r(x)$ has coefficients from {-1,0,1}, the proposed truncated polynomial ring multiplier and modular arithmetic unit can support the full operation during encryption. The addition of the message $m(x)$ can be perform by initial the registers e with coefficients of $m(x)$. The architecture for encryption is shown in Fig.4.5. which contains n registers and n MAUs.

Algorithm 4.3 LFSR Based NTRUEncrypt (Encryption)

Input: $m = m_0, ..., m_{n-1}$, $h = h_0, ..., h_{n-1}$, $r = r_0, ..., r_{n-1}$

Output: $e = e_0, ..., e_{n-1}$

1: $e^{(0)} = m$
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** $e = e^{(b)}$

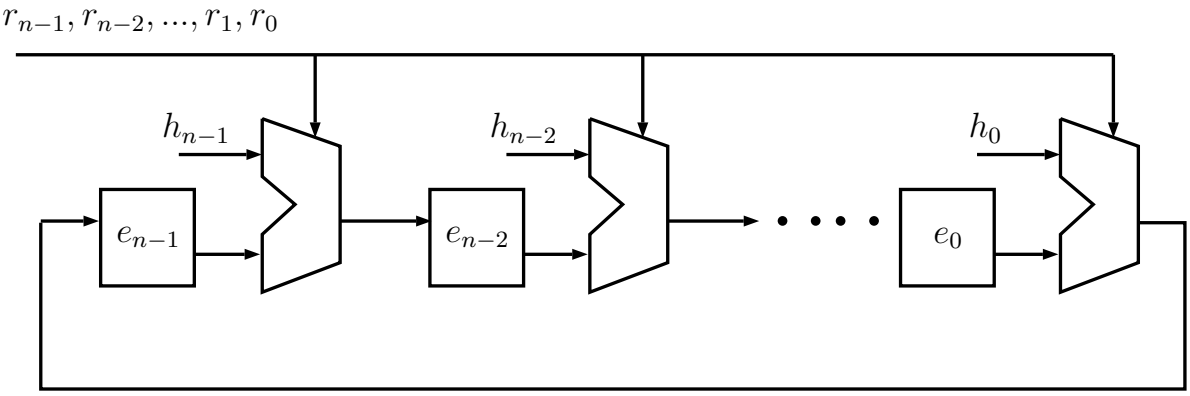


Fig. 4.5: LFSR Based NTRUEncrypt (Encryption)

The registers $e = (e_{n-1}, ..., e_1, e_0)$ are initially loaded with $m = (m_{n-1}, ..., m_1, m_0)$ in one clock cycle. After n clock cycles, the registers will store the encryption result $e = (e_{n-1}, ..., e_1, e_0)$. The content of registers at cycle j, $j = 0, 1, ..., n$, is given in Table 4.2.

Table 4.2: Register Contents for LFSR Based NTRUEncrypt (Encryption)

Cycle j	Input r_j	$e_{n-1}^{(j)}$	$e_{n-2}^{(j)}$	...	$e_0^{(j)}$
0	-	m_{n-1}	m_{n-2}	...	m_0
1	r_0	$m_0 + h_0 r_0$	$m_{n-1} + h_{n-1} r_0$	...	$m_1 + h_1 r_0$
2	r_1	$m_1 + h_1 r_0 + h_0 r_1$	$m_0 + h_0 r_0 + h_{n-1} r_1$	...	$m_2 + h_2 r_0 + h_1 r_1$
...	...	...	...	...	...
n	r_{n-1}	$m_{n-1} + h_{n-1} r_0 + ... + h_0 r_{n-1}$	$m_{n-2} + h_{n-2} r_0 + ... + h_{n-1} r_{n-1}$	...	$m_0 + h_0 r_0 + ... + h_1 r_{n-1}$
$e = m + r * h$		$= e_{n-1}^{(n)}$	$= e_{n-2}^{(n)}$	...	$= e_0^{(n)}$

4.4 Proposed Decryption Architecture

We made some adjustment to the algorithm step, so that the Truncated polynomial ring multiplier can be used within decryption process in hardware. The operations in decryption are listed as follows:

- $a(x) = f(x) \times e(x) \times f_p(x) \mod q$.
- Shift coefficients of $a(x)$ to the range $(-\frac{q}{2}, \frac{q}{2})$
- $b(x) = a(x) \mod p$
- $m(x) = f_p(x) \times b(x) \mod p$.

Both the operand $f(x)$ in the first and the operand $b(x)$ in the last step has coefficients from {-1,0,1}. As a result, the proposed truncated polynomial ring multiplier and modular arithmetic unit can support most of the operations during decryption. Besides, a design to shift coefficients of a(x) and a modulo a(x) by p is require to complete the decryption operation. A proposed Modular Unit (MU) is designed performance these operations.

4.4.1 Proposed Modular Unit

The coefficients are required to shift from $(0, q-1)$ to $(-\frac{q}{2}, \frac{q}{2})$. In binary form, it can be performed easily by inverting the digits and add one to the result.

Algorithm 4.4 Shift Coefficient Operation

Input: a_i
Output: a'_i
1: **if** $a_i > \frac{q}{2}$ **then**
2: $\quad a'_i = -(\overline{a_i} + 1)$
3: **else**
4: $\quad a'_i = a_i$
5: **end if**

Since p is usually chosen as 3 which is a Mersenne number, we can use the Mersenne primes algorithm to calculate $a(x) \mod p$. And we assume that the maximum operand is 1023.

Algorithm 4.5 Modulo 3 Operation

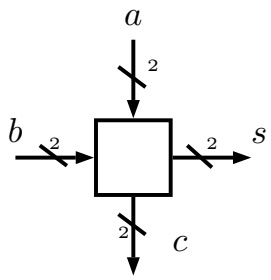
Input: $a : a_{(n-1)}, ..., a_{(1)}, a_{(0)}$, $p = 3$
Output: $m = a \mod 3$
1: split a into $|a_{(n-1)}a_{(n-2)}|a_{(n-3)}a_{(n-4)}|...|a_{(3)}, a_{(2)}|a_{(1)}, a_{(0)}|$
2: b = $a_{(n-1)}a_{(n-2)} + a_{(n-3)}a_{(n-4)} + ... + a_{(3)}a_{(2)} + a_{(1)}a_{(0)}$
3: c = $b_{(3)}b_{(2)} + b_{(1)}b_{(0)}$
4: m = $c_{(2)} + c_{(1)}c_{(0)}$
*$x_{(i)}$ represent ith bit of x;

Since the coefficients of a(x) are shifted to $(-\frac{q}{2}, \frac{q}{2})$, modulo of a negative number is also required during decryption. A look up table is used to perform both positive and negative modulo operation and output the result in ternary form {-1,0,1}.

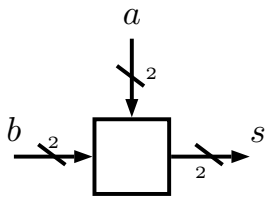
Table 4.3: Look Up Table for Modular Unit

$m_{(2)}$	$m_{(1)}$	$m_{(0)}$	$s_{(1)}s_{(0)}$
0	0	0	00
0	0	1	01
0	1	0	11
0	1	1	00
1	0	0	00
1	0	1	11
1	1	0	01
1	1	1	00

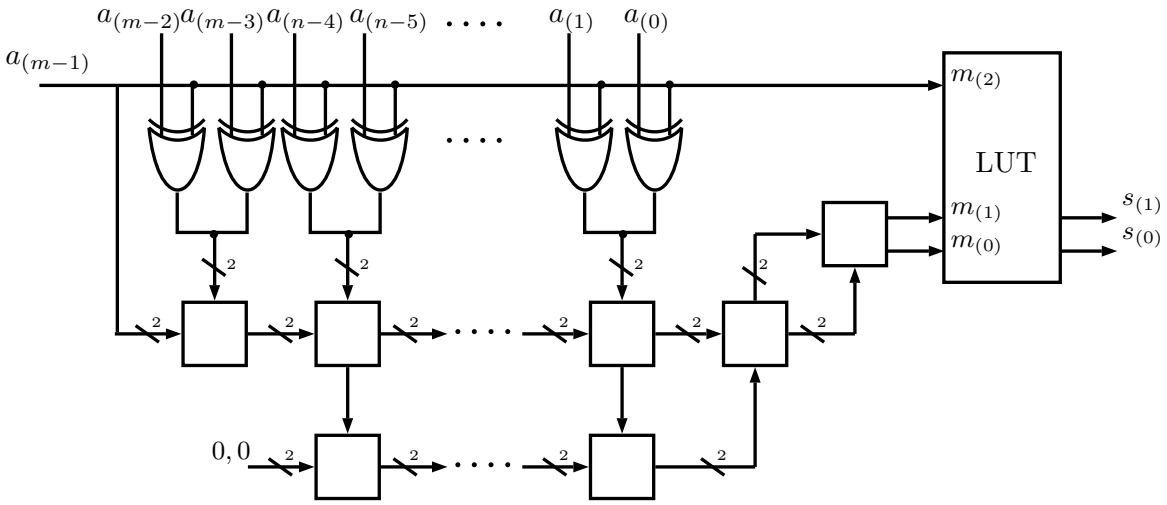
A circuit to perform both coefficients shift operation and modulo 3 operation is shown in Fig.4.6. It consists of number of 2-bit full adders and number of 2-bit half adders. The amount of the units can be varied according to the bit number of the coefficient.



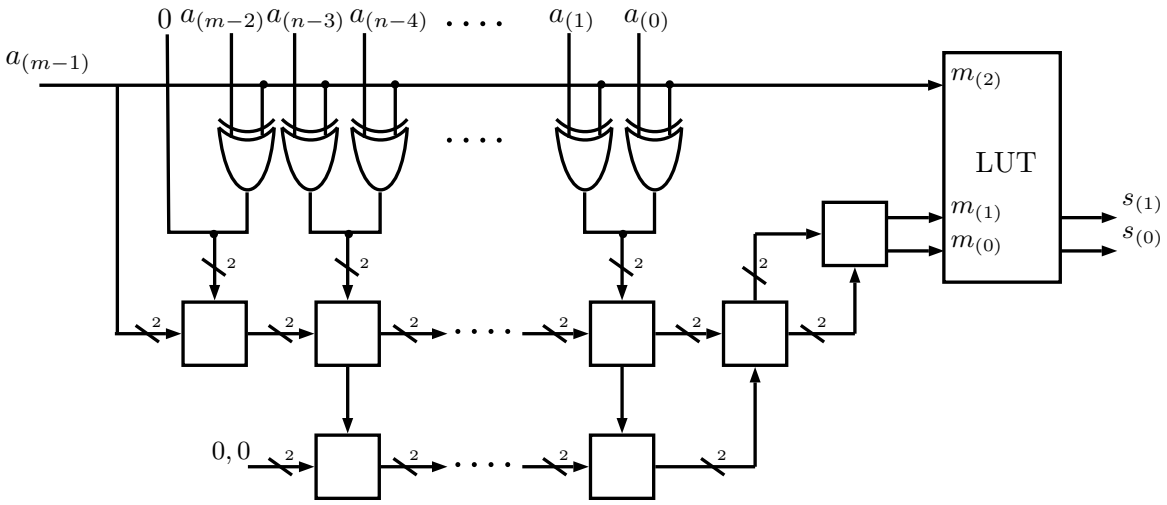
(a) 2-Bit Full Adder



(b) 2-Bit Half Adder



(c) m is an odd number



(d) m is an even number

Fig. 4.6: Proposed Modular Unit

4.4.2 Proposed Decryption Architecture

The architecture for encryption is shown in Fig.4.7. which contains an n-bit truncated polynomial ring multiplier and n modular unit.

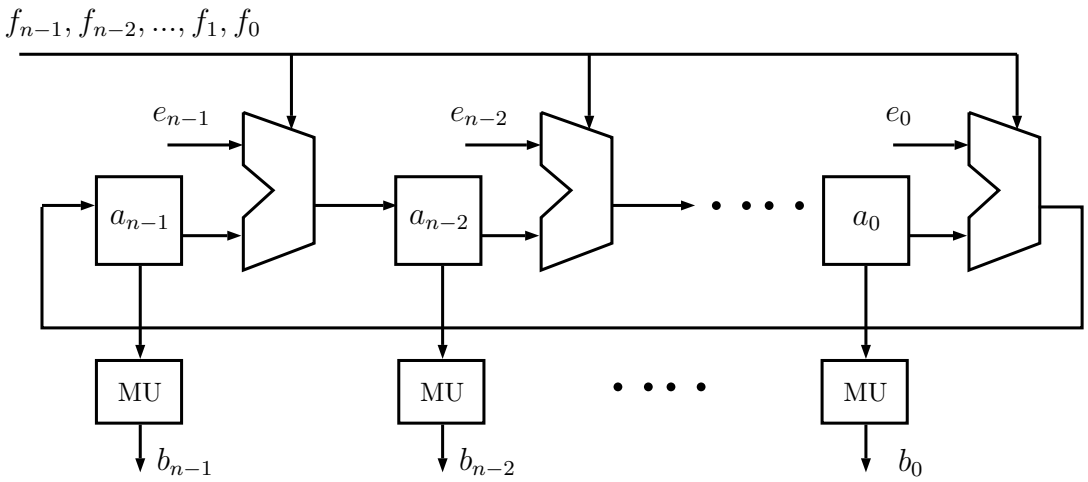


Fig. 4.7: LFSR Based NTRUEncrypt (Decryption)

The registers $a = (a_{n-1}, ..., a_1, a_0)$ are initially loaded with 0. It require the circuit to run twice to complete the decryption procedure. The first round can evaluate $b(x)$ with input $f(x)$ and $e(x)$. Then we can recover message by executing the circuit one more time with input $f_p(x)$ and $b(x)$.

4.5 Implementation of Proposed Encryption Architecture

The proposed encryption architecture has been implemented in FPGA. The following tools were used for the implementation.

- Quartus II Web Edition Software
- ModelSim-Altera software

Cyclone IV EP4CE115F23C7 was chosen as the target device in provided implementation.

4.5.1 Implementation Result

The post-synthesis simulation results are shown in Table 4.4 with parameter set $ees401ep1$. The encryption speed is fast, in the range of microsecond while the area consumption remains reasonable.

Table 4.4: Simulation Result for LFSR Based NTRUEncrypt (Encryption)

Resource		Latency	
#Logic Element	18,049	#Clock Cycle	252
#Registers	3,526	Max Frequency	143.0Mhz
		Total Latency	1.76 μs

When different sets of parameters are applied to our proposed design, the result is still satisfactory. The simulation result for different parameter sets are shown in Table 4.5.

Table 4.5: Simulation Result with Different Parameter Sets

Security Level	Parameter set	#LE(S)	#RE	#CC	FMax	Latency(T)
112	$ees401ep1$	18,049	8,826	402	98.86 MHz	4.06 μs
	$ees541ep1$	24,349	11,906	542	100.7 MHz	5.38 μs
	$ees659ep1$	29,659	14,502	660	90.46 MHz	7.29 μs
128	$ees449ep1$	20,209	9,882	450	103.44 MHz	4.35 μs
	$ees613ep1$	27,589	13,490	614	94.23 MHz	6.51 μs
	$ees761ep1$	34,249	16,746	762	89.29 MHz	8.53 μs
192	$ees677ep1$	30,469	14,898	678	89.37 MHz	7.58 μs
	$ees887ep1$	39,919	19,518	888	86.77 MHz	10.23 μs
	$ees1087ep1$	48,919	23,918	1088	72.42 MHz	15.02 μs
256	$ees1087ep2$	48,919	23,918	1088	72.42 MHz	15.02 μs
	$ees1171ep1$	52,699	25,766	1172	62.31 MHz	18.80 μs
	$ees1499ep1$	67,460	32,982	1500	72.86 MHz	20.58 μs

4.5.2 Comparison

Our proposed design, along with some other existing works, is implemented using the same version of FPGA technology with parameter set $ees401ep1$ and the results are shown in Table 4.6.

- #LE, number of logic elements used by the systems.
- #CC, number of clock cycles required by the systems.
- FMax, the maximum operating frequency of the system.
- Delay, the time requires to encrypt a polynomial.
- S×T, the area-delay product of the systems.

It can be seen from the table that the proposed one has clear advantage over most works. For example, compared to [35], the proposed one has much higher maximum frequency and utilizes much shorter time to perform encryption. In comparison of [36], it has lower resource consumption and higher frequency compared to our design, however it needs much more clock cycles. It is noted that the number of clock cycles for [37] (= 113) is optimized for the condition that the maximum distance between two none-zero coefficient is no more than 8. Nonetheless, the maximum frequency of [37] is much lower than our design and the resources it needs are nearly 3 times larger.

Table 4.6: Comparison of Proposed LFSR Architecture with Other Exist Works

For $ees401ep1$ parameter set					
Work	#LE(S)	#CC	FMax	Latency(T)	$S \times T$
[35]	14,807	402	49.41 MHz	8.13 μs	274.4%
[36]	3,782	3,618	169.49 MHz	21.34 μs	110.1%
[37]	49,001	113	39.91 MHz	2.83 μs	189.2%
Proposed	18,049	402	98.86 MHz	4.06 μs	100.0%

The simulation results in Table 4.6 show that the proposed design outperforms all the existing works in terms of area-delay product. The new architecture is expected to be used for a system require both high speed with limit resource.

5 PROPOSED EXTENDED LFSR ARCHITECTURE TO IMPLEMENT NTRUENCRYPT

5.1 General Ideal

The proposed MAU in the last chapter shows that when the control input r, which is encoded by $r_{(1)}r_{(0)}$, equals to "00" or "10", the delay required by MAU is T_{mux}. When $r_{(1)}r_{(0)}$ equals to "01" or "11", required time equals to $T_{mux}+T_{add}+T_{xor}$. Thus, the required time of each clock cycle varies and depends on the input r.

The coefficients of $r(x)$ always have value from $\{-1, 0, 1\}$, which means the input bits $r_{(1)}r_{(0)}$ only have three states "11", "00" and "01". The state "10" is considered as a redundant state for MAU.

Meanwhile, $r(x) \in L(d_r, d_r)$ and d_r is usually much smaller than n. It is required to find out how many pairs of "0, 0" coefficients appear consecutively in r(x) to calculate number of cycles the system can save.

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

$$n_1 = n_{-1} = d_r. \tag{9}$$

So the number of "0" is

$$n_0 = n - 2d_r. \tag{10}$$

The maximum number of "0, 0" pair happen when all of the "0" coefficient appear consecutively. The number is calculated by

$$n_{00_{max}} = (n - 2d_r)/2. \tag{11}$$

The minimum number of "0, 0" pair happens when all the none-zero coefficients

are assigned discontinuously, which is given by

$$n_{00_{min}} = \begin{cases} 0 & n_1 + n_{-1} \geq n_0 \\ (n - 4d_r)/2 & n_1 + n_{-1} < n_0 \end{cases} \tag{12}$$

The average number of "0, 0" pair in $r(x)$ is

$$n_{00_{avg}} = \sum_{i=n_{00_{min}}}^{n_{00_{max}}} \left(\frac{{}^{n_1+n_{-1}}P_{n_1+n_{-1}}}{{}^{n_1}P_{n_1} \cdot {}^{n_{-1}}P_{n_{-1}}} \cdot {}^{n_1+n_{-1}+1}C_{n_0-1} \cdot \sum_{j=0}^{i} {}^{n_1+n_{-1}+1}C_{j+1} \cdot {}^{i-1}C_j\right) \tag{13}$$

For example, in the parameter set $ees659ep1$, the value of n is 659, while d_r is equal to 38. It could be calculated that the number of non-zero coefficients in polynomial $r(x)$ is 76, while the number of zero coefficients is 583. There are a large number of zero coefficients contained in polynomial $r(x)$ and many of these zero coefficients appear consecutively. We can find at least 254 pairs of "$0, 0$" in $r(x)$. And the average number of "$0, 0$" pair can reach to 273.42. Table 5.3 shows the number of "$0, 0$" pairs in the polynomial for other parameter sets.

Table 5.1: Number of "0, 0" Pair in Different Parameter Set

Security Level	Parameter set	n	#"0"	#"0, 0"		
				Avg	Min	Max
112	$ees401ep1$	401	176	52.99	0	88
	$ees541ep1$	541	444	199.21	173	222
	$ees659ep1$	659	584	273.42	254	292
128	$ees449ep1$	449	182	51.83	0	91
	$ees613ep1$	613	504	226.48	197	252
	$ees761ep1$	761	680	318.49	299	340
192	$ees677ep1$	677	364	126.51	25	182
	$ees887ep1$	887	726	325.84	282	141
	$ees1087ep1$	1087	962	450.70	418	481
256	$ees1087ep2$	1087	848	370.72	304	424
	$ees1171ep1$	1171	960	431.54	374	480
	$ees1499ep1$	1499	1342	633.68	592	671

A considerable reduction in the number of clock cycle can be achieved if a "0, 0"

pair can be processed in one clock cycle. The proposed method is designed to reduce the total processing time by deal with a "0, 0" pair within one clock cycle.

5.2 Extended Linear Feedback Shift Register

In order to handle two zero coefficients in one clock cycle, an extended LFSR is required to perform shifts for both one and two positions. An extended LFSR was proposed by Wu *et al.* [38].

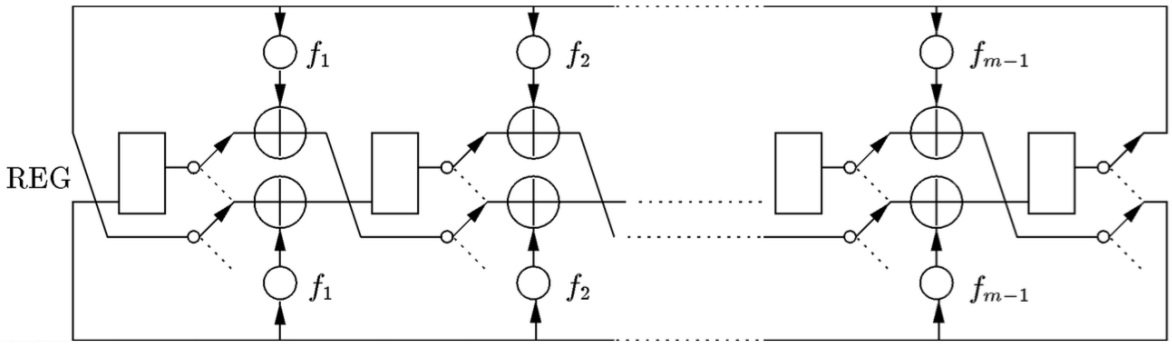


Fig. 5.1: Extended Linear Feedback Shift Register

Fig.5.1 shows the structure for both $x^{\pm 1}$ and $x^{\pm 2}$ multiplication, which is called extended LFSR. When the switches are at upper positions, the circuit are configured to perform $x^{\pm 2}$ multiplication, which means the contents of the registers are shifted two positions on their right side. When the switches are at lower positions, the circuit, which works as a normal LFSR, is shifted one position in each clock cycle. This circuit can be integrated in our design with minor modification.

5.3 Proposed Extended Modular Arithmetic Unit

Instead of switches, input bits r can be used to control the circuit switch between one position shift and two positions shift. We extend MAU to support this operation (Fig.5.2).

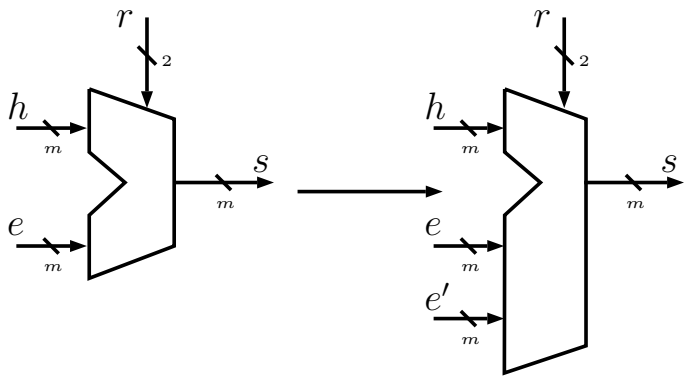


Fig. 5.2: Extended Modular Arithmetic Unit

Table 5.2: Operations Supported with the Extended Modular Arithmetic Unit

Input r $(r_{(1)}r_{(0)})$	Output s
01	$e + h$
11	$e + \overline{h} + 1$
00	e
10	e'

As compared to previous circuit, an additional input e' is added to MAU. It outputs directly when input r is "10". The truth table of this circuit is given in table 5.2. As stated before, "10" was a redundant state in previous circuit. In proposed circuit, this state is assigned to handle the operation for output e'. Four states of proposed circuit and their output is shown. The details for performing these operations in hardware is presented in Algorithm 5.1. The inputs of the algorithm is e, e', h, r and the output is s. Depending on the control signal r, different operations are performed.

Algorithm 5.1 Proposed Extended Modular Arithmetic Unit

Input:

$e : e_{(m-1)}, ..., e_{(1)}, e_{(0)}$;
$e' : e'_{(m-1)}, ..., e'_{(1)}, e'_{(0)}$;
$h : h_{(m-1)}, ..., h_{(1)}, h_{(0)}$
$r : r_{(1)}, r_{(0)}$;

Output: $s : s_{(m-1)}, ..., s_{(1)}, s_{(0)}$;

1: **if** $r_{(0)} = 0$ **then**
2: **if** $r_{(1)} = 0$ **then**
3: $(s_{(m-1)}, ..., s_{(1)}, s_{(0)}) = (e_{(m-1)}, ..., e_{(1)}, e_{(0)})$
4: **else** $\{r_{(1)} = 1\}$
5: $(s_{(m-1)}, ..., s_{(1)}, s_{(0)}) = (e'_{(m-1)}, ..., e'_{(1)}, e'_{(0)})$
6: **end if**
7: **else** $\{r_{(0)} = 1\}$
8: $(h_{(m-1)}, ..., h_{(1)}, h_{(0)}) = (h_{m-1} \oplus r_{(1)}), ..., (h_{(1)} \oplus r_{(1)}), (h_{(0)} \oplus r_{(1)})$;
9: $(s_{(m-1)}, ..., s_{(1)}, s_{(0)}) = (e_{(m-1)}, ..., e_{(1)}, e_{(0)}) + (h_{(m-1)}, ...h_{(1)}, h_{(0)}) + r_{(1)}$;
10: **end if**

$^*x_{(i)}$ represtent ith bit of x;

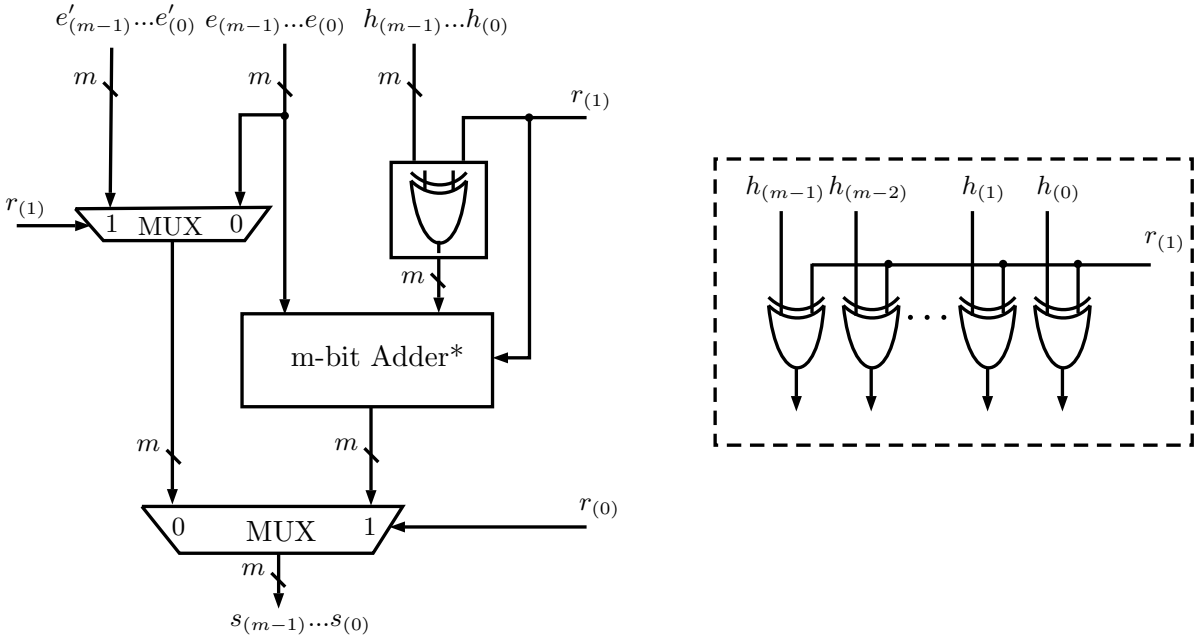


Fig. 5.3: Proposed Extended Modular Arithmetic Unit

The corresponding Extended Modular Arithmetic Unit (EMAU) is shown in Fig.5.3. An additional multiplexer is applied in this modular arithmetic unit. The time complexity becomes $2T_{mux}$ when $r_1 r_0$ equals to "00" or "10".

5.4 Proposed NTRUEncrypt Architecture with Extended LFSR

As presented in previous chapter, operations in encryption is defined as:

$$e(x) = h(x) \times r(x) + m(x) \mod q \tag{14}$$

Since the new design can process two consecutive zero coefficient at the same time, minor modification on input r is required. When two "0" coefficient appear consecutively in $r(x)$, it should input "10" to the circuit instead of two "00". n' is the coefficient number of $r(x)$ after replace all the "0, 0" pairs with "10'. Algorithm 5.2 shows a typical algorithm to implement ecnryption with EMAU. A corresponding structure is shown in Fig.5.4.

Algorithm 5.2 Extended Encryption in NTRUEncrypt

Input:
$m = m_0, ..., m_{n-1}$;
$h = h_0, ..., h_{n-1}$;
$r = r_0, ..., r_{n'-1}$;

Output: $e = e_0, ..., e_{n-1}$;

1: $e^{(0)} = m$
2: **for** $j = 1$ to n' **do**
3: **if** $r_{j-1} = 10$ **then**
4: **for** $i = 0$ to $n - 1$ **do**
5: $e_i^{(j)} = e_{i+2 \mod n}^{(j-1)}$
6: **end for**
7: **else**
8: **for** $i = 0$ to $n - 1$ **do**
9: $e_i^{(j)} = e_{i+1 \mod n}^{(j-1)} + h_{i+1 \mod n} \times r_{j-1} \mod q$
10: **end for**
11: **end if**
12: **end for**
13: **return** $e = e^{(n)}$

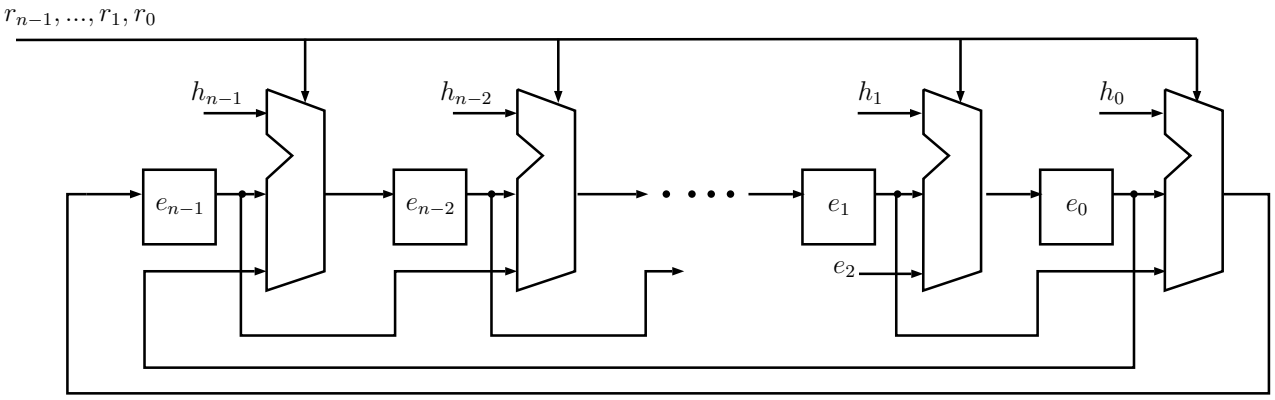


Fig. 5.4: Extended LFSR Based NTRUEncrypt

The registers $e = (e_{n-1}, ..., e_1, e_0)$ are initially loaded with $m = (m_{n-1}, ..., m_1, m_0)$ in one clock cycle. The clock cycles required to encrypt a message depends on the number of "0, 0" pairs in $r(x)$. The average clock cycle required in encryption for different parameter sets is shown in Table 5.3. It can be seen that number of clock cycle $\#CC$ is reduced from original required clock cycle n. With a higher security level, the required clock cycles also increased. More reduced clock cycle can be seen on higher security level, which means proposed circuit shows more improvements on more complex parameter set. The content of registers at cycle j, $j = 0, 1, ..., n'$, is given in Table 5.4.

Table 5.3: Average Clock Cycle for Different Parameter Sets

Security Level	Parameter set	n	p	q	d_r	#CC
112	$ees401ep1$	401	3	2048	113	350
	$ees541ep1$	541	3	2048	49	343
	$ees659ep1$	659	3	2048	38	387
128	$ees449ep1$	449	3	2048	134	359
	$ees613ep1$	613	3	2048	55	388
	$ees761ep1$	761	3	2048	42	444
192	$ees677ep1$	677	3	2048	157	552
	$ees887ep1$	887	3	2048	81	563
	$ees1087ep1$	1087	3	2048	63	638
256	$ees1087ep2$	1087	3	2048	120	718
	$ees1171ep1$	1171	3	2048	106	741
	$ees1499ep1$	1499	3	2048	79	867

Table 5.4: Register Contents for Extended LFSR Based NTRUEncrypt

Cycle j	Input r_j	$e_{n-1}^{(j)}$	$e_{n-2}^{(j)}$	...	$e_0^{(j)}$
0	-	m_{n-1}	m_{n-2}	...	m_0
1	r_0	$m_0 + h_0 r_0$	$m_{n-1} + h_{n-1} r_0$	...	$m_1 + h_1 r_0$
2	r_1	$m_1 + h_1 r_0 + h_0 r_1$	$m_0 + h_0 r_0 + h_{n-1} r_1$	...	$m_2 + h_2 r_0 + h_1 r_1$
3	r_2 ="10"	$m_3 + h_3 r_0 + h_2 r_1 + h_1 r_2 + h_0 r_3$	$m_2 + h_2 r_0 + h_1 r_1 + h_0 r_2 + h_{n-1} r_3$	...	$m_4 + h_4 r_0 + h_3 r_1 + h_2 r_2 + h_1 r_3$
...	...	...	...	...	...
n'	$r_{n'-1}$	$m_{n-1} + h_{n-1} r_0 + ... + h_0 r_{n'-1}$	$m_{n-2} + h_{n-2} r_0 + ... + h_{n-1} r_{n'-1}$	...	$m_0 + h_0 r_0 + ... + h_1 r_{n'-1}$
$e = m + r * h$		$= e_{n-1}^{(n')}$	$= e_{n-2}^{(n')}$	...	$= e_0^{(n')}$

5.5 Implementation of Proposed Encryption Architecture

The proposed architecture also has been implemented in FPGA for performance verification. The following tools were used for the implementation.

- Quartus II Web Edition Software
- ModelSim-Altera software

Cyclone IV EP4CE115F23C7 was chosen as the target device in provided implementation.

5.5.1 Implementation Results

The simulation results with various parameter sets are shown in Table 5.5. It can be seen from the table that with the increasing of security level, the operation frequency of the circuit gradually decreases in an acceptable range. More logic elements and registers are also required.

Table 5.5: Simulation Result with different Parameter Set (Extended LFSR)

Security Level	Parameter set	#LE(S)	#RE	#CC	FMax	Latency(T)
112	$ees401ep1$	22,460	8,826	350	97.36 MHz	3.58 μs
	$ees541ep1$	30,300	11,906	343	94.58 MHz	3.62 μs
	$ees659ep1$	36,908	14,502	387	81.12 MHz	4.77 μs
128	$ees449ep1$	25,148	9,882	359	92.29 MHz	3.88 μs
	$ees613ep1$	34,332	13,490	388	85.02 MHz	4.56 μs
	$ees761ep1$	42,620	16,746	444	75.36 MHz	5.89 μs
192	$ees677ep1$	37,916	14,898	552	89.18 MHz	6.18 μs
	$ees887ep1$	49,676	19,518	563	87.42 MHz	6.44 μs
	$ees1087ep1$	60,876	23,918	638	73.71 MHz	8.65 μs
256	$ees1087ep2$	60,876	23,918	718	73.71 MHz	9.74 μs
	$ees1171ep1$	65,581	25,766	741	84.07 MHz	8.81 μs
	$ees1499ep1$	83,949	32,982	867	63.64 MHz	13.62 μs

5.5.2 Comparison

Table 5.6: Comparison between LFSR Structure and Extended LFSR Structure

Security Level	Parameter set	LFSR		Extended LFSR	
		#LE(S)	Latency(T)	#LE(S)	Latency(T)
112	*ees*401*ep*1	18,049	4.06 μs	22,460 (24.4%)	3.58 μs (11.8%)
	*ees*541*ep*1	24,349	5.38 μs	30,300 (24.4%)	3.62 μs (32.7%)
	*ees*659*ep*1	29,659	7.29 μs	36,908 (24.4%)	4.77 μs (34.5%)
128	*ees*449*ep*1	20,209	4.35 μs	25,148 (24.4%)	3.88 μs (10.8%)
	*ees*613*ep*1	27,589	6.51 μs	34,332 (24.4%)	4.56 μs (29.9%)
	ees761ep1	34,249	8.53 μs	42,620 (24.4%)	5.89 μs (30.9%)
192	*ees*677*ep*1	30,469	7.58 μs	37,916 (24.4%)	6.18 μs (18.4%)
	*ees*887*ep*1	39,919	10.23 μs	49,676 (24.4%)	6.44 μs (37.0%)
	*ees*1087*ep*1	48,919	15.02 μs	60,876 (24.4%)	8.65 μs (42.4%)
256	*ees*1087*ep*2	48,919	15.02 μs	60,876 (24.4%)	9.74 μs (35.1%)
	*ees*1171*ep*1	52,699	18.80 μs	65,581 (24.4%)	8.81 μs (53.1%)
	*ees*1499*ep*1	67,460	20.58 μs	83,949 (24.4%)	13.62 μs(33.8%)

A comparison between the previous LFSR structure and the extended LFSR structure is listed in Table 5.6. The logic elements required by the extended LFSR structure are 24.4% larger than the LFSR structure. Because of the reduction on clock cycle, the extended LFSR structure can save at least 10.8% time with 128 security level and ees449ep1 parameter set. The improvement can be up to 53.1% with 256 security level and ees1171ep1 parameter set. The simulation results in Table 4.6 show that the proposed design outperforms all the existing works in terms of area-delay product. The extended architecture is expected to be used for a system with higher speed requirement.

5.6 Discussion on Parameter Selection in Hardware Implementation

A parameter set generation algorithm (Section 2.3.1) is required to generate parameter set according to different cost metrics. To obtain the best performance in hardware efficiency, the cost metric should make some modifications. The original speed metric

is derived from the convolution times for a parameter set $cost_{speed} = n \cdot d_f$. But for hardware implementation in my previous work, a clock cycle is always required to handle a zero-coefficient or a non-zero coefficient. So that at previous work, the value of speed cost function of hardware is n^2. For the proposed work in this chapter, as stated before, consecutive "0" only requires one clock cycle. Thus, it can be derived that the cost function becomes

$$cost_{speed} = n \cdot (n - n_{00}), \tag{15}$$

which is from the original cost function reduced by number of consecutive "0, 0". Referring to (12), the worst case of number of n_{00} is $(n - 4d_r)/2$. Substituting the value of n_{00} to (15), we get

$$cost_{speed} = \begin{cases} n^2 & n_1 + n_{-1} \geq n_0 \\ n \cdot (n/2 + 2d_f) & n_1 + n_{-1} < n_0 \end{cases}. \tag{16}$$

Through this cost function, the speed optimization for our design can be performed.

6 PROPOSED SYSTOLIC ARRAY ARCHITECTURE TO IMPLEMENT NTRUENCRYPT

6.1 Systolic Array

A systolic system is a network of processors which rhythmically compute and pass data through the system. It is first proposed by H.T. Kung and C.E. Leserson in 1979 [39]. Physiologists use the work 'systole' to refer to the rhythmically recurrent contraction of the heart and arteries which pulses blood through the body. In a systolic computing system, the function of a processor is analogous to that of the heart. Every processor regularly pumps data in and out, each time performing some short computation, so that a regular flow of data is kept up in the network. Systolic architecture can result in cost-effective , high-performance special-purpose systems for a wide range of problems.

Systolic systems consists of an array of processing elements (PE) processors are called cells, each cell is connected to a small number of nearest neighbours in a mesh like topology. Each cell performs a sequence of operations on data that flows between them. Generally the operations will be the same in each cell, each cell performs an operation or small number of operations on a data item and then passes it to its neighbour. Systolic array is a computing network possessing include the following property [40]:

- **The design makes multiple use of each input data item**

  Because of this property , systolic systems can achieve high throughputs with modest I/O bandwidths for outside communication.

- **The design uses extensive concurrency**

  Concurrency can be obtained by pipelining the stages involved in the computation of each single result , by multiprocessing many results in parallel, or by

both.

- **There are only a few types of simple cells**

  To achieve performance goals, a systolic system is likely to use a large number of cells which must be simple and of only a few types to curtail design and implementation cost.

- **Data and control flow are simple and regular**

  Pure systolic system totally avoid long-distance or irregular wires for data communication.

Systolic arrays can be used to build many cost-effective, high-performance special-purpose system, such as matrix multiplication, polynomial evaluation, convolution and image processing.

6.2 Algorithm

As we mention in the previous chapter, operations in encryption is defined as:

$$e(x) = m(x) + h(x) \times r(x) = e_{n-1}x^{n-1} + ... + e_1x + e_0$$

where

$$e_k = m_k + \sum_{\substack{i,j=0,1,...,n-1 \\ i+j=k \mod n}} h_i r_j,\ k = 0, 1, ..., n-1$$

Since all polynomials are taken modulo $x^n - 1$, the degree exceed N-1 should be truncated, which means the power x^n should be replaced by 1, the power x^{n+1} should be replaced by x, and so on. Also because of each partial product term is reduced modulo q, there is no carry propagation across the column. In Fig.6.1, $e(x)$, $m(x)$, $h(x)$ and $r(x)$ are represented as coefficient vectors. It shows the parallel nature of encryption operations in NTRUEncrypt.

	h_{n-1}	h_{n-2}	h_{n-3}	$\cdots$	h_2	h_1	h_0
$\times$	r_{n-1}	r_{n-2}	r_{n-3}	$\cdots$	r_2	r_1	r_0
$+$	m_{n-1}	m_{n-2}	m_{n-3}	$\cdots$	m_2	m_1	m_0
	m_{n-1}	m_{n-2}	m_{n-3}	$\cdots$	m_2	m_1	m_0
	$h_{n-1}r_0$	$h_{n-2}r_0$	$h_{n-3}r_0$	$\cdots$	h_2r_0	h_1r_0	h_0r_0
	$h_{n-2}r_1$	$h_{n-3}r_1$	$h_{n-4}r_1$	$\cdots$	h_1r_1	h_0r_1	$h_{n-1}r_1$
	$h_{n-3}r_2$	$h_{n-4}r_2$	$h_{n-5}r_2$	$\cdots$	h_0r_2	$h_{n-1}r_2$	$h_{n-2}r_2$
	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
	h_2r_{n-3}	h_1r_{n-3}	h_0r_{n-3}	$\cdots$	h_5r_{n-3}	h_4r_{n-3}	h_3r_{n-3}
	h_1r_{n-2}	h_0r_{n-2}	$h_{n-1}r_{n-2}$	$\cdots$	h_4r_{n-2}	h_3r_{n-2}	h_2r_{n-2}
$+$	h_0r_{n-1}	$h_{n-1}r_{n-1}$	$h_{n-2}r_{n-1}$	$\cdots$	h_3r_{n-1}	h_2r_{n-1}	h_1r_{n-1}
	e_{n-1}	e_{n-2}	e_{n-3}	$\cdots$	e_2	e_1	e_0

Fig. 6.1: Encryption Operation in Vertical Form

6.3 Proposed NTRUEncrypt Architecture with Systolic Array

In Fig.6.1, you can see that beside the first row, every cell of the array is a multiplication operation. The coefficients e_i can be calculated by accumulate the product of each cell in each column and add m_i. This parallel nature property makes this algorithm ideally suitable for use in systolic array architecture.

Also because of each column is independent of the other, the same result can be obtain if we only change the order of the elements of each column. We make some modification to Fig.6.1 to make it perfect fit the systolic array architecture (Fig.6.2).

	m_{n-1}	m_{n-3}	$\cdots$	m_0	m_{n-2}	$\cdots$	m_3	m_1
	$h_{n-1}r_0$	$h_{n-2}r_{n-1}$	$\cdots$	$h_{(n-1)/2}r_{(n-1)/2+1}$	$h_{(n-1)/2-1}r_{(n-1)/2}$	$\cdots$	h_1r_2	h_0r_1
	$h_{n-2}r_1$	$h_{n-3}r_0$	$\cdots$	$h_{(n-1)/2-1}r_{(n-1)/2+2}$	$h_{(n-1)/2-2}r_{(n-1)/2+1}$	$\cdots$	h_0r_3	$h_{n-1}r_2$
	$h_{n-3}r_2$	$h_{n-4}r_1$	$\cdots$	$h_{(n-1)/2-2}r_{(n-1)/2+3}$	$h_{(n-1)/2-3}r_{(n-1)/2+2}$	$\cdots$	$h_{n-1}r_4$	$h_{n-2}r_3$
	$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$
	h_2r_{n-3}	h_1r_{n-4}	$\cdots$	$h_{(n-1)/2+3}r_{(n-1)/2-2}$	$h_{(n-1)/2+2}r_{(n-1)/2-3}$	$\cdots$	h_4r_{n-1}	h_3r_{n-2}
	h_1r_{n-2}	h_0r_{n-3}	$\cdots$	$h_{(n-1)/2+2}r_{(n-1)/2-1}$	$h_{(n-1)/2+1}r_{(n-1)/2-2}$	$\cdots$	h_3r_0	h_2r_{n-1}
$+$	h_0r_{n-1}	$h_{n-1}r_{n-2}$	$\cdots$	$h_{(n-1)/2+1}r_{(n-1)/2}$	$h_{(n-1)/2}r_{(n-1)/2-1}$	$\cdots$	h_2r_1	h_1r_0
	e_{n-1}	e_{n-3}	$\cdots$	e_0	e_{n-2}	$\cdots$	e_3	e_1

Fig. 6.2: Modified Encryption Operation in Vertical Form

Since n is an odd number ($n > 2$ and n is prime number), the order of columns is reordered as $e_{n-1}, e_{n-3}, ..., e_0, e_{n-2}, ..., e_3, e_1$. And the order of elements in each column is moved circularly from top to bottom according to the new position of the column.

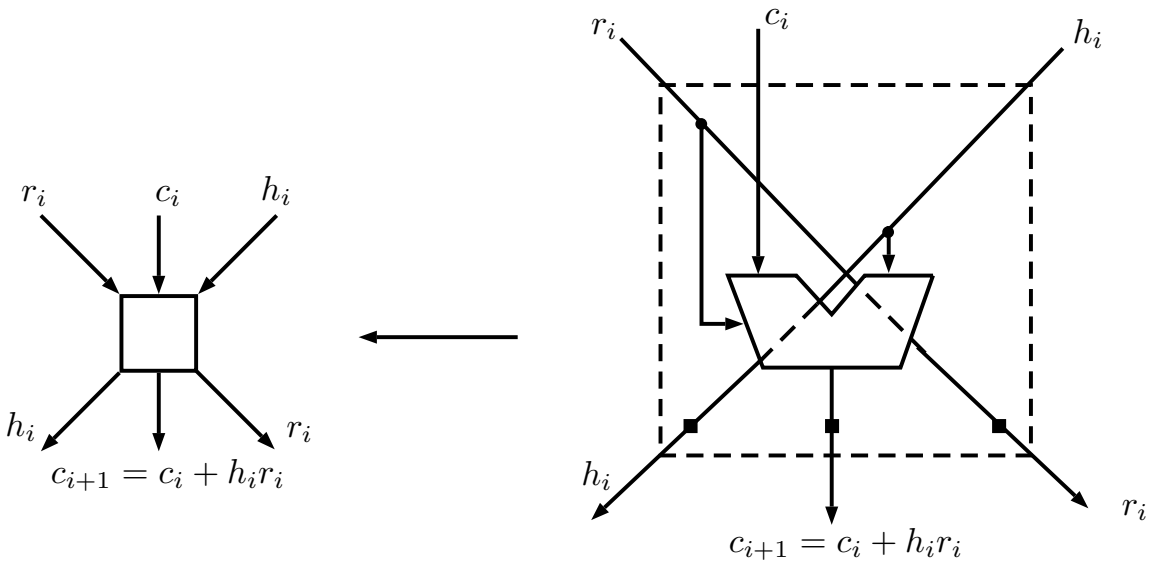


Fig. 6.3: Processing Element

A processing element is designed with a MAU, which proposed in chapter 4, shown in Fig.6.3. c_i,h_i and r_i are input of the PE. h_i and r_i are output directly, while the other output is calculate through the MAU and output c_{i+1} which is the sum of c_i and product $h_i r_i$. A ■ represents a m-bits (m is equal $\log_2 q$) latch. With the PE, a systolic array architecture for encryption is designed and shown in Fig.6.4.

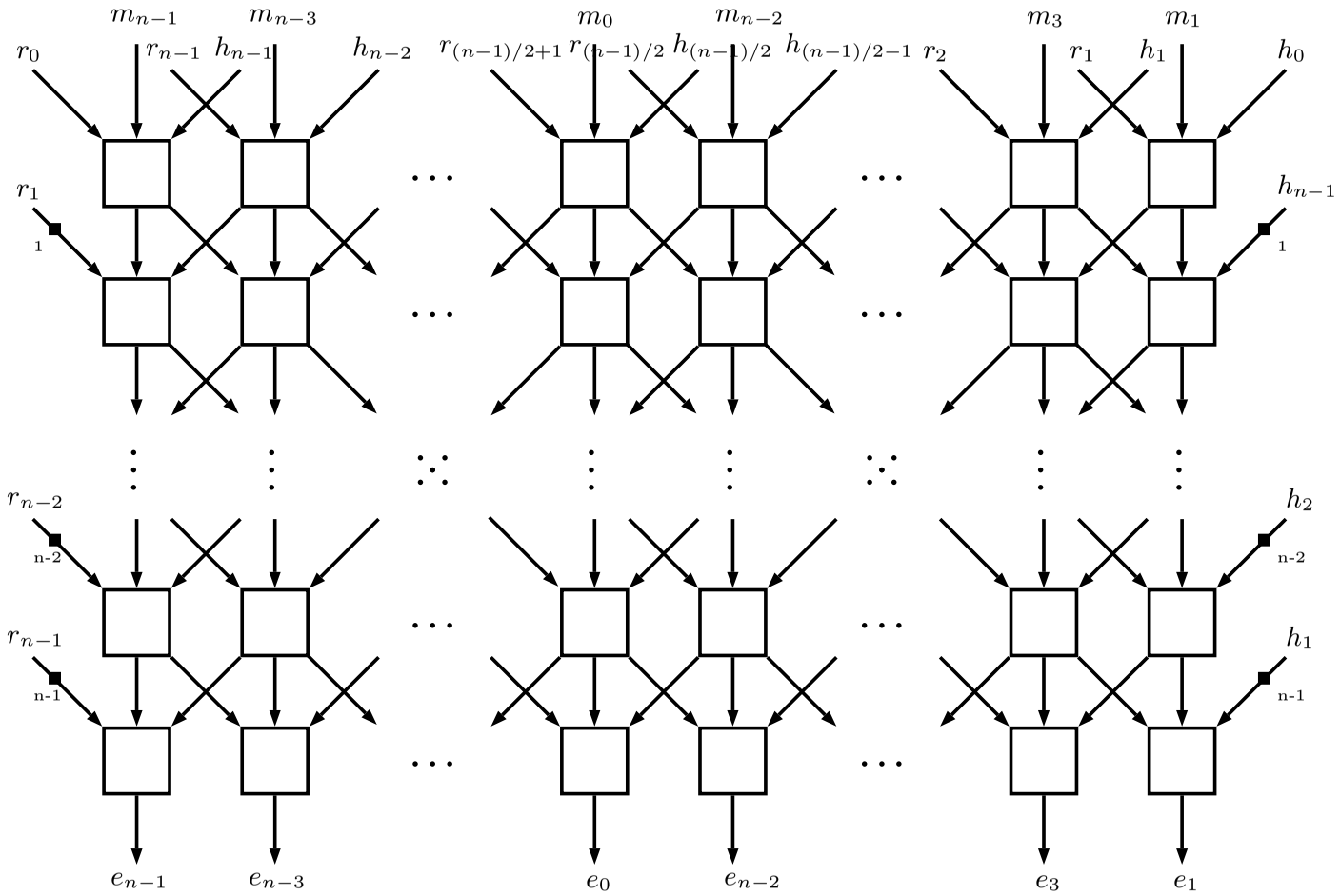


Fig. 6.4: Systolic Array Based NTRUEncrypt

The systolic array base architecture consist of n^2 PE. Every PE is connected to its neighbour PEs. The coefficients of $h(x)$ are input from the top edge and right edge of the systolic array and transmits diagonally from upper right to lower left while the coefficient $r(x)$ are input from the top edge and left edge of the systolic array and transmits diagonally from upper left to lower right. The coefficients of $m(x)$ are initially input to the first row of the systolic array and vertically propagate the result to the next PE below. The numbers beside ■ denote the amount of latches. It takes n clock cycle to evaluate every coefficient of e(x).

7 NTRU APPLICATION IN CLOUD SECURITY

7.1 Homomorphic Encryption

Homomorphic encryption is a hot topic in modern communication system. It is a perfect cloud computing security solution. The application scenario allows computations to be carried out on ciphertext, while preserving the privacy of the message.

However, when faced with a large pool of users who wish to compute a function on all their private inputs, FHE can not be applied easily since it can only process data with one private key. A new primitive is created to deal with this issue, which is called multikey fully homomorphic encryption. It is able to compute data which is encrypted under multiple unrelated keys.

7.2 NTRU based Fully Homomorphic Encryption System

In 2012, López-Alt *et al.* [26] found some multikey homomorphic properties in the revised NTRUEncrypt scheme. The properties are given as follows:

$$f_1(x) \cdot f_2(x)[c_1(x) + c_2(x)] = 2e_{add}(x) + f_1(x) \cdot f_2(x)[m_1(x) + m_2(x)] \tag{17}$$

$$f_1(x) \cdot f_2(x)[c_1(x) + c_2(x)] \mod 2 = m_1(x) + m_2(x) \tag{18}$$

$$f_1(x) \cdot f_2(x)[c_1(x) \times c_2(x)] = 2e_{mult}(x) + f_1(x) \cdot f_2(x)[m_1(x) \times m_2(x)] \tag{19}$$

$$f_1(x) \cdot f_2(x)[c_1(x) \times c_2(x)] \mod 2 = m_1(x) \times m_2(x) \tag{20}$$

Some adjustments were made to covert the scheme from the base NTRUEncypt

to the On-the-fly multiparty computation step by step.

First, a relinearizaion technique is used to convert the base NTRUEncrypt scheme to somewhat homomorphic encryption. Then it is converted to fully homographic encryption by using modulus reduction and Gentrys bootstrapping theorem. Finally, a protocol is designed to achieve the on-the-fly multiparty computation. On-the-Fly multiparty computation fulfil these requirements:

- Each of the users encrypts their data and upload to the cloud.
- The cloud can perform computation on the data non-interactively, without any further help from the users. The result is still encrypted.
- The users cooperate to retrieve the evaluated output with their own private key.

In the following chapter, a brief overview of this NTRU based fully homomorphic system is given, which includes key generation, encryption, decryption and evaluation operations.

7.2.1 Key Generation

- Choose decreasing sequence of primes $q_0 > q_1 > ... > q_d$
- Private key: sample $u_{(i)}(x)$ and $g^{(i)}(x)$, set $f^{(i)}(x) = 2u^{(i)}(x) + 1$
- Public Key: $h^{(i)}(x) = 2g^{(i)}(x) \times f^{(i)}(x)^{-1}$
- Evaluation Key: $\zeta_\tau^{(i)}(x) = h^{(i)}(x) \times s_\tau^{(i)}(x) + 2^{(i)}e_\tau(x) + 2^\tau f^{(i-1)}(x)$, where $i = 0, 1, ..., d$ and $\tau = 0, ..., [log_2 q_i]$

$h(x)$ is the public key, while the pair $f(x)$ is the corresponding private key of the system, and there is an additional evaluation key $\zeta_\tau^{(i)}(x)$.

7.2.2 Encryption

- Encode message to binary polynomial $m(x)$.
- Random samples s(x) and e(x)
- Encrypt message $c^{(i)}(x) = h^{(i)}(x) \times s^{(i)}(x) + 2e^{(i)}(x) + m(x) \mod q_i$

7.2.3 Decryption

- Compute $m(x) = c^{(i)}(x) \times f^{(i)}(x) \mod p$

7.2.4 Evaluation

- $c_1^{(i)}(x) = Encrypt[m_1(x)]$ and $c_2^{(i)}(x) = Encrypt[m_2(x)]$
- Addition: $Encrypt[m_1(x) + m_2(x)] = c_1^{(i)}(x) + c_2^{(i)}(x)$
- Multiplication: $Encrypt[m_1(x) \times m_2(x)] = [q_i \varsigma^{(i+1)}(x)/q_{i-1}]_2$

  $\varsigma^{(i)} = c_1^{(i)}(x) + c_2^{(i)}(x)$ and $\varsigma^{(i+1)} = \sum_\tau \zeta_\tau^{(i)}(x)\varsigma^{(i-1)}(x)$

7.3 Proposed Architectures for NTRU based FHE Systems

Since the FHE scheme is based on the revised NTRUEncrypt scheme, it is possible to extend our work to FHE, if we can implement the revised NTRUEncrypt scheme in hardware. Some changes of the revised NTRUEncrypt that may impact the original implementation in hardware are listed below:

- $R = \mathbb{Z}[x]/(x^n + 1)$ is used instead of $R = \mathbb{Z}[x]/(x^n - 1)$, which can be achieved by change the sign of the coefficients.
- Because $p = 2$, the plaintext can be encoded in binary polynomial and the result of modulo p operation can be easily calculated by picking the least significant bit of the coefficient.

- q is a prime integer instead of power of 2, a modulo arithmetic unit is required to calculate modulo q.
- These small polynomials $f(x)$ and $r(x)$ are now sampled from some distributions. The coefficients of these polynomials are wide range numbers instead of $\{-1, 0, 1\}$.
- Because a small polynomial $s(x)$ is added during the encryption, which is presented by $e(x) = h(x) \times r(x) + p \cdot s(x) + m(x) \mod q$, fully integer adders and multipliers are required.

To implement the encryption step and decryption step, the multiplication between two coefficients with arbitrary integer is the most time consuming operation. We extend our previous work and design a LFSR based architecture and a systolic array based architecture to implement the following operation used in NTRU based FHE.

$$c(x) = h(x) \times r(x) \tag{21}$$

7.3.1 LFSR based Architecture

The LFSR based architecture proposed in chapter four can be used. However, since the polynomial ring is under $R = \mathbb{Z}[x]/(x^n + 1)$, some changes need to be made to adapt the structure. Moreover, the small polynomial, which is input serially to the circuit, is no longer a polynomial with coefficients from $\{-1, 0, 1\}$, but a polynomial with coefficients sampled from Gaussian distribution. Therefore, the proposed MAU can not be used in this design, it is replaced with a multiplier and an adder. An LFSR based architecture designed to implement multiplication in FHE, is shown in algorithm 7.1 and Fig.7.1.

Algorithm 7.1 Multiplication in Truncated Polynomial Ring for NTRU based FHE

Input: $h = h_0, ..., h_{n-1}$, $r = r_{n-1}, ..., r_0$

Output: $c = c_0, ..., c_{n-1}$

1: $c^{(0)} = 0$
2: **for** $j = n$ to 1 **do**
3: **for** $i = 0$ to $n - 1$ **do**
4: **if** $i = 0$ **then**
5: $c^{(j)}_{i+1 \mod n} = h_{i+1 \mod n} \times r_{j-1} - c_i^{(j-1)} \mod q$
6: **else**
7: $c^{(j)}_{i+1 \mod n} = c_i^{(j-1)} + h_{i+1 \mod n} \times r_{j-1} \mod q$
8: **end if**
9: **end for**
10: **end for**
11: **return** $c = c^{(n)}$

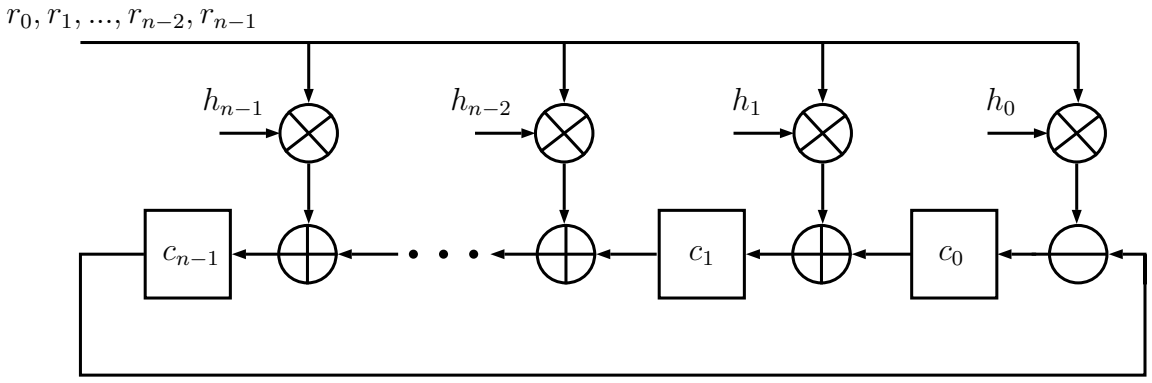


Fig. 7.1: LFSR based Truncated Polynomial Ring Multiplier for NTRU based FHE

Some modifications are made based on the architecture proposed in chapter 4. Firstly, the polynomial $r(x)$ is input from its highest degree coefficient to lowest degree coefficient. Secondly, the propagation direction is from right to left instead of left to right. Lastly the first adder on the right is replaced by a subtractor. The registers $c(x) = (c_0, c_1, ..., c_{n-1})$ are initially loaded with 0. The registers will store the product of $h(x) \times r(x) \mod (x^n + 1)$ after n clock cycles.

7.3.2 Systolic Array based Architecture

The systolic array based architecture proposed in chapter 6 can be also extended to FHE. Because n is a power of 2 number, which must be an even number, the order

of columns is reordered as $c_{n-1}, c_{n-3}, ..., c_1, c_{n-2}, ..., c_2, c_1$. And since the polynomial ring is under $\mathbb{R} = \mathbb{Z}[x]/(x^n + 1)$, some of the elements are negative. It is shown in Fig.7.2.

$$
\begin{array}{cccccccccc}
 & h_{n-1}r_0 & -h_{n-2}r_{n-1} & \cdots & -h_{n/2}r_{n/2+1} & h_{n/2-2}r_{n/2} & \cdots & h_0r_2 & -h_{n-1}r_1 \\
 & h_{n-2}r_1 & h_{n-3}r_0 & \cdots & -h_{n/2-1}r_{n/2+2} & h_{n/2-3}r_{n/2+1} & \cdots & -h_{n-1}r_3 & -h_{n-2}r_2 \\
 & h_{n-3}r_2 & h_{n-4}r_1 & \cdots & -h_{n/2-2}r_{n/2+3} & h_{n/2-4}r_{n/2+2} & \cdots & -h_{n-2}r_4 & -h_{n-3}r_3 \\
 & \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots \\
 & h_2r_{n-3} & h_1r_{n-4} & \cdots & -h_{n/2+3}r_{n/2-2} & h_{n/2+1}r_{n/2-3} & \cdots & -h_3r_{n-1} & -h_2r_{n-2} \\
 & h_1r_{n-2} & h_0r_{n-3} & \cdots & -h_{n/2+2}r_{n/2-1} & h_{n/2}r_{n/2-2} & \cdots & h_2r_0 & -h_1r_{n-1} \\
+ & h_0r_{n-1} & -h_{n-1}r_{n-2} & \cdots & -h_{n/2+1}r_{n/2} & h_{n/2-1}r_{n/2-1} & \cdots & h_1r_1 & h_0r_0 \\
\hline
 & c_{n-1} & c_{n-3} & \cdots & c_1 & c_{n-2} & \cdots & c_2 & c_0
\end{array}
$$

Fig. 7.2: Modified Truncated Polynomial Ring Multiplication for NTRU based FHE in Vertical Form

Two types of processing element are required in the systolic array architecture. One of which is used to deal with positive elements (Fig.7.3) consist of a multiplier and an adder, the other one is used to process negative elements (Fig.7.4) which include a multiplier and a subtractor. A ■ represents a m-bits (m is equal $\log_2 q$) latch. With these two PEs, a systolic array architecture for multiplication in revised NTRUEncrypt is designed and shown in Fig.7.5.

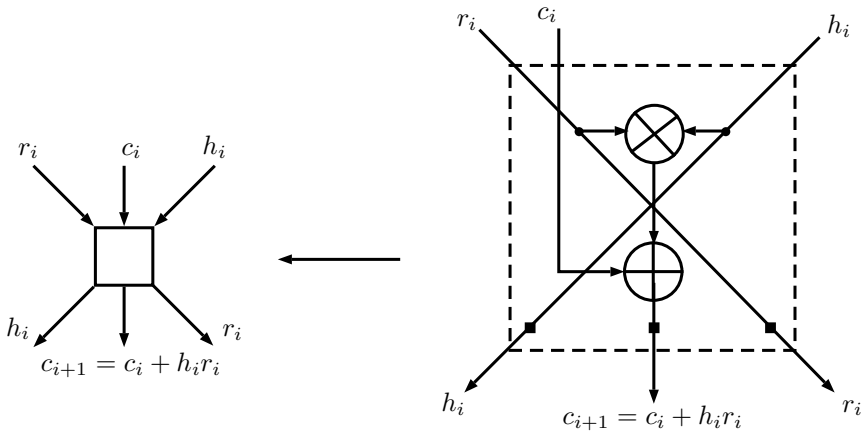


Fig. 7.3: PE for Positive Elements

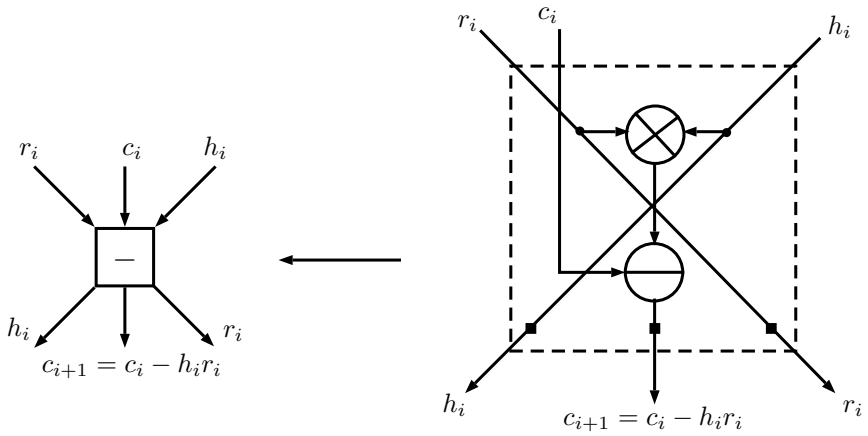


Fig. 7.4: PE for Negative Elements

The systolic array base architecture consist of n^2 PEs which include $n(n+1)/2$ positive PEs and $n(n+1)/2$ negative PEs. Every PE is connected to its neighbour PEs same as the previous proposed architecture, except the connection between the $n/2$ and $n/2+1$ columns. The output h_i of PEs of the $n/2+1$ column are connected to the PEs of $n/2$ column in the manner shown in Fig.7.5. The coefficients of $c(x)$ are initially as 0 and input to the first row of the systolic array. It takes N clock cycle to evaluate every coefficient of $c(x)$.

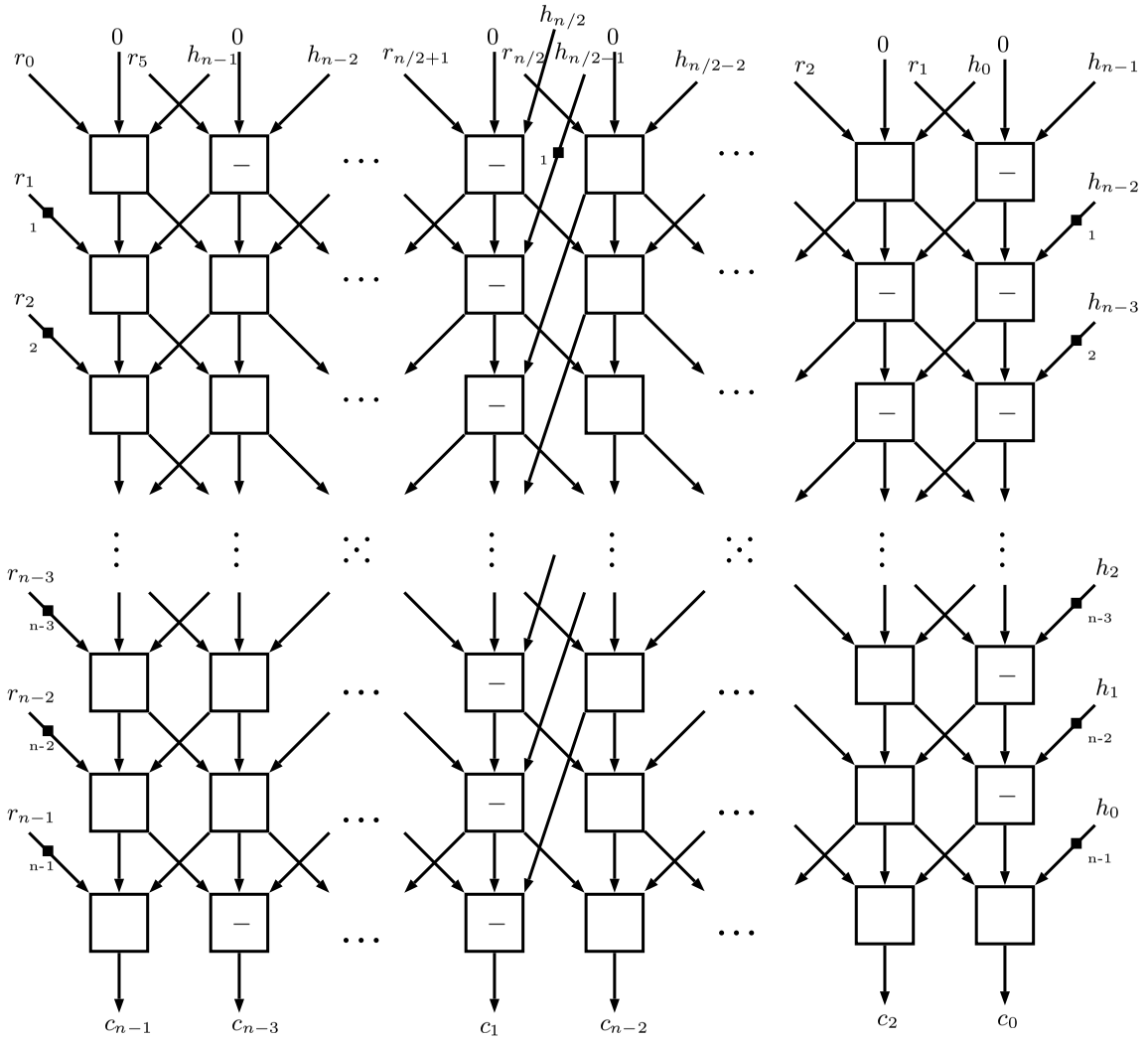


Fig. 7.5: Systolic Array based Truncated Polynomial Ring Multiplier for NTRU based FHE

8 CONCLUSIONS AND POSSIBLE FUTURE WORK

In this thesis several new architectures for NTRUEncrypt and NTRU based homomorphic encryption have been proposed. FPGA implementations have also been performed for the LFSR based architectures.

Firstly a new architecture using LFSR for NTRUEncrypt system is presented. The new architecture takes advantage of LFSR structure for its compact circuitry and high speed. Note that this architecture can be used for both encryption (Fig.4.5) and decryption (Fig.4.5) in NTRUEncrypt system. Our FPGA implementation has shown that the proposed design uses modest area and works at relatively high speed. It outperforms all the existing works in comparison in term of area-delay product.

Further enhancement on the efficiency of the LFSR based architecture is proposed by using extended LFSR architecture (Fig.5.4). It takes advantage of small polynomials with large number of zero coefficients, and thus significantly reduces the clock cycle required. The FPGA implementation result shows that the proposed design uses slightly larger resource but has much faster speed.

A systolic array architecture is proposed for NTRUEncrypt (Fig.6.4). There is only one type of PE in the array and the PE was designed with optimized arithmetic. The systolic array yields all the output in N clock cycles.

Besides, new architectures are proposed for computation of NTRU based fully homomorphic encryption system. LFSR is applied in one of the architecture with a novel design of the modular multiplication unit (Fig.7.1), while the other proposed architecture is systolic array with two types of PEs (Fig.7.5).

In the future, the research works presented in this thesis can be further extended in the following aspects:

- The systolic array architecture for NTRUEncrypt and two new architectures for NTRU based fully homomorphic encryption system will be implemented using

FPGA.

- Since NTRU based fully homomorphic encryption requires large-valued parameters, further enhancement will focus on how to improve the efficiency of the system.

- The next goal of our work is to apply these architectures to other steps of the systems including key generation and evaluation part in FHE system.

- There are other lattice based cryptosystems which have similar property and operation as NTRU system. We are expected to extended our work to these systems.

REFERENCES

[1] W. G. of the C/MM Committee *et al.*, "IEEE p1363.1 standard specification for public-key cryptographic techniques based on hard problems over lattices," no. March, 2009.

[2] D. Cabarcas, P. Weiden, and J. Buchmann, "On the efficiency of provably secure NTRU," in *Post-Quantum Cryptography*. Springer, 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 *Applied cryptography and network security*. Springer, 2009, pp. 437–455.

[4] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM journal on computing*, vol. 26, no. 5, pp. 1484–1509, 1997.

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

[6] R. J. McEliece, "A public-key cryptosystem based on algebraic coding theory," *DSN progress report*, vol. 42, no. 44, pp. 114–116, 1978.

[7] H. Niederreiter, "Knapsack-type cryptosystems and algebraic coding theory," *Problems of Control and Information Theory*, vol. 15, no. 2, pp. 159–166, 1986.

[8] L. Lamport, "Constructing digital signatures from a one-way function," Technical Report CSL-98, SRI International Palo Alto, Tech. Rep., 1979.

[9] R. C. Merkle, "Secrecy, authentication, and public key systems." Ph.D. dissertation, 1979.

[10] J. Ding and D. Schmidt, "Rainbow, a new multivariable polynomial signature scheme," in *Applied Cryptography and Network Security*. Springer, 2005, pp. 164–175.

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

[12] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *Algorithmic number theory*. Springer, 1998, pp. 267–288.

[13] D. Stehlé and R. Steinfeld, "Making NTRU as secure as worst-case problems over ideal lattices," in *Advances in Cryptology–EUROCRYPT 2011*. Springer, 2011, pp. 27–47.

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

[15] C. Gentry *et al.*, "Fully homomorphic encryption using ideal lattices." in *STOC*, vol. 9, 2009, pp. 169–178.

[16] C. Gentry, "Toward basing fully homomorphic encryption on worst-case hardness," in *Advances in Cryptology–CRYPTO 2010*. Springer, 2010, pp. 116–137.

[17] D. Stehlé and R. Steinfeld, "Faster fully homomorphic encryption," in *Advances in Cryptology-ASIACRYPT 2010*. Springer, 2010, pp. 377–394.

[18] N. P. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," in *Public Key Cryptography–PKC 2010*. Springer, 2010, pp. 420–443.

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

[20] 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.

[21] C. Gentry and S. Halevi, "Implementing gentrys fully-homomorphic encryption scheme," in *Advances in Cryptology–EUROCRYPT 2011*. Springer, 2011, pp. 129–148.

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

[23] 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.

[24] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Advances in Cryptology–CRYPTO 2012*. Springer, 2012, pp. 868–886.

[25] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 75–92.

[26] 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.

[27] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, "Improved security for a ring-based fully homomorphic encryption scheme," in *Cryptography and Coding*. Springer, 2013, pp. 45–64.

[28] Y. Doröz, Y. Hu, and B. Sunar, "Homomorphic aes evaluation using ntru." *IACR Cryptology ePrint Archive*, vol. 2014, p. 39, 2014.

[29] W. Dai, Y. Doroz, and B. Sunar, "Accelerating ntru based homomorphic encryption using gpus," in *High Performance Extreme Computing Conference (HPEC), 2014 IEEE*. IEEE, 2014, pp. 1–6.

[30] K. Rohloff and D. B. Cousins, "A scalable implementation of fully homomorphic encryption built on ntru," in *Financial Cryptography and Data Security*. Springer, 2014, pp. 221–234.

[31] V. Lyubashevsky, "Lattice-based identification schemes secure under active attacks," in *Public Key Cryptography–PKC 2008*. Springer, 2008, pp. 162–179.

[32] V. Lyubashevsky and D. Micciancio, "Generalized compact knapsacks are collision resistant," in *Automata, Languages and Programming*. Springer, 2006, pp. 144–155.

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

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

[35] D. V. Bailey, D. Coffin, A. Elbirt, J. H. Silverman, and A. D. Woodbury, "NTRU in constrained devices," in *Cryptographic Hardware and Embedded SystemsCHES 2001*. Springer, 2001, pp. 262–272.

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

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

[38] H. Wu and M. A. Hasan, "Efficient exponentiation of a primitive root in gf (2 m)," *Computers, IEEE Transactions on*, vol. 46, no. 2, pp. 162–172, 1997.

[39] H. Kungt and C. E. Leisersont, *Systolic arrays (for VLSI)*, 1979.

[40] H.-T. Kung, "Why systolic architectures?" *IEEE computer*, vol. 15, no. 1, pp. 37–46, 1982.

VITA AUCTORIS

NAME:	Bingxin Liu
PLACE OF BIRTH:	Guangzhou, China
YEAR OF BIRTH:	1988
EDUCATION:	Guangdong University of Technology, Guangzhou, China Bachelor of Engineering, Surveying and Control Technology and Instrumentation 2007-2011 University of Windsor, Windsor ON, Canada Master of Applied Science, Electrical and Computer Engineering 2012-2015