Accelerating homomorphic encryption for privacy-preserving applications

Ho, Truong Phu Truan

2020

Ho, T. P. T. (2020). Accelerating homomorphic encryption for privacy-preserving applications. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/138531>

<https://doi.org/10.32657/10356/138531>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 11:18:25 SGT

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

**ACCELERATING HOMOMORPHIC
ENCRYPTION FOR PRIVACY-PRESERVING
APPLICATIONS**

HO TRUONG PHU TRUAN

**SCHOOL OF ELECTRICAL AND ELECTRONIC
ENGINEERING**

2020

ACCELERATING HOMOMORPHIC ENCRYPTION FOR PRIVACY-PRESERVING APPLICATIONS

HO TRUONG PHU TRUAN

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2020

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

02 April 2020

.....
Date

.....
Ho Truong Phu Truan

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

10 April 2020

.....
Date

.....
Chang Chip Hong

Authorship Attribution Statement

This thesis contains material from 3 paper(s) published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Chapter 3 is published as Truong Phu Truan Ho and Chip-Hong Chang, “Accelerating residue-to-binary conversion of very high cardinality moduli set for fully homomorphic encryption”, in *Proceedings of 2016 IEEE Asia Pacific Conference on Circuits and Systems*, Jeju, Korea, pp. 9-12, October 2016.

The contributions of the co-authors are as follows:

- I proposed the initial idea. The idea was revised by Prof Chang.
- I designed the study and performed all the laboratory work at VIRTUS, IC Design Centre of Excellence, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore.
- I prepared the manuscript draft. The manuscript was revised by Prof Chang.

Chapter 4 is published as Truong Phu Truan Ho and Chip-Hong Chang, “Towards Ideal Lattice-Based Cryptography on ASIC: A Custom Implementation of Number Theoretic Transform”, in *Proceedings of 2018 IEEE 23rd International Conference on Digital Signal Processing*, Shanghai, China, pp. 1-5, November 2018.

The contributions of the co-authors are as follows:

- I proposed the initial idea. The idea was revised by Prof Chang.
- Prof Chang suggested to use a memory compiler for dual-port RAM generation.
- I designed the study and performed all the laboratory work at VIRTUS, IC Design Centre of Excellence, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore.
- I prepared the manuscript draft. The manuscript was revised by Prof Chang.

Chapter 5 is under review as Truong Phu Truan Ho and Chip-Hong Chang, “Privacy Preserving Outsourced Functional Evaluation via Residue Number System and Fuzzy Vault”, submitted to IEEE Transactions on Dependable and Secure Computing.

The contributions of the co-authors are as follows:

- Prof Chang suggested the concept of fuzzy vault
- I proposed to incorporate residue number system to fuzzy vault. The idea was revised by Prof Chang.
- I designed the study and performed all the laboratory work at VIRTUS, IC Design Centre of Excellence, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore.
- I prepared the manuscript draft. The manuscript was revised by Prof Chang.

02 April 2020

A handwritten signature in black ink, appearing to read 'Phu Truan', with a long horizontal stroke extending to the right.

Ho Truong Phu Truan

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisor, Associate Professor Chang Chip Hong, for his restless support and guidance throughout the research. Without his continuous encouragement, I would have given up on my PhD study.

I am also very grateful to my parents and my younger brother, who have always stood by me and strengthened me throughout the years. Without their endless love, support, and encouragement, this work would not have been possible.

Last but not least, I would like to say thank you to all the friends I made during the last year of my graduate study. The time we have spent together is definitely my most enjoyable experience in Singapore.

Table of Contents

Acknowledgements	i
Table of Contents	ii
Abstract	v
List of Figures	viii
List of Tables	x
List of Abbreviations	xi
Chapter 1	1
Introduction	1
1.1 Motivation	1
1.2 Objectives	6
1.3 Major Contributions of the Thesis	7
1.4 Organization of the Thesis	9
Chapter 2	11
Background and Literature Review	11
2.1 Partially Homomorphic Encryption Schemes	11
2.1.1 RSA Scheme	11
2.1.2 Benaloh Scheme	12
2.1.3 Remark	14
2.2 Gentry's Integer-based Fully Homomorphic Encryption Scheme	14
2.2.1 Symmetric-key Somewhat Homomorphic Encryption Scheme	15
2.2.2 Public-key Somewhat Homomorphic Encryption Scheme	16
2.2.3 Public-key Fully Homomorphic Encryption Scheme	19
2.3 State-of-the-art Homomorphic Encryption Schemes	23
2.3.1 Learning-with-Errors and Ring Learning-with-Errors Problems	23
2.3.2 Homomorphic Encryption Schemes from Ring Learning-with-errors	27
2.4 Algorithm and Architecture for Hardware Accelerators	34
2.4.1 Accelerating Gentry's Ideal Lattice Based Scheme	34
2.4.2 Accelerating CNT's Integer-based Scheme	46
2.4.3 Accelerating Schemes from Ring Learning-with-errors Problem	54
2.5 Conclusion	58
Chapter 3	60
Accelerating Conversion from Residue to Binary Representatio of Very High Cardinality Arbitrary Moduli Set for Homomorphic Encryption	60

3.1 Introduction	60
3.2 Background	62
3.3 Proposed Algorithm and Hardware Architecture	66
3.4 Performance Analysis and Comparison	73
3.4.1 Delay of the Proposed Method.....	73
3.4.2 Delay of Wang's Method	75
3.4.3 Delay of Roy's Method	75
3.4.4 Delay of Doroz's Method.....	76
3.4.5 Comparison.....	76
3.5 Summary	79
Chapter 4	80
Exploring Number Theoretic Transform on ASIC for Ring Learning-with-Error Based Cryptography	80
4.1 Introduction	80
4.2 Number Theoretic Transform	81
4.3 Proposed Architecture	84
4.3.1 Simplified Modulo Multiplier	85
4.3.2 Modulo Adder and Subtractor	87
4.3.3 Overall Architecture	88
4.4 Evaluation and Comparison	90
4.5 Conclusion	94
Chapter 5	95
Privacy Preserving Polynomial Evaluation via Residue Number System and Fuzzy Vault.....	95
5.1 Introduction	95
5.2 Background	96
5.2.1 Residue Number System	96
5.2.2 Fuzzy Vault	97
5.2.3 Homomorphic Encryption with Noise	97
5.3 System and Privacy Models	98
5.3.1 System Model	98
5.3.2 Privacy Model	99
5.4 Proposed Scheme.....	99
5.4.1 Intuition	99
5.4.2 Construction.....	100
5.4.3 Homomorphic Operations	103
5.5 Security	106

5.5.1 Analysis of ENCRYPT	107
5.5.2 Analysis of COMP	110
5.5.3 Parameters Requirements	112
5.6 Applications	113
5.6.1 Scenarios	113
5.6.2 Statistical Functions	115
5.6.3 Predictive Analysis for Medical Applications	118
5.7 Performance and Comparison	121
5.7.1 Benchmarking Settings	121
5.7.2 Applications Performance	123
5.7.3 Discussion.....	123
5.8 Conclusion	125
Chapter 6	126
Conclusions and Future Works	126
6.1 Conclusions.....	126
6.2 Future Works.....	128
Author's Publications.....	131
Bibliography.....	132

Abstract

In the first industrial revolution, steam power transformed production from manual to mechanized. In the second industrial revolution, electric energy is the driving force of mass production. The third industrial revolution witnessed the birth of automatic production from electronics and information technology. Today, it is believed that data is the most essential factor that lays the foundation of the fourth industrial revolution. The recent emergence of new paradigms of automation such as Big Data, Internet of Things, Cloud Computing, Artificial Intelligence, etc. has assisted humanity greatly in collecting, processing, sharing, and exploiting the massive data generated in every second. The disappearance or closing of physical space and time barriers has led to boosted efficiency, higher level of productivity and cost reduction. However, these developments come with the sacrifice of user privacy since intensive collection and sharing user information can significantly increase the chance of data leakage. Recent incidents of data breaches have raised an increasing public concern about whether current practices can completely prevent private information from falling into the hands of unwanted party. Homomorphic encryption, a concept that is regarded as the cryptography holy grail, is considered the most promising solution for privacy preservation for sensitive applications. This concept guarantees data privacy by allowing computation to be performed directly on encrypted data without the need for decryption, and keeping the final result in encrypted form. Thus, only data owners, who are in possession of the secret key, can decrypt the encrypted result. Despite its theoretical elegance and correctness, as well as major advancements since its inception, homomorphic encryption is still far from being practical for large scale deployment due to its computational complexity, and exorbitantly large and massive parameters. Thus, the primary objective of this thesis is to improve the performance of homomorphic encryption.

Throughout the history of cryptology, hardware platforms have been successfully used for alleviating performance and bottleneck computations. The limitations of state-of-art hardware implementations of homomorphic encryption

are identified and viable solutions are proposed in this research. First of all, a fast residue-to-binary conversion method is presented. To accelerate large integer arithmetic, existing hardware implementation of homomorphic encryption exploit very high cardinality arbitrary moduli sets of residue number system. However, large modulo operation and limited number theoretic property of arbitrary moduli set cause slow residue-to-binary conversion and significantly undermine the gained benefit. The proposed method base-extends from arbitrary moduli set to the special moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ before the final conversion. Thanks to the smaller and parallel operations of base extension as well as efficient reverse conversion method for this special three-moduli set, the proposed method is 2.25 ~ 6.8 times faster than existing residue-to-binary converters for the same large arbitrary moduli sets for homomorphic encryption, and can improve the overall speed of existing hardware implementation of homomorphic encryption by 10.2% to 32.08%. Secondly, a custom implementation of Number Theoretic Transform is presented. Number Theoretic Transform is the essential algorithm to accelerate the bottleneck operations in homomorphic encryption. Although efficient implementation of Number Theoretic Transform on various software and reconfigurable hardware platforms has been widely studied, its ASIC design remains unexplored at the time of this research. By optimizing the modulo multiplier designs of Barrett and Montgomery multiplication algorithm, and designing the butterfly circuit around two parallel half-sized dual-port RAMs, the proposed custom architecture substantially cut down the latency and boost the throughput of homomorphic encryption for large parameter sets by 28.38% to 54.85% comparing with previous architectures on reconfigurable hardware.

While the performance can be boosted by hardware accelerators, noise growth in homomorphic encryption can increase the ciphertext size and severely limits the performance of homomorphic applications. A method is proposed to prevent the noise growth from affecting the size of the ciphertexts via the homomorphism of residue number systems and fuzziness offered by fuzzy vault construction. For the same application, the proposed method achieves a smaller ciphertext size compared with other homomorphic encryption schemes. For logistic

regression model, the CPU implementation of the proposed method can achieve a speedup of 5.34 times over the implementation of an existing scheme on a powerful CPU-GPU platform.

List of Figures

Figure 1.1 : A typical application of homomorphic encryption.....	4
Figure 2.1: An example of learning-with-errors problem with $n = 4$ and $q = 13$	24
Figure 2.2: An example of negacyclic shift in ring learning-with-errors problem with $n = 4$ and $q = 13$	26
Figure 2.3: An example of ring learning-with-errors problem in polynomial ring $Z_{13}[x]/(x^4+1)$	27
Figure 2.4: N-point NTT-based integer multiplication.	38
Figure 2.5: Architecture of modular multiplication unit.	43
Figure 2.6: Architecture of radix-16 64K point NTT processor.....	44
Figure 2.7: Architecture of unit for performing 16-point NTT.....	45
Figure 2.8: Architecture of unit for modulo summing 16 operands.....	45
Figure 2.9: Parallelizing multiplication using Comba method for efficient DSP-based multiplication.	48
Figure 2.10: Low-area (top) and parallel (bottom) architecture design of encryption step.	50
Figure 2.11: Architecture of low Hamming weight multiplier.	52
Figure 2.12: Block accumulation logic used in NTT-based large integer multiplication for Barrett reduction.	52
Figure 2.13: Architecture of NTT-based large integer multiplier for Barrett modulo reduction.	53
Figure 2.14: NTT-RNS polynomial multiplication algorithm.....	57
Figure 2.15: Parallel hardware architecture for in-place NTT-RNS polynomial multiplication algorithm.....	58
Figure 3.1: An example to illustrate Mixed Radix Conversion method	64
Figure 3.2: Architecture of base extension operation in Stage 1.....	69
Figure 3.3: Architecture to determine r for base extension operation.	70
Figure 3.4: Residue-to-binary conversion architecture for $\{2^n-1, 2^n, 2^n+1\}$	71
Figure 3.5: Blocks "nFA1" and "nFA2"	72
Figure 4.1: Cooley-Tukey radix-2 Decimation-in-time butterfly.....	84
Figure 4.2: Hardware architecture of simplified Barrett multiplier.....	86
Figure 4.3: Hardware architecture of simplified Montgomery multiplier.....	87
Figure 4.4: Hardware architecture of (a) modulo adder and (b) modulo subtractor. ...	88
Figure 4.5: Hardware architecture for the computation of NTT.....	89
Figure 4.6: Comparison of normalized synthesized area of ASIC reimplementation of FPGA designs and the proposed designs with either Barrett or Montgomery multiplier under different combination of (N, p)	91
Figure 4.7: Comparison of frequency of designs on FPGA, their ASIC reimplementation, and the proposed designs with either Barrett or Montgomery multiplier under different combination of (N, p)	92
Figure 4.8: Comparisons of required cycles per NTT of other designs and the proposed designs with either Barrett or Montgomery multiplier under different combination of (N, p)	92
Figure 4.9: Comparisons of normalized NTT operation per second of designs on FPGA, their ASIC reimplementation, and the proposed designs with either Barrett or	

Montgomery multiplier under different combination of (N, p)	93
Figure 5.1: Illustrative example of multiplying two ciphertexts by algorithm COMP. .	105
Figure 5.2: Outsourcing a public computation by the proposed method.	114
Figure 6.1: Deploying RRNS in multi clouds settings.	129

List of Tables

Table 2.1: Different parameter sets in the simplified ideal-lattice based scheme.	35
Table 2.2: Performance of the simplified ideal-lattice scheme. Timing on an IBM x3500 server with 64-bit quad-core Intel Xeon E5450 at 3GHz, 12MB L2 cache and 24GB RAM.	35
Table 2.3: Performance of the simplified ideal-lattice based homomorphic encryption scheme's primitive on CPU vs. GPU.	41
Table 2.4: Memory usage on Tesla C2050 GPU.	42
Table 2.5: Performance of Coron-Naccache-Tibouchi integer-based scheme. Timing on a single core of Intel Core2 Duo E8400 processor at 3GHz.	46
Table 2.6: Different parameter sets in encryption step of CNT integer-based scheme.	49
Table 2.7: Average encryption time of Coron-Naccache-Tibouchi integer-based scheme using Comba multiplication.	51
Table 2.8: Average encryption time comparison of NTT-based design and Comba-based design.	54
Table 2.9: Timing of FV's primitives on Intel Core i7-2600 @ 3.4 GHz with no turbo-boost and hyper-threading.	54
Table 3.1: Performance comparison of residue-to-binary converters.	77
Table 3.2: Synthesis results of different residue-to-binary converters.	78
Table 4.1: Delay (D) and area (A) of the butterfly circuit constructed with either the simplified Barrett or Montgomery multiplier.	91
Table 5.1: Implementation environment of the proposed work and others.	121
Table 5.2: Comparison of the performance of operations between this work and others.	122
Table 5.3: Comparison of the performance of candidate applications between this work and others.	123
Table 5.4: Comparison of ciphertext size and form between the proposed scheme and others.	125

List of Abbreviations

AI	Artificial Intelligence
ASIC	Application Specific Integrated Circuit
BGV	Brakerski – Gentry – Vaikuntanathan
CGL	Carry Generation Logic
CLA	Carry Look Ahead
CMOS	Complementary Metal Oxide Semiconductor
CNT	Coron – Naccache – Tibouchi
CPHM	Cox Proportional Hazard Model
CPU	Central Processing Unit
CRT	Chinese Remainder Theorem
DSP	Digital Signal Processing
ECC	Elliptic Curve Cryptography
FA	Full Adder
FV	Fan – Vercauteren
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GMP	GNU Multiple Precision arithmetic library
GPU	Graphics Processing Unit
HA	Half Adder
IFFT	Inverse Fast Fourier Transform
IoT	Internet of Things
INTT	Inverse Number Theoretic Transform
LRM	Logistic Regression Model
LSB	Least Significant Bit
M&V	Mean and Variance
MACC	Multiply and Accumulate
MRC	Mixed Radix Conversion
MSB	Most Significant Bit
MUX	Multiplexor
NTL	A Library for doing Number Theory
NTT	Number Theoretic Transform
R2B	Residue to Binary
RAM	Random Access Memory
RNS	Residue Number System
RRNS	Redundant Residue Number System
ROM	Read Only Memory
RSA	Rivest – Shamir – Adleman
SMB	Small and Medium Business

Chapter 1

Introduction

1.1 Motivation

In these days, technologies have become a dominant factor in growing businesses by boosting efficiency, driving higher level of productivity and reducing costs. With the convergence of smart technologies like artificial intelligence (AI), Internet-of-Things (IoT), Big Data, 5G, etc., powerful computing resources are the key success drivers for the purpose of storing, sharing, and analyzing the enormous digital data generated in every second. The building or hiring of these powerful computing resources, often called datacenters, required a large initial investment from businesses, let alone the investment for specialized human resources and utility cost. Unsurprisingly, such datacenter is expected to be expensive and its cost may rise up to \$100 million [1]. Besides, there are inevitable expenses on maintenance and upgrading as well as the cost for further IT training. Such costs are only affordable by business giants, but not the small and medium businesses (SMBs) and start-ups. The huge capital investment on datacenter can hinder their potential and capacity to grow.

A solution to this problem is to outsource computations and storage to clouds. A cloud is defined as a data center or multiple data centers consisting of computing and storage resources connected by a network. In cloud computing, all resources are virtualized as a big giant shared pool of resources which is automatically and intelligently configured to meet dynamic needs of applications. The cloud is capable of adapting to the ever changing utilization and availability of each resource. Therefore, cloud computing frees companies from the need to own and maintain datacenters, which results in a significantly lower entry barrier with substantially reduced initial investment. Furthermore, since a cloud is a shared resource, the more customers a cloud serves, the less a customer will have to pay to the cloud service providers. In addition, clouds

automatically monitor, maintain and upgrade all hosted programs, run regular data backup, handle all backup costs, etc., and all of these features become a huge convenience that cloud computing can offer to existing and new SMBs.

Being cost effective and efficient, it is not surprising that over the last few years, cloud computing has evolved from a nascent technology into an established solution with widespread acceptance and deployment. A recent study reported that for the fourth quarter of 2015, cloud market revenues reached \$110 billion with 28 percent of annual growth rate [2]. In 2020, clouds are expected to process 92 percent of total workloads, accounting for more than 92 percent of total data center traffic, and hold 88 percent shares of total data center storage capacity [3]. Together with AI, IoT, Big Data, and 5G, cloud computing is clearly one of the most massive and impactful endeavors of this century and is expected to be an indispensable key driver of future infocomm technologies.

However, it is necessary to guarantee privacy and confidentiality preservation in adopting cloud computing. The emergence of new technologies like AI, IoT, Big Data, 5G ... have boosted the intensity of data collection, sharing and processing on clouds, which has led to the increased likelihood of data exposure to and abuse by unwanted parties. Recent data breaches have raised questions on whether clouds are able to safeguard user's sensitive information from cyberespionage such as hackers and resource-rich government organizations [4]. The most notably case is the mega breach from Target Corporation in late 2013, where 40 million credit/debit cards and 70 million records including names, addresses, email addresses and phone numbers of Target's shoppers are stolen [5]. Due to the Target breach, business management concern rose significantly and methods to fortify security was applied, such as allocating more resources, strengthening cyber defense, changing operations and compliance processes, etc. [5]. Nevertheless, big data breaches still occur in the following years from 2014 to the current year after the Target breach and cause tremendous economic losses and reputation damage to businesses [6], [7], [8], [9], [10], [11].

Various organizations provide different reports trying to estimate the economic

damage of a data breach to a business. In 2014, McAfee informed that the annual cost to the global economy from data breaches is \$160 billion [12]. In 2015, Kaspersky Lab estimated an average loss of more than half a million dollars caused by a data breach to a typical enterprise [13]. In 2016, a study carried out by Ponemon Institute and IBM reported that the average total cost of a data breach is \$4 million, and the average cost per lost or stolen record works out to \$158 [14]. It is anticipated that in 2019, the annual cost incurred by data breaches will rise above \$2 trillion, which accounts for 2.2% of the projected global GDP [15]. Nonetheless, all of these estimations are believed to be just the tip of an iceberg. This is due to several factors. First of all, to prevent reputation damage, companies are more likely to hide or avoid revealing the full extent of their data breaches; secondly, not all of the data breaches are detected immediately after their occurrences; lastly, it is hard to estimate the tangible financial gain a criminal can get from the stolen data.

Clearly, encrypted data stolen in any breach are of no value to the thieves unless they manage to acquire the decryption key of the data owners. Therefore, by keeping sensitive data encrypted at all time, businesses can stay secure in the cloud. However, with conventional encryption algorithm, it is impossible to perform computation on encrypted data without having to decrypt it first, thus alleviating the advantages of cloud platform. To this end, it is necessary to construct encryption schemes where arbitrary computation can be performed directly on encrypted data without requiring access to a decryption key, leaving the result in encrypted form. Such encryption schemes are referred to as homomorphic encryption. Figure 1.1 (source: www.microsoft.com) illustrates an overview of a private medical prediction - a typical application of homomorphic encryption in the context of cloud computing. In this figure, a user can request the cloud for the encrypted prediction on the private medical data. The cloud learns nothing about the input data. Therefore, privacy is assured even if data leakage occurs in the cloud.

Figure 1.1 : A typical application of homomorphic encryption.

The concept of homomorphic encryption is first introduced in 1978 by Rivest, Adleman, and Dertouzos [16] and quickly becomes the holy grail of cryptography. Although conceptually simple, constructing cryptography's Holy Grail is well-known as a difficult problem. In thirty years after 1978, despite many efforts, all proposed schemes are only partially homomorphic, i.e., homomorphic with respect to a single algebraic operation, either addition or multiplication [17], [18], [19], [20], [21]. Although partially homomorphic encryption schemes have found their uses in practice [22], they are application-specific and not suitable for cloud's environment where the types of applications are diverse and versatile. The problem of constructing a homomorphic encryption scheme stayed open until 2009 when Craig Gentry showed the first scheme based on ideal lattices [23]. By constructing the ciphertext as a linear function of the plaintext marked with a small noise term, Gentry achieved a somewhat homomorphic encryption scheme, i.e., the scheme is homomorphic with respect to a limited number of additions modulo 2 (XOR gates) and multiplications modulo 2 (AND gates) because of accumulated noise. He ingeniously observed that if the complexity of decryption circuit is simple enough, i.e., the Boolean

form of decryption circuit consists of a limited number of AND gates, then noise in ciphertext can be removed by running the decryption circuit homomorphically. Thus, any arbitrary computation can be homomorphically evaluated by expressing it as a binary circuit. When noise in ciphertext reaches a certain level, the ongoing computation is paused to homomorphically run decryption circuit, then it is resumed.

Although being theoretically elegant, Gentry's scheme offers a very poor performance: noise term grows quadratically for every homomorphic multiplication of two binary messages, and translating an arbitrary function to its equivalent binary form can result in a circuit with lots of AND gates, which further accelerates noise growth. These facts are confirmed by several following works trying to implement it, where a single encryption takes 82 seconds on CPU [24] and 4.1 seconds on Nvidia GTX GPU [25]. It is even worse when performing a single homomorphic decryption: 800 seconds on CPU [24] and 235 seconds on Nvidia GTX Graphic Processing Unit (GPU) [25]. Nevertheless, Gentry's idea has ignited a great interest in cryptographic community, which leads to the emergence of more efficient and simpler schemes than his original work [26], [27], [28]. Major advancements have been made since then, which includes the reduction of noise growth from quadratic to linear via extra computations after a homomorphic multiplication [29 - 30], and further reduction of the effect of noise growth by the support homomorphism [31]. Even then, current schemes are still not efficient enough for practice due to algorithmic complexity and large size of parameters. In a recent software implementation, it takes 36.7, 1.3 and 70 minutes to perform key generation, encryption and homomorphic function evaluation, respectively, of only a lightweight block cipher SIMON-64/128 on a 4-core Intel Core i7 processor [32].

Accelerating performance-critical cryptographic computations by using hardware platforms, such as application specific integrated circuit (ASIC), field programmable gate array (FPGA), and GPU, has been a common practice throughout the history of cryptology. Though the number of homomorphic encryption hardware implementation is limited, their results showed that hardware acceleration could bring homomorphic encryption technology closer

to practice [33], [34], [35], [36], [37], [38], [39]. For example, homomorphic evaluation of SIMON-64/128 time in [32] can be reduced significantly from 70 minutes to 2.6 minutes with the help of a coprocessor implemented on FPGA [34]. Thus, it is evident that the poor performance of homomorphic encryption can be alleviated significantly by using hardware platforms as a coprocessor to handle intensive and performance-critical operations. Even though, homomorphic encryption time overhead is still relatively large and not viable for most applications in practice. Comparing to a typical time cost for a random Google search, which is about 0.5 seconds, homomorphic evaluation of SIMON-64/128 on FPGA is still many times larger [34].

In short, the intensive data collection and processing alongside the convergence of multidisciplinary technologies of cloud, big data analytics, pervasive sensing, wireless connectivity and internet of things, also inflict rampant data breaches and growing privacy concerns when these systems offload their computations to remote cloud servers. A practical homomorphic encryption scheme, or any hardware or software acceleration method that can close in the bottleneck operations of a poor-performance scheme to a practical one, will be a potential solution driver to completely securing computation on the cloud. Besides that, practical homomorphic encryption schemes or performance-enhancing methods are also useful in constructing interesting privacy-sensitive applications such as multiparty computation, secret sharing scheme, zero-knowledge proof, and oblivious transfer [40].

1.2 Objectives

The primary objective of this research is to develop solutions that can significantly enhance the performance of homomorphic encryption schemes. The strategy is to study the existing approaches, find out their limitations, and develop solutions to remove the identified limitations. Since hardware acceleration has been proved to be a promising option to bring the performance of homomorphic encryption closer to an acceptable level, hardware-architecture of state-of-the-art homomorphic encryption accelerator will be firstly studied. Specifically, hardware architectures for accelerating multiplication of very high

degree and very large coefficient polynomials, which constitute the performance bottleneck of homomorphic encryption, are evaluated to identify the gap for improvement. After that, algorithms and circuits will be developed to remove the identified weakness without undermining the security strength. Besides architectures for hardware accelerators, algorithmic construction of state-of-the-art homomorphic encryption scheme are also studied to find out the constraints that impede leapfrog performance improvement. New algorithmic constructions, which are expected to offer homomorphism and good performance, will be developed. To validate its practicality, the newly developed algorithm will be applied onto selected privacy-sensitive applications to determine its contribution to the processing speed at application level. The practicality of the newly developed algorithm will be verified by comparing its performance with state-of-the-art works based on their reported figures of merit.

1.3 Major Contributions of the Thesis

In this thesis, the following contributions have been made to address the aforementioned research objectives:

- a) A method for accelerating residue-to-binary conversion of very high cardinality arbitrary moduli set has been developed. State-of-the-art hardware implementations of homomorphic encryption rely on very high cardinality arbitrary moduli set for efficient computation of large number arithmetic. However, the benefit they gained are heavily offset by the slow residue-to-binary conversion due to the large modulo operations and limited number theoretic properties of arbitrary moduli. The proposed method transforms a large number of residues of arbitrary moduli into only three residues corresponding to the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ before the final conversion. This strategy offers two distinct advantages: firstly, base extension for the mapping to $\{2^n - 1, 2^n, 2^n + 1\}$ consists of smaller and parallel arithmetic operations when compared to direct mapping from a large number of residues of arbitrary moduli; secondly, special number theoretic

property of the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ enables an efficient fully combinational adder circuit based residue-to-binary conversion to further reduce the computational complexity. The proposed algorithm significantly outperforms other residue-to-binary conversion algorithms used to accelerate the bottleneck operation of residue number system based hardware implementation of homomorphic encryption by 2.25 to 6.8 times.

- b) A custom hardware architecture of Number Theoretic Transform has been proposed for ASIC implementation. Number Theoretic Transform is a well-known technique in literature to accelerate polynomial multiplication, which is the most critical and computationally intensive operation in homomorphic encryption schemes as well as quantum-resistant ideal-lattice based cryptography. The proposed architecture possesses three unique features. First, the algorithmic properties of either Barrett or Montgomery multiplication algorithm are exploited to produce two simplified modulo multipliers. Second, to increase the throughput, two parallel half-sized dual-port memories are employed in the proposed design. Third, pipelined flip-flops are inserted into the butterfly circuits according to the access time of the memories. Compared to other FPGA hardware architectures directly ported to ASIC, for parameter sets targeting homomorphic encryption, the proposed custom architecture has achieved not only the lower latency but also at least 28.38% higher throughput at the cost of at most 7.5% hardware overhead.
- c) A scheme for outsourcing polynomial evaluation to untrusted parties without compromising privacy has been proposed. The proposed scheme is a hybrid solution combining the merits of distributed computing of residue number system and privacy preservation of fuzzy vault. The secret message is encoded as a residue representation with respect to a secret set. To conceal the secret message, chaff pairs, which are pairs of random modulus and residue that does not encode the secret message, are generated and added into the set of pairs representing the secret message. While fuzzy vault helps to secure the confidentiality of user's information, residue number system enables homomorphism and separates noise term from the message. The

constructed scheme eliminates the concern of noise growth and help to achieve small ciphertext size, as well as a better performance when compared to other state-of-the-art homomorphic encryption schemes. Candidates applications, including statistical functions and medical predictive analysis, are evaluated with the constructed scheme via polynomial approximation. Smaller ciphertext size due to noise growth isolation boosts up the performance of the proposed scheme compared to other homomorphic encryption schemes. It is noteworthy that for logistic regression model, the CPU implementation of the proposed scheme gains 5.34 times speedup over the implementation on a much more powerful CPU-GPU platform.

1.4 Organization of the Thesis

This thesis is organized into six chapters. Chapter 1 presents the motivation, the objectives as well as the major contributions of this research work.

Chapter 2 briefly describes the background of this research work and contains two major parts. In the first part, partially homomorphic encryption schemes, Gentry's homomorphic encryption scheme, and important major improvements following Gentry's scheme are discussed. Specifically, the construction, property of homomorphism, and computational hardness assumption behind the security are presented. In the second part, algorithms and architectures for hardware accelerators are discussed. Specifically, Number Theoretic Transform and Residue Number Systems are focused. In both parts, advantages and limitations of algorithmic construction of state-of-art homomorphic encryption schemes, as well as their corresponding hardware acceleration techniques, will be pointed out.

Major contributions of thesis are presented from Chapter 3 to Chapter 5. Chapter 3 is devoted to algorithm and hardware architecture for accelerating the conversion from residue-to-binary of very high cardinality arbitrary moduli set. A review of residue-to-binary conversion techniques in state-of-the-art hardware implementation of homomorphic encryption is presented, followed by an

analysis to find out the main drawback of these approaches. The concept of the proposed algorithm is introduced before its detailed description and an example is given for its demonstration. The hardware architecture for the proposed algorithm are also derived and discussed. The proposed hardware architecture is then compared with other works in terms of silicon area and delay, both theoretically and experimentally.

In Chapter 4, the performance of Number Theoretic Transform on ASIC is explored with a custom hardware architecture to accelerate polynomial multiplication in finite polynomial number ring. A review of the state-of-the-art algorithms and hardware architectures of Number Theoretic Transform is presented. The custom architecture is then presented. The proposed design is compared with other related works on the same ASIC technology under various parameter sets. After that, the performance gain of the proposed architecture is discussed.

In Chapter 5, a method that allows users to outsource a polynomial computation to another party without disclosing the user's inputs is proposed. A brief explanation about the poor performance of state-of-the-art homomorphic encryption schemes is provided. Followed by the fundamentals about fuzzy vault and residue number systems. The proposed scheme is then presented. Illustrative examples and security analysis are provided. Discussion on how the proposed scheme can be applied to several privacy-sensitive applications is given. The proposed scheme and candidate applications are implemented and compared with other works. The significant performance improvement provided by the proposed scheme is also justified by mathematical analysis.

Chapter 6 concludes this thesis by a summary of the major contributions achieved by the research conducted in this thesis. Based on the achieved results, several promising directions worthy of further exploration are also outlined.

Chapter 2

Background and Literature Review

2.1 Partially Homomorphic Encryption Schemes

Homomorphic encryption refers to encryption schemes where arbitrary computation can be performed directly on encrypted data without a secret key. It is proposed by Rivest et al. [16] after noticing the homomorphic properties of Rivest-Shamir-Adleman (RSA) scheme [17]. Since then, several schemes were proposed but could achieve only partial homomorphism, which severely limits their practicality. Partial homomorphism refers to homomorphism with respect to either addition or multiplication but not both. The key generation, encryption and decryption processes, as well as the hardness assumption and homomorphic property of the schemes described below are provided to illustrate this inherent limitation of early homomorphic encryption schemes.

2.1.1 RSA Scheme

The RSA scheme [16] is one of the first public-key cryptosystems, and is currently the most popular scheme used for key exchange and digital signature. This scheme derives its security from the hardness of integer factorization problem: multiplying two very large prime numbers is easy, but factoring the resulting product is considered infeasible even with today's supercomputers. RSA can be found in common internet protocols such as SSH (Secure Shell), S/MIME (Secure Multi-Purpose Internet Mail Extensions), TLS (Transport Layer Security), and SSL (Secure Sockets Layer).

Key generation:

- Two large prime numbers p and q are chosen.
- n is computed as the product of the chosen prime numbers, i.e., $n = pq$.
- The Euler totient function $\Phi(n) = (p-1)(q-1)$ is computed.

- A larger integer d is randomly selected such that the greatest common denominator of d and $\Phi(n)$ is one.
- The integer e is determined as the multiplicative inverse of d modulo $\Phi(n)$, i.e., $ed \equiv 1 \pmod{\Phi(n)}$.
- e and n are made public and d is kept private.

Encryption: Given a message m , its encryption into ciphertext c is computed as follows.

$$c = \text{Enc}(m) = m^e \pmod{n} \quad (2.1)$$

Decryption: Given a ciphertext c , its decryption is computed as follows.

$$m = \text{Dec}(c) = c^d \pmod{n} \quad (2.2)$$

Hardness assumption: The security of RSA scheme is based on the hardness of integer factorization problem. Given n , there exists no efficient algorithm to factor it to recover p and q .

Homomorphic property: Given two messages, m_1 and m_2 , and their corresponding encryption, $c_1 = \text{Enc}(m_1)$ and $c_2 = \text{Enc}(m_2)$, respectively, the following equation holds.

$$c_1 c_2 = m_1^e m_2^e \pmod{n} = (m_1 m_2)^e \pmod{n} \quad (2.3)$$

Thus, $c_1 c_2 \pmod{n} = \text{Enc}(m_1 m_2)$. The encryption of the product of two messages can be determined by the multiplication of their corresponding ciphertexts. For this reason, RSA scheme is said to exhibit homomorphism with respect to multiplication.

2.1.2 Benaloh Scheme

This scheme was proposed by Josh Benaloh in 1994 based on the hardness of the higher residuosity problem. The weakness of this scheme is the cost of its

decryption. However this weakness is acceptable for applications such as cryptographic voting, which requires many encryptions of ballots and homomorphic addition operations for tallying, but only one decryption determines the final tally.

Key generation:

- A block size r and two large prime numbers, p and q , are chosen such that r divides $p-1$ and $\gcd(r, (p-1)/r) = \gcd(r, q-1) = 1$, where $\gcd$ denotes the greatest common denominator function.
- $n = pq$ and $\Phi = (p-1)(q-1)$ are computed.
- Let Z_n^* be the multiplicative group of integers modulo n . An integer y is randomly selected from Z_n^* such that $y < r$ and $\gcd(y, r) = 1$.
- $x = y^{\Phi/r} \bmod n$ is computed.
- y and n are made public and x and Φ are kept private.

Encryption: For a message $m \in [0, r)$, its encryption is computed as follows.

$$c = \text{Enc}(m) = y^m u^r \bmod n \quad (2.4)$$

where u is chosen randomly from Z_n^* .

Decryption: Given a ciphertext c , it is decrypted as follows.

$$m = \text{Dec}(c) = \log_x(c^{\Phi/r}) \quad (2.5)$$

i.e., m is found such that $x^m \equiv c^{\Phi/r} \bmod n$.

Hardness assumption: The security of Belanoh scheme is based on the hardness of higher residuosity problem. Specifically, there exists no efficient algorithm to determine if there is an integer x such that $x^r \equiv a \bmod n$, given r , a and n .

Homomorphic property: Given two messages, m_1 and m_2 , and their corresponding encryption, $c_1 = \text{Enc}(m_1)$ and $c_2 = \text{Enc}(m_2)$, respectively, the

following equation holds.

$$c_1 c_2 = (y^{m_1} u_1^r) (y^{m_2} u_2^r) = y^{m_1+m_2} (u_1 u_2)^r \mod n \quad (2.6)$$

Thus, $c_1 c_2 \mod n = \text{Enc}(m_1 + m_2)$. The encryption of the sum of two messages can be determined by the multiplication of their corresponding ciphertexts. Therefore, Belanoh scheme is said to exhibit homomorphism with respect to addition.

2.1.3 Remark

Besides RSA scheme and Belanoh scheme, there are also other schemes that express partial homomorphism [18], [19], [21]. These early homomorphic cryptosystems have their ciphertext in the form of b^e , where b and e are the base and the exponent, respectively. They achieve homomorphism with respect to addition or multiplication when the message is embedded in the exponent or the base, respectively, but they cannot achieve homomorphism with respect to addition and multiplication at the same time. To overcome this drawback, modern homomorphic cryptosystems have the ciphertext as a linear function of the message, as shown in subsequent sections.

2.2 Gentry's Integer-based Fully Homomorphic Encryption Scheme

Although Gentry's construction is the first ever scheme to achieve homomorphism with respect to both addition and multiplication [23], the complexity of its ideal-lattice based origin obscures a good understanding of its underlying concept. Thus, this section adopts the scheme from [41], also proposed by Gentry, for a simpler explanation on how to design a homomorphic scheme. The scheme from [41] can be considered as an integer version of the scheme from [23], and does not require knowledge of ideal-lattice to understand it. The process of designing a homomorphic scheme consists of three steps. Firstly, a symmetric-key somewhat homomorphic encryption is constructed where a ciphertext exhibits linearity with respect to a message. Secondly the

symmetric scheme is turned into a public key scheme. Lastly, it was made fully homomorphic by squashing its decryption circuit.

2.2.1 Symmetric-key Somewhat Homomorphic Encryption Scheme

Key generation: Choose an odd η -bit integer p as the secret key.

Encryption: For a bit $m \in \{0, 1\}$, its encryption is computed as follows.

$$c = \text{Enc}(m) = pq + 2r + m \quad (2.7)$$

where the integers, q and r , are randomly chosen in predetermined intervals such that $2r + m \ll p/2$.

Decryption: Given a ciphertext c , it is decrypted as follows.

$$m = \text{Dec}(c) = (c \bmod p) \bmod 2 \quad (2.8)$$

The equation for decryption holds since :

$$\begin{aligned} & (c \bmod p) \bmod 2 \\ &= ((pq + 2r + m) \bmod p) \bmod 2 \\ &= (2r + m) \bmod 2 \quad (\text{since } 2r + m \ll p) \\ &= m \end{aligned} \quad (2.9)$$

Example: Let $p = 31$ and $m = 1$

$$c = \text{Enc}(m) = pq + 2r + m = 31 \times 3 + 2 \times 2 + 1 = 98 \quad (2.10)$$

where $q = 3$ and $r = 2$.

$m = 1$ can be recovered by applying the decryption equation on $c = 98$ as follows:

$$\begin{aligned} m &= \text{Dec}(c) = (c \bmod p) \bmod 2 \\ &= (98 \bmod 31) \bmod 2 \\ &= 5 \bmod 2 \\ &= 1 \end{aligned} \quad (2.11)$$

Homomorphic properties: Given two messages, m_1 and m_2 , and their

corresponding encryption, $c_1 = \text{Enc}(m_1)$ and $c_2 = \text{Enc}(m_2)$, respectively, the following equation holds.

$$c_1 + c_2 = p(q_1 + q_2) + 2(r_1 + r_2) + (m_1 + m_2) \quad (2.12)$$

$$\begin{aligned} c_1 \times c_2 = & p(pq_1q_2 + 2q_1r_2 + 2q_2r_1 + m_1q_2 + m_2q_1) \\ & + 2(2r_1r_2 + m_1r_2 + m_2r_1) + m_1m_2 \end{aligned} \quad (2.13)$$

If $2(r_1 + r_2) + (m_1 + m_2) \ll p/2$, the sum of the plaintexts $(m_1 + m_2) \bmod 2$ can be recovered by computing $((c_1 + c_2) \bmod p) \bmod 2$. Therefore, this scheme is said to exhibit homomorphism with respect to addition.

If $2(2r_1r_2 + m_1r_2 + m_2r_1) + m_1m_2 \ll p/2$, the product of the plaintexts $(m_1 \times m_2) \bmod 2$ can be recovered by computing $((c_1 \times c_2) \bmod p) \bmod 2$. Therefore, this scheme is said to exhibit also homomorphism with respect to multiplication.

Noted that after a number of ciphertext multiplications and additions, the noise term in a ciphertext, i.e., $2r + m$, can grow to an extent, i.e., $p/2$, that invalidates the decryption equation. Thus, to be more precise, this scheme actually exhibit somewhat homomorphism. Noise growth is much more severe in ciphertext multiplication than ciphertext addition, since noise only doubles on a ciphertext addition, but squares on a ciphertext multiplication.

2.2.2 Public-key Somewhat Homomorphic Encryption Scheme

The symmetric scheme is turned into a public key scheme as follows: the public key is a set of integers $x_i = pq_i + 2r_i$, where q_i and r_i are chosen from predetermined intervals. Each integer in the public key can be viewed as an encryption of zero. To encrypt a bit m , the ciphertext is set as m plus a subset sum of the integers in the public key. The detailed construction of this public key scheme is shown below.

Parameters: Υ is the bit-length of the integers in the public key, τ is the number of integers in the public key, η is the bit-length of the secret key, ρ and ρ' are the bit-lengths of the noise. These parameters, according to [41], must be set under some specific constraints. Let λ be the security parameter. For convenience, the recommended settings given by [41] are $\rho = \lambda$, $\rho' = 2\lambda$, $\eta = O(\lambda^2)$, $\Upsilon = O(\lambda^5)$ and $\tau = \Upsilon + \lambda$.

Key generation: A η -bit odd integer p is chosen to be the secret key. For $i = 0, \dots, \tau$, $x_i = pq_i + r_i$ is set, where q_i is an integer in $[0, 2^\Upsilon/p)$ and r_i is an integer in $(-2^\rho, 2^\rho)$. The index is relabeled such that x_0 is the largest. The key generation process is restarted unless x_0 is odd and r_0 is even. The public key is $pk = \langle x_0, x_1, \dots, x_\tau \rangle$.

Encryption: To encrypt a bit $m \in \{0, 1\}$, a random subset $S \subseteq \{1, 2, \dots, \tau\}$ and a random integer $r \in (-2^{\rho'}, 2^{\rho'})$ are chosen to output

$$c = \text{Enc}(m) = m + 2r + 2 \sum_{i \in S} x_i \bmod x_0 \quad (2.14)$$

Decryption: Given a ciphertext c , it is decrypted as follows.

$$m = (c \bmod p) \bmod 2 \quad (2.15)$$

The equation for decryption holds since :

$$\begin{aligned} c &= m + 2r + 2 \sum_{i \in S} x_i \bmod x_0 \\ &= m + 2r + 2 \sum_{i \in S} (pq_i + r_i) - k(pq_0 + r_0) \text{ for some integer } k \quad (2.16) \\ &= m + (2r + 2 \sum_{i \in S} r_i - kr_0) + p(2 \sum_{i \in S} q_i - kq_0) \end{aligned}$$

Since r_0 is even, m can be simply recovered by computing $(c \bmod p) \bmod 2$.

The decryption equation can be simplified as follows:

$$\begin{aligned}
m &= (c \bmod p) \bmod 2 \\
&= (c - p \times \lceil c / p \rceil) \bmod 2 \\
&= (c \bmod 2) \text{ XOR } (\lceil c / p \rceil \bmod 2)
\end{aligned} \tag{2.17}$$

where $\lceil \cdot \rceil$ denotes rounding to the nearest integer.

Hardness assumption: The security of the symmetric and asymmetric schemes relies on the hardness of approximate greatest common denominator problem. It states that given $\{x_1, x_2, \dots, x_t\}$ where each $x_i = pq_i + r_i$ is close to a multiple of p , finding p is computationally infeasible.

Example: The asymmetric-key scheme has the same homomorphic properties as the symmetric-key scheme. Addition and multiplication modulo 2 in the plaintext domain correspond to addition and multiplication modulo x_0 in the ciphertext domain, respectively. Let the secret key $p = 10001$.

$\langle q_0, q_1, q_2, q_3 \rangle = \langle 41, 29, 31, 37 \rangle$ and $\langle r_0, r_1, r_2, r_3 \rangle = \langle 16, 7, 11, 13 \rangle$ is set. Following the key generation procedure, the public key is computed as $pk = \langle x_0, x_1, x_2, x_3 \rangle = \langle 410057, 290036, 310042, 370050 \rangle$.

Let $m_1 = 0$. Take $r_1 = 11$, m_1 is encrypted as

$$c_1 = m_1 + 2r_1 + 2 \sum_{i \in \{2,3\}} x_i \bmod x_0 = 130035 \tag{2.18}$$

Let $m_2 = 1$. Take $r_2 = 13$, m_2 is encrypted as

$$c_2 = m_2 + 2r_2 + 2 \sum_{i \in \{1,3\}} x_i \bmod x_0 = 90028 \tag{2.19}$$

Thus, $c_1 + c_2 \bmod x_0 = 226063$ and $c_1 \times c_2 \bmod x_0 = 73687$. By decrypting these two values, 1 and 0 corresponding to $(m_1 + m_2) \bmod 2$ and $(m_1 \times m_2) \bmod 2$ are obtained, respectively.

2.2.3 Public-key Fully Homomorphic Encryption Scheme

It is important to have homomorphic properties with respect to addition/multiplication modulo 2. Addition/multiplication modulo 2 is equivalent to taking XOR/AND of the input bits. Thus, if an encryption scheme is homomorphic with respect to addition/multiplication modulo 2, it can homomorphically evaluate XOR/AND over the encrypted input bits. Since an arbitrary function can be expressed as a Boolean circuit constructed from XOR/AND gates, we can homomorphically evaluate that function using only homomorphic operations.

Unfortunately, the public key scheme described above cannot handle every arbitrary function. As discussed previously, after a number of homomorphic XOR (i.e., ciphertext addition) and homomorphic AND (i.e., ciphertext multiplication), decryption may not hold. The growth in noise places a limitation on the functions for which this public-key scheme can be correctly evaluated. Specifically, let C be a Boolean circuit with t inputs and d be the degree of the multivariate Boolean polynomial computed by C . Let t, τ be bounded by λ^β . With the recommended parameters set in Section 2.2.2, the public key scheme can correctly evaluate C on the ciphertext space as long as $d \leq \alpha \lambda \log^2 \lambda$ for some constants α and β (see Lemma 3.5 in [41]).

The trick to achieve fully homomorphic encryption, i.e., infinite number of ciphertext additions and multiplications, is: if the decryption equation of the described scheme can be expressed as a shallow Boolean circuit, i.e., the degree of its polynomial is less than d , then it can be homomorphically evaluated. The result of homomorphically decrypting a noisy ciphertext is a ciphertext with much less noise. By performing homomorphic decryption when the noise in ciphertexts reaches a certain level, the number of homomorphic operations is unlimited, thus enables homomorphic evaluation of any arbitrary function.

However, in the described scheme, the complexity of division by p in the

decryption equation $m = (c \bmod 2) \text{ XOR } ([c/p] \bmod 2)$ makes it impossible to express it as a low depth Boolean circuit for homomorphic evaluation. Thus, to achieve full homomorphism, it is required to squash the decryption circuit, i.e., to modify it in such a way that it can be expressed as a low depth circuit. In [41], this is accomplished by adding extra information about $[c/p]$ in the ciphertexts. The construction of the squashed scheme are described as follows.

Parameters: The parameters are the same as the public key scheme. κ , θ , and Θ are three additional parameters, which are functions of λ .

Key generation: $sk^* = p$ and pk^* are generated as before. $x_p = [2^\kappa/p]$. A Θ -bit vector with Hamming weight θ and $s = \langle s_1, s_2, \dots, s_\Theta \rangle$ are selected at random. Let S be the set of index i such that $s_i = 1$. Θ random positive integers $u_i < 2^{\kappa+1}$ are selected such that $\sum_{i \in S} u_i = x_p \pmod{2^{\kappa+1}}$. Let $y = \langle y_1, y_2, \dots, y_\Theta \rangle$, where each $y_i = u_i / 2^\kappa$ is a positive number with κ -bit precision after the binary point. Hence $\sum_{i \in S} y_i \bmod 2 \approx 1/p$. $sk = s$ and $pk = (pk^*, y)$ are output as the secret and public key, respectively.

Encryption: A ciphertext c^* is generated as before. $z_i = c^* \times y_i \bmod 2$ is set for $i = 1, 2, 3, \dots, \Theta$, and $\lceil \log \theta \rceil + 3$ bits of precision after the binary point are kept for each z_i . Both c^* and $z = \langle z_1, z_2, \dots, z_\Theta \rangle$ are output.

Decryption: The plaintext m is computed by

$$m = \left(c^* - \left[\sum s_i z_i \right] \right) \bmod 2 \quad (2.20)$$

The equation for decryption holds since :

$$\begin{aligned}
& (c^* \bmod p) \bmod 2 \\
&= (c^* - p \times \lceil c^* / p \rceil) \bmod 2 \\
&= (c^* - \lceil c^* / p \rceil) \bmod 2 \quad (\text{since } p \text{ is an odd integer}) \\
&= \left(c^* - \left\lceil c^* \times \sum_{i \in S} y_i \right\rceil \right) \bmod 2 \quad \left(\text{since } \sum_{i \in S} y_i \approx 1/p \right) \quad (2.21) \\
&= \left(c^* - \left\lceil \sum_{i \in S} s_i z_i \right\rceil \right) \bmod 2 \quad (z_i = c^* \times y_i) \\
&= \left(c^* - \left\lceil \sum s_i z_i \right\rceil \right) \bmod 2
\end{aligned}$$

Hardness assumption: Besides the approximate greatest common denominator problem, the fully homomorphic encryption scheme relies on the hardness of sparse subset sum problem since it provides a hint about the secret key in the public key. The sparse subset sum problem states that given a set of integers, it is infeasible to determine a non-empty subset whose sum is zero.

Example: Let the secret key $p = 10001$. Similar to the example in Section 2.2.2, $pk^* = \langle 410057, 290036, 310042, 370050 \rangle$. With $\kappa = 31$,

$$x_p = \lceil 2^\kappa / p \rceil = \lceil 2^{31} / 10001 \rceil = 214727 \quad (2.22)$$

Let $s = \langle s_1, s_2, \dots, s_8 \rangle = \langle 1, 0, 0, 1, 0, 0, 1, 1 \rangle$ be an 8-bit vector with a Hamming weight of 4.

Let $u = \langle u_1, u_2, \dots, u_8 \rangle$, where $u_1 = 1073741824$, $u_2 = 3735307509$, $u_3 = 230156356$, $u_4 = 1196956551$, $u_5 = 2982501561$, $u_6 = 925691835$, $u_7 = 1061964157$, and $u_8 = 962519491$.

Noted that $\sum u_i$ such that $s_i = 1$ works out to be

$$\begin{aligned}
& (1073741824 + 1196956551 + 1061964157 + 962519491) \\
&= 214727 \pmod{2^{32}} \quad (2.23)
\end{aligned}$$

Therefore, $y = \langle y_1, y_2, \dots, y_8 \rangle$, where $y_1 = 0.5$, $y_2 = 1.73938810313$, $y_3 = 0.10717490501$, $y_4 = 0.55737632839$, $y_5 = 1.38883551629$, $y_6 = 0.43105885153$, $y_7 = 0.49451559642$, and $y_8 = 0.44820806523$. The public key consists of pk^* and y .

Noted that $\sum y_i \bmod 2$ such that $s_i = 1$ is equal to 0.00009999, which is closely approximated to $1/p \bmod 2$.

Let c^* be c_1 from the example in Section 2.2.2. Thus, $c^* = 130035$, which is the encryption of 0. $z_i = c^* \times y_i \bmod 2$ for $i = 1, 2, 3, \dots, 8$ are computed as follows:

$$\begin{aligned} z_1 &= 1.5 \\ z_2 &= 1.331990509 \\ z_3 &= 0.4887729753 \\ z_4 &= 0.4308621936 \\ z_5 &= 1.22636077 \\ z_6 &= 0.7377587035 \\ z_7 &= 0.3355804747 \\ z_8 &= 0.735762183 \end{aligned} \tag{2.24}$$

The ciphertext will compose of c^* and $z = \langle z_1, z_2, \dots, z_8 \rangle$. To decrypt this ciphertext,

$$\begin{aligned} &\left(c^* - \left[\sum s_i z_i \right] \right) \bmod 2 \\ &= 130035 - [1.5 + 0.4308621936 + 0.3355804747 + 0.735762183] \bmod 2 \\ &= 130035 - [3.002204851] \bmod 2 \\ &= 130035 - 3 \bmod 2 \\ &= 0 \end{aligned} \tag{2.25}$$

The Boolean circuit of the decryption equation consists of three parts. In the first part, $s_i z_i$ ($s_i z_i = z_i$ when $s_i = 1$ and $s_i z_i = 0$ otherwise) can be computed by 1-level sub-circuit of multiplication gates. In the second part, the sum $\left[\sum s_i z_i \right]$ can

be computed using a shallow Boolean circuit by carry-save addition technique. In the third part, c^* and $\left[\sum s_i z_i\right]$ modulo 2 are summed. The Boolean circuit of the decryption equation has its degree at most $64\lambda \log^2 \lambda$ (see [41] for proof). Thus it can be homomorphically evaluated.

2.3 State-of-the-art Homomorphic Encryption Schemes

The scheme presented in Section 2.2 relies on the unproven hardness assumption of sparse subset sum problem. Besides, it suffers from quadratic noise growth for every homomorphic multiplication. State-of-the-art homomorphic encryption schemes rely on the (ring) learning-with-errors problem, which was proven to be as hard as the worst-case problem in lattice [42], [43]. More importantly, they feature techniques to reduce noise growth from quadratic to linear. In this section, the (ring) learning-with-errors problem is first introduced, followed by two important schemes with different noise management techniques, which are further clarified by a subsequent remark. A compact listing of other schemes based on the (ring) learning-with-errors problem can be found in [44] and [45].

2.3.1 Learning-with-Errors and Ring Learning-with-Errors Problems

Let the secret s be a n -dimensional integer vector with its elements belong to Z_q , i.e., the finite ring of integers modulo q . Given the following system of linear equations:

$$\begin{aligned} b_1 &= \langle a_1, s \rangle + e_1 \in Z_q \\ b_2 &= \langle a_2, s \rangle + e_2 \in Z_q \\ b_3 &= \langle a_3, s \rangle + e_3 \in Z_q \\ &\vdots \end{aligned} \tag{2.26}$$

where each a_i is an n -dimensional integer vector chosen uniformly from Z_q^n . $\langle a_i, s \rangle = \sum_j (a_i)_j s_j$ denotes the inner product modulo q of s and a_i . Each noise term e_i is randomly sampled from a distribution such that its absolute value is

much smaller than q . The learning-with-errors problem states that given the knowledge of a_i and b_i , it is computationally infeasible to determine s .

random $\mathbb{Z}_{13}^{7 \times 4}$		secret $\mathbb{Z}_{13}^{4 \times 1}$		small noise $\mathbb{Z}_{13}^{7 \times 1}$		$\mathbb{Z}_{13}^{7 \times 1}$
4		6		0		4
5		9		-1		7
3	×	11	+	1	=	2
1		11		1		11
12				1		5
6				0		12
3				-1		8

Figure 2.1: An example of learning-with-errors problem with $n = 4$ and $q = 13$.

An example is taken from [46] to illustrate the learning-with-errors problem in Figure 2.1. The right-most blue column vector is a result of the product of the left-most blue matrix with the secret red column vector, marked with small noise yellow column vector. Learning-with-errors problem states that it is computationally infeasible to find the secret red column vector, given the blue left-most matrix and right-most column vector.

A simple public-key encryption scheme whose security is based on the hardness of learning-with-errors problem is given below [42].

Key generation: The private key s is chosen uniformly from $\mathbb{Z}_q^n$. m vectors a_i for $i = 1, 2, \dots, m$ are chosen uniformly from $\mathbb{Z}_q^n$. m integers e_i for $i = 1, 2, \dots, m$ are sampled from a distribution such that $e_i \ll q$. The public key is $(a_i, b_i)_{i=1}^m$, where $b_i = \langle a_i, s \rangle + e_i$.

Encryption: To encrypt a binary message, a set S that is a subset of the set $\{1,2,3,\dots,m\}$ is uniformly selected. The ciphertext is computed as:

$$\left(\sum_{i \in S} a_i, m \left\lfloor \frac{p}{2} \right\rfloor + \sum_{i \in S} b_i \right) \quad (2.27)$$

where $\lfloor \cdot \rfloor$ denotes rounding down to the nearest integer.

Decryption: Given a ciphertext in the form of (a, b) , 0 is output if $b - \langle as \rangle$ is closer to 0 than $\left\lfloor \frac{p}{2} \right\rfloor$, otherwise 1 is output.

There are several undesired inefficiencies of cryptosystems based on learning-with-error problems. Firstly, having one more element in the private key means having m more elements in the public key. Secondly, the complexity of the size of the public key is quadratic in terms of the size of the private key. Thirdly, key generation and decryption involve vector multiplication whose computation complexity is quadratic.

Ring learning-with-errors overcomes the above inefficiencies by having each a_i , for $i = 2, 3, \dots, m$, as a negative cyclic shift of a_1 [43]. Thus, instead of keeping all a_i 's, only the first element needs to be kept. This corresponds to changing from sampling a_1 from Z_q^n to the polynomial ring $Z_q[x]/(x^n + 1)$. An example taken from [46] to illustrate the negative cyclic shift is shown in Figure 2.2. In this example, each row vector is a cyclic shift of the row vector above it with a special wrapping rule: the right-most element of the row vector above it is mapped to its negation modulo 13. This corresponds to the finite ring of integer polynomials $Z_{13}[x]/(x^4 + 1)$. By taking advantage of the negative cyclic shift, ring learning-with-errors problem offers three advantages compared to learning-with-errors problem. Firstly, having one more element in the private key means having one, instead of m , more elements in the public key. Secondly, the complexity of the size of the public key is reduced from quadratic to linear in terms of the size of the private key. Thirdly, key generation and decryption involve polynomial

multiplication instead of vector multiplication, and the computation complexity is reduced from quadratic to quasilinear.

random
 $\mathbb{Z}_{13}^{7 \times 4}$

4	1	11	10
3	4	1	11
2	3	4	1
12	2	3	4
9	12	2	3
10	9	12	2
11	10	9	12

Figure 2.2: An example of negacyclic shift in ring learning-with-errors problem with $n = 4$ and $q = 13$.

Let $R = \mathbb{Z}_q[x]/(x^n + 1)$ be the ring of integer polynomials modulo $x^n + 1$ and q , where n is a power of 2 and q is a large prime number such that $q \equiv 1 \pmod{2n}$. An element in R can be represented by a polynomial of degree less than n with the coefficients belong to $\{0, 1, 2, \dots, q-1\}$. Ring learning-with-errors problem states that if s is a polynomial having small coefficients uniformly chosen from R , i.e., given $(a, b = as + e)$, where a is randomly sampled from R and e is a small polynomial sampled from an error distribution, it is computationally infeasible to determine s . An example taken from [46] is shown in Figure 2.3 to illustrate the ring learning-with-errors problem. In this example, the bottom-most blue polynomial is a result of the product of the top-most blue polynomial with the secret red polynomial, marked with small noise yellow polynomial. Ring learning-with-errors problem states that it is computationally infeasible to find the secret red polynomial, given the blue top-most polynomial and bottom-most polynomial.

	$4 + 1x + 11x^2 + 10x^3$	random
×	$6 + 9x + 11x^2 + 11x^3$	secret
+	$0 - 1x + 1x^2 + 1x^3$	small noise
=	$10 + 5x + 10x^2 + 7x^3$	

Figure 2.3: An example of ring learning-with-errors problem in polynomial ring $\mathbb{Z}_{13}[x]/(x^4+1)$.

2.3.2 Homomorphic Encryption Schemes from Ring Learning-with-errors

2.3.2.1 Fan and Vercauteren (FV) Somewhat Fully Homomorphic Cryptosystem

This scheme [47] works in the integer polynomial ring $R = \mathbb{Z}[x]/(x^n + 1)$ where n is a power of two. It is described in this section with some simplification.

Parameters: Let t and q be the plaintext modulus and ciphertext modulus, respectively, such that t is much smaller than q . Let χ be a distribution over R such that it outputs only polynomials with small norm.

Key generation: A polynomial s is sampled from χ . Let the secret key $sk = s$. a is sampled from R_q and e is sampled from χ . The public key is given by $pk = (as + e \bmod q, -a \bmod q)$.

Encryption: given a message $m \in R_t$, u , e_1 and e_2 are sampled from χ and the ciphertext is computed by

$$ct = \left(p_1 u + e_1 + \left\lfloor \frac{q}{t} \right\rfloor m \bmod q, p_2 u + e_2 \bmod q \right) \quad (2.28)$$

where p_1 and p_2 are the first and second elements of the public key, respectively. $\lfloor \cdot \rfloor$ denotes rounding down to the nearest integer.

Decryption: Given the secret key $sk = s$ and a ciphertext ct , the plaintext is computed by

$$\left\lfloor \frac{t(c_1 + c_2 s \bmod q)}{q} \right\rfloor \bmod t \quad (2.29)$$

where c_1 and c_2 are the first and second elements, respectively of the ciphertext ct . $\lfloor \cdot \rfloor$ denotes rounding to the nearest integer.

Homomorphism properties: Note that in the decryption equation

$$c_1 + c_2 s \bmod q = \left\lfloor \frac{q}{t} \right\rfloor m + E \quad (2.30)$$

where $E = eu + e_1 + e_2 s$ is the noise contained in the ciphertext ct .

By treating the elements of the ciphertext ct as the coefficients of the polynomial $ct(x)$,

$$ct(s) \bmod q = \left\lfloor \frac{q}{t} \right\rfloor m + E \quad (2.31)$$

Let ct_1 and ct_2 be the encryption of m_1 and m_2 , respectively. It is easy to see that:

$$ct_1(s) + ct_2(s) = \left\lfloor \frac{q}{t} \right\rfloor (m_1 + m_2) + (E_1 + E_2) \quad (2.32)$$

Thus, homomorphic addition is defined as

$$H.Add(ct_1, ct_2) = (c_{1,1} + c_{2,1} \bmod q, c_{1,2} + c_{2,2} \bmod q) \quad (2.33)$$

where $c_{1,1}$ and $c_{1,2}$ are the first and second elements of ct_1 , respectively. $c_{2,1}$ and $c_{2,2}$ are the first and second elements of ct_2 , respectively.

Multiplying $ct_1(s)$ and $ct_2(s)$ gives:

$$\begin{aligned}
ct_1(s) \times ct_2(s) &= \left(\left\lfloor \frac{q}{t} \right\rfloor \right)^2 m_1 m_2 + \left\lfloor \frac{q}{t} \right\rfloor (m_1 E_2 + m_2 E_1) + E_1 E_2 \\
&= c_{1,2} c_{2,2} s^2 + (c_{2,2} c_{1,1} + c_{1,2} c_{2,1}) s + c_{1,1} c_{2,1}
\end{aligned} \tag{2.34}$$

From Equations (2.30) and (2.34), to recover $m_1 m_2$, Equation (2.34) needs to be scaled by t/q . Ignoring the error due to the approximation,

$$\begin{aligned}
\frac{t}{q} ct_1(s) \times ct_2(s) &= \left\lfloor \frac{q}{t} \right\rfloor m_1 m_2 + (m_1 E_2 + m_2 E_1) + \frac{t}{q} E_1 E_2 \\
&= \frac{t}{q} (c_{1,2} c_{2,2} s^2 + (c_{2,2} c_{1,1} + c_{1,2} c_{2,1}) s + c_{1,1} c_{2,1})
\end{aligned} \tag{2.35}$$

Noted that the decryption equation is a linear polynomial in term of s while Equation (2.35) is a quadratic polynomial in s . To overcome this problem, the authors of [47] proposed to provide a ciphertext called re-linearization key that encrypts s^2 marked with some noise.

$$rlk = (a_0 s + e_0 + s^2 \bmod q, -a_0 \bmod q) \tag{2.36}$$

where a_0 is uniformly chosen from R_q and e_0 is sampled from χ .

Let ct' be a ciphertext defined as follows.

$$ct' = \begin{pmatrix} \frac{t}{q} (c_{1,2} c_{2,2} rlk_1 + c_{1,1} c_{2,1}) \bmod q, \\ \frac{t}{q} (c_{1,2} c_{2,2} rlk_2 + c_{2,2} c_{1,1} + c_{1,2} c_{2,1}) \bmod q \end{pmatrix} \tag{2.37}$$

where rlk_1 and rlk_2 are the first and second elements, respectively of the re-linearization key.

Evaluating $ct'(s)$ gives:

$$\begin{aligned}
ct'(s) &= \frac{t}{q} (c_{1,2} c_{2,2} (rlk_2 s + rlk_1) + (c_{2,2} c_{1,1} + c_{1,2} c_{2,1}) s + c_{1,1} c_{2,1}) \\
&= \frac{t}{q} (c_{1,2} c_{2,2} (s^2 + e_0) + (c_{2,2} c_{1,1} + c_{1,2} c_{2,1}) s + c_{1,1} c_{2,1}) \\
&= \left\lfloor \frac{q}{t} \right\rfloor m_1 m_2 + (m_1 E_2 + m_2 E_1) + \frac{t}{q} (E_1 E_2 + c_{1,2} c_{2,2} e_0)
\end{aligned} \tag{2.38}$$

As long as the noise in ct' is small enough, i.e., $(m_1E_2 + m_2E_1) + \frac{t}{q}(E_1E_2 + c_{1,2}c_{2,2}e_0) \ll \frac{q}{2t}$, m_1m_2 can be recovered by the decryption equation.

The homomorphic multiplication is then defined as:

$$H.Mul(ct_1, ct_2, rlk) = \begin{pmatrix} \frac{t}{q}(c_{1,2}c_{2,2}rlk_1 + c_{1,1}c_{2,1}) \bmod q, \\ \frac{t}{q}(c_{1,2}c_{2,2}rlk_2 + c_{2,2}c_{1,1} + c_{1,2}c_{2,1}) \bmod q \end{pmatrix} \quad (2.39)$$

2.3.2.2 Brakerski, Gentry, and Vaikuntanathan Fully Homomorphic Encryption Without Bootstrapping (BGV) Cryptosystem

This scheme [29] also operates in the integer polynomial ring $R = \mathbb{Z}[x]/(x^n + 1)$, where n is a power of two. It is described in this section with some simplification.

Parameters: Let t and q be the plaintext modulus and ciphertext modulus, respectively, such that t is much smaller than q . Let χ be a distribution over R such that it outputs only polynomials with small norm.

Key generation: A polynomial s is sampled from χ . Let the secret key $sk = s$. a is sampled from R_q and e is sampled from χ . The public key is computed as $pk = (as + te \bmod q, -a \bmod q)$.

Encryption: Given a message $m \in R_t$, u , e_1 and e_2 are sampled from χ and the ciphertext is computed by

$$ct = (p_1u + te_1 + m \bmod q, p_2u + te_2 \bmod q) \quad (2.40)$$

where p_1 and p_2 are the first and second elements, respectively of the public key.

Decryption: Given the secret key $sk = s$ and a ciphertext ct , the plaintext is computed by

$$(c_1 + c_2 s \bmod q) \bmod t \quad (2.41)$$

where c_1 and c_2 are the first and second elements, respectively of the ciphertext ct .

Homomorphism properties: Note that in the decryption equation

$$c_1 + c_2 s \bmod q = m + tE \quad (2.42)$$

where $E = eu + e_1 + e_2 s$ is the noise contained in the ciphertext ct .

By treating the elements of the ciphertext ct as the coefficients of the polynomial $ct(x)$:

$$ct(s) \bmod q = m + tE \quad (2.43)$$

Let ct_1 and ct_2 be the encryption of m_1 and m_2 , respectively. It is easy to see that

$$ct_1(s) + ct_2(s) = (m_1 + m_2) + t(E_1 + E_2) \quad (2.44)$$

Thus, the homomorphic addition is defined as

$$H.Add(ct_1, ct_2) = (c_{1,1} + c_{2,1} \bmod q, c_{1,2} + c_{2,2} \bmod q) \quad (2.45)$$

where $c_{1,1}$ and $c_{1,2}$ are the first and second elements of ct_1 , respectively. $c_{2,1}$ and $c_{2,2}$ are the first and second elements of ct_2 , respectively.

Multiplying $ct_1(s)$ and $ct_2(s)$ gives:

$$\begin{aligned} ct_1(s) \times ct_2(s) &= m_1 m_2 + t(m_1 E_2 + m_2 E_1) + t^2 E_1 E_2 \\ &= c_{1,2} c_{2,2} s^2 + (c_{2,2} c_{1,1} + c_{1,2} c_{2,1}) s + c_{1,1} c_{2,1} \end{aligned} \quad (2.46)$$

Similar to FV cryptosystem, Equation (2.46) is a quadratic polynomial in s while the decryption equation is a linear polynomial in s . To overcome this problem, the usage of re-linearization key that encrypts s^2 marked with some noise, similar to FV cryptosystem, is employed.

$$rlk = (a_0 s + t e_0 + s^2 \bmod q, -a_0 \bmod q) \quad (2.47)$$

where a_0 is uniformly chosen from R_q and e_0 is sampled from χ .

Let ct' be a ciphertext defined as follows.

$$ct' = \begin{pmatrix} (c_{1,2} c_{2,2} rlk_1 + c_{1,1} c_{2,1}) \bmod q, \\ (c_{1,2} c_{2,2} rlk_2 + c_{2,2} c_{1,1} + c_{1,2} c_{2,1}) \bmod q \end{pmatrix} \quad (2.48)$$

where rlk_1 and rlk_2 are the first and second elements, respectively of the re-linearization key.

Evaluating $ct'(s)$ gives:

$$\begin{aligned} ct'(s) &= (c_{1,2} c_{2,2} (rlk_2 s + rlk_1) + (c_{2,2} c_{1,1} + c_{1,2} c_{2,1}) s + c_{1,1} c_{2,1}) \\ &= (c_{1,2} c_{2,2} (s^2 + t e_0) + (c_{2,2} c_{1,1} + c_{1,2} c_{2,1}) s + c_{1,1} c_{2,1}) \\ &= m_1 m_2 + t(m_1 E_2 + m_2 E_1) + t(t E_1 E_2 + c_{1,2} c_{2,2} e_0) \end{aligned} \quad (2.49)$$

As long as the noise in ct' is small enough, i.e., $t(m_1 E_2 + m_2 E_1) + t(t E_1 E_2 + c_{1,2} c_{2,2} e_0) \ll \frac{q}{2}$, $m_1 m_2$ can be recovered by the decryption equation.

Let $(q_L, q_{L-1}, q_{L-2}, \dots, q_0)$ be a decreasing sequence of number such that q_i is double in bit-length compared to q_{i-1} for $i = 1, 2, 3, \dots, L$. Let ct_1 be a valid ciphertext encrypting m_1 . If ct_1 is scaled by $\frac{q_{i-1}}{q_i}$ to obtain ct_2 , then under some specific conditions, the following equality holds

$$(ct_1(s) \bmod q_i) \bmod t = (ct_2(s) \bmod q_{i-1}) \bmod t = m_1 \quad (2.50)$$

This technique is called modulus switching and the proof can be found in Section 4.3 of [29]. Homomorphic multiplication is then defined as:

$$H.Mul(ct_1, ct_2, rlk) = \frac{q_{i-1}}{q_i} \begin{pmatrix} (c_{1,2} c_{2,2} rlk_1 + c_{1,1} c_{2,1}) \bmod q_i, \\ (c_{1,2} c_{2,2} rlk_2 + c_{2,2} c_{1,1} + c_{1,2} c_{2,1}) \bmod q_i \end{pmatrix} \quad (2.51)$$

2.3.2.3 Remarks

In the integer-based scheme described in Section 2.2, a ciphertext has the form $c = pq + 2r + m$, where the term $2r + m$ is called noise. The decryption equation holds when the noise term stays below $p/2$. Homomorphic addition and homomorphic multiplication increase the noise in the ciphertext. Assume that the initial noise magnitude in a fresh ciphertext is B and $p \approx B^k$. Adding two ciphertexts will increase the noise to at most $2B$ while multiplying two ciphertexts will result in a ciphertext with noise as large as B^2 . By performing ciphertext multiplication continuously, the noise level grows very quickly $B^2 \rightarrow B^4 \rightarrow B^8 \rightarrow B^{16} \dots$ and reaches B^k after only $\log_2 k$ multiplications. This growth of noise is a common feature in Gentry's schemes and other works following his approach.

The homomorphic encryption scheme based on ring learning-with-errors problem, in contrast, have the noise grows linearly with ciphertext multiplication. This feature is made possible through modulus switching technique [29] presented in Section 2.2.3.2. In this technique, a set of gradually decreasing moduli $\{p_i \approx p/B^{k-i}\}$ are chosen for $i < k$. After multiplying two mod p_i ciphertexts, it is switched to the smaller modulus $p_{i-1} \approx p_i/B$ through appropriate scaling by $1/B$. Hence the noise level of the new ciphertext is reduced from B^2 to B . After multiplying two mod p_{i-1} ciphertexts, the noise becomes B^2 again. It is then switched back to the modulus p_{i-2} to bring the noise back to B . With this approach, k consecutive homomorphic multiplications can be performed instead of just $\log_2 k$ in Gentry's scheme. However, changing the modulus after every ciphertext is cumbersome. In a subsequent work [42], which is presented in Section 2.2.3.1, this modulus switching is eliminated to achieve a scale invariant scheme by scaling the ciphertext obtained by the multiplication by t/q (see Equation (2.39)).

2.4 Algorithm and Architecture for Hardware Accelerators

2.4.1 Accelerating Gentry's Ideal Lattice Based Scheme

2.4.1.1 Software Performance

The first attempt to implement a homomorphic encryption scheme that supports both ciphertext addition and multiplication was made by Smart and Vercauteren [48] in 2010. They focused on the ideal-lattice based scheme proposed by C. Gentry [23]. Although several optimizations were introduced, due to the high complexity of this ideal-lattice based scheme, Smart and Vercauteren cannot support large enough parameters to perform homomorphic decryption. Thus, they were only successful at implementing the underlying somewhat homomorphic encryption scheme. In 2011, C. Gentry and S. Halevi [24] successfully implemented the full scheme by further optimizing Smart and Vercauteren techniques and introducing several simplification. The simplified scheme is given as follows.

Key generation: The key generation procedure sets the public key as $pk = \langle d, r \rangle$ and the secret key as $sk = \omega$, where d , r , and ω are integers of the same size.

Encryption: To encrypt a message $m \in \{0, 1\}$ with the public key $pk = \langle d, r \rangle$, a random length n noise vector $\vec{u} = \langle u_0, u_1, u_2, \dots, u_{n-1} \rangle$ is chosen such that each u_i is 0 with some probability q and ± 1 with probability $(1-q)/2$ each. The ciphertext is represented by the integer $c = m + 2 \sum_{i=0}^{n-1} u_i r^i \bmod d$.

Decryption: The message m is recovered with the secret key ω by $m = (c \times \omega \bmod d) \bmod 2$.

Homomorphic addition and multiplication are equivalent to ciphertext addition and multiplication modulo d , respectively. Different parameter sets for

encryption and decryption procedures of the simplified scheme are listed in Table 2.1. The performance of the simplified ideal-lattice scheme is given in Table 2.2.

Table 2.1: Different parameter sets in the simplified ideal-lattice based scheme.

Settings	Dimension n	Size of determinant d
Toy	512	$\log_2 d \approx 195764$
Small	2048	$\log_2 d \approx 785006$
Medium	8192	$\log_2 d \approx 3148249$
Large	32768	$\log_2 d \approx 12625500$

Table 2.2: Performance of the simplified ideal-lattice scheme. Computation time is based the implementation on an IBM x3500 server with 64-bit quad-core Intel Xeon E5450 at 3GHz, 12MB L2 cache and 24GB RAM.

Settings	Key Generation	Encrypt	Decrypt	Homomorphically Decrypt
Toy	2.5 sec	0.19 sec	n/a	6 sec
Small	41 sec	1.8 sec	0.02 sec	32 sec
Medium	8.4 min	19 sec	0.13 sec	2.8 min
Large	2.2 hours	3 min	0.66 sec	31 min

As can be seen in Table 2.2, the performance of the the simplified ideal-lattice scheme is not acceptable although they are implemented on powerful processors. For medium settings, encryption time for a single-bit message takes 19 seconds. The time required to homomorphically decrypt a ciphertext is even worse: 2.8 minutes. The reason behind this poor performance lies on the extremely large parameters that are unseen in any existing cryptographic applications, as shown in Table 2.1.

From the description of the simplified ideal-lattice scheme and its parameters in Table 2.1, it is evident that integer multiplication and modular reduction are the major bottlenecks. Thus, to achieve a high-performance implementation, it is crucial to have hardware accelerators to overcome these bottlenecks. For modular reduction, two methods are commonly used: Barrett reduction [49] or

Montgomery reduction [50]. In Barret reduction method (see Algorithm 2.1), costly division is avoided by precomputing the reciprocal of the input modulus approximately. Then the quotient can be estimated by using this precomputed value. Since the quotient is obtained by an estimation, the product of the estimated quotient and the input modulus need to be subtracted from the input Z . To recover the exact remainder, a few more subtractions of the modulus from the remainder are necessary.

Unlike Barrett reduction method, in Montgomery reduction method (see Algorithm 2.2), costly division by the modulus M is replaced by a simple power of 2 division via a clever representation of the residue class modulo M . The downside of Montgomery reduction method is that the output, i.e., the computed remainder, stays in the Montgomery domain. Thus, to obtain the correct remainder, the output of Montgomery algorithm need to be post-processed by performing a modulo M operation $rR = ZR^{-1}R \bmod M = Z \bmod M$. Consequently, Barrett algorithm is greatly preferable to Montgomery algorithm in implementing hardware accelerators for the modular reduction of not only Gentry's ideal-lattice based scheme but also other schemes [25], [37], [38], [51], [52].

Algorithm 2.1: Barrett modular reduction

Input: Z, M where $0 \leq Z < M^2$, $2^{k-1} \leq M < 2^k$

Precomputation: $\mu = \lfloor 2^{2k}/M \rfloor$

Output: $r = Z \bmod M$

Procedure: $q = \lfloor \lfloor Z/2^k \rfloor (\mu/2^k) \rfloor$

$r = Z - qM$

while $(r \geq M)$ $r = r - M$;

return r

Algorithm 2.2: Montgomery modular reduction

Input: Z, M where $0 \leq Z < RM$, $2^{k-1} \leq M < 2^k$, $R = 2^k$

Precomputation: $M' = -M^{-1} \bmod R$

Output: $M' = -M^{-1} \bmod R$

Procedure: $q = (Z \bmod R)M' \bmod R$;

$r = (Z + qM) / R$;

if $r \geq M$ then return $r - M$ else return r

Since the cost of Barrett modular reduction is heavily determined by two integer multiplications (one is to estimate the quotient and one is to compute the product of the estimated quotient and the input modulus), a high-performance integer multiplier will be helpful in solving the bottlenecks of the simplified ideal-lattice based scheme. Given two integers, A and B , where each integer consists of n words, a naive way to compute the product of $A \times B$ is via a textbook multiplication. In this method, every word of the integer A is multiplied by every word of integer B to generate n^2 partial products, which in turn are accumulated by their corresponding weights to form the final product. Ignoring integer addition, the complexity of the textbook method is quadratic. When n is small, the complexity of the textbook multiplication is acceptable. However, this is not the case for the simplified ideal-lattice based scheme, where the size of input operands can be in the order of millions of bits. Thus, for more practical implementation of the simplified ideal-lattice based scheme, it is necessary to improve the textbook algorithm or explore other algorithms and platforms to further accelerate large size integer multiplication.

2.4.1.2 NTT Based Integer Multiplication

The first attempt to reduce the complexity of multiplication was due to Karatsuba [53]. His method is a recursive divide-and-conquer algorithm where an n -bit multiplication is performed by three $n/2$ -bit multiplications and a number of additions/subtractions, thus reducing the complexity from $O(n^2)$ to $O(n^{\log_2 3})$. Karatsuba approach was later generalized by Toom and Cook by breaking each n -bit number into three or more pieces. Depending on the number of pieces k ,

the complexity of Toom-Cook method is $O(n^{\log(2k-1)/\log k})$ [54]. The complexity of integer multiplication can be further minimized to quasilinear using Number Theoretic Transform (NTT) algorithm.

NTT algorithm is similar to Fast Fourier Transform (FFT) algorithm that is commonly used in the implementation of convolution for digital signal processing. If we treat the two input signal sequences as two polynomials, then the convolution result is exactly the product of these two polynomials. Thus, to apply NTT algorithm to compute the product C of two integers A and B , A and B must be expressed as two polynomials. A and B can be expressed as two multi-digit numbers, where each digit is k -bit long. Then the integer values of A and B are the polynomials $A(x) = \sum_i a_i x^i$ and $B(x) = \sum_i b_i x^i$ evaluated at $x = 2^k$, respectively and the coefficient of each polynomial is k -bit long. The NTT-based integer multiplication algorithm is shown in Figure 2.4.

Figure 2.4: N -point NTT-based integer multiplication.

The NTT-based integer multiplication algorithm depicted in Figure 2.4 is performed in the same way as FFT-based convolution in digital signal processing, except that there is an additional block called “carry resolver”. The purpose of this block is to recover the integer $C = A \times B$ from the coefficients of the polynomial $C(x) = \sum_i c_i x^i$. Clearly C can be recovered by evaluating $C(x)$ at $x = 2^k$. If all of $C(x)$ coefficients is less than 2^k , then recovering C is simply the concatenation of c_i 's. Unfortunately, this is not always the case and to correctly recover C , each c_i should be examined from $i = 0$ to $N - 1$ as follows: if $c_i \geq 2^k$, $c_i = c_i \bmod 2^k$ and $c_{i+1} = c_{i+1} + (c_i \text{ div } 2^k)$. After that, C is formed by simply concatenating all c_i 's.

An N -point NTT requires the existence of the N -th root of unity w_N , i.e., $w_N^N = 1$. In the well-known FFT in digital signal processing functions, the N -th root of unity always exists, which is the complex number $e^{-j2\pi/N}$. However, floating point arithmetic is not suitable for integer multiplication due to the complexity of round-off error analysis. The solution to this problem is to perform the NTT over a finite ring where the computations are carried out with exact integer arithmetic. The trade-off is the existence of the required root of unity is strongly dependent on the selected finite integer ring structure. Finite integer ring, denoted as Z_p , where p is a prime number, is the set of p integers $\{0, 1, 2, 3, \dots, p-2, p-1\}$. In the finite integer ring Z_p , addition and multiplication are performed modulo p . To enable an efficient NTT-based integer multiplication over the finite ring Z_p , the prime p should meet the following constraints:

- To support an N -point NTT, the finite ring Z_p must have N -th root of unity, i.e., there exists an integer w_N such that $w_N^N = 1 \bmod p$. This condition holds when $p \equiv 1 \bmod N$.
- The maximum value in the convolution must stay inside the ring, i.e., $\frac{N}{2}(2^k - 1)^2 < p$, to avoid reduction.

- $\frac{1}{N} \bmod p$ should exist in the finite ring. This condition is easy to satisfy when N is chosen as a power-of-2 integer and p is a prime number.
- Computations in finite ring are performed modulo p . Therefore, p is desirable to have special number theoretic properties that allow fast modulo reduction after a multiplication.

A good candidate of p is the Solinas prime number $2^{64} - 2^{32} + 1$ discovered by Emmart and Weems [55], which supports all power-of-two NTT up to 2^{32} . Using this number, they are able to implement integer multiplication with size up to 16384K bit by 16384K bit, which is highly suitable for implementing the scheme presented in Section 2.4.1.1. In addition, multiplication modulo this special number can be easily simplified. Let the multiplication result of two ring elements be an arbitrary 128-bit number z . z can be represented as $z = 2^{96}a + 2^{64}b + 2^{32}c + d$, where a, b, c and d are 32-bit numbers. By using two special number theoretic properties of p , i.e., $2^{96} \equiv -1 \bmod p$ and $2^{64} \equiv 2^{32} - 1 \bmod p$, $z \bmod p$ can be computed rapidly by:

$$\begin{aligned}
z &= 2^{96}a + 2^{64}b + 2^{32}c + d \\
&= (-1)a + (2^{32} - 1)b + 2^{32}c + d \\
&= 2^{32}(b + c) - a - b + d \pmod{p}
\end{aligned} \tag{2.52}$$

The process of performing NTT is similar to FFT in the complex domain, all it need is to replace the root of unity in the complex domain $e^{-i2\pi/N}$ by the integer root of unity in the finite ring w_N . Thus, the forward NTT is computed as

$$X_i = \sum_{j=0}^{N-1} x_j (w_N)^{ij} \pmod{p} \tag{2.53}$$

and the inverse NTT, or INTT, is computed as:

$$x_i = \frac{1}{N} \sum_{j=0}^{N-1} X_j (w_N)^{-ij} \pmod{p} \tag{2.54}$$

2.4.1.3 GPU-based Accelerator

The algorithm proposed by Emmart and Weems [55], which is presented in Section 2.4.1.2, is directly ported into the design of Wang et al. [25] to explore the feasibility of the simplified ideal-lattice based homomorphic encryption scheme (see Section 2.4.1.1) on graphical processing unit (GPU). Wang et al. also introduced several algorithmic optimizations. They proposed to precompute and store the public key in NTT form inside the GPU memory to reduce the number of necessary NTTs and INTTs. The number of necessary NTTs and INTTs is further reduced by taking advantage of the linearity of NTT domain for evaluating addition directly. As can be seen in Table 2.3, while precomputation brings significant speedup in small and medium settings, it is not the case for large setting because of the memory bottleneck, i.e., the precomputed values cannot be loaded into GPU memory simultaneously (see Table 2.4).

Table 2.3: Performance of the simplified ideal-lattice based homomorphic encryption scheme’s primitives on CPU vs. GPU.

SMALL SETTING				
OPERATION	CPU	Tesla C2050	GTX 690	Speedup
Encrypt	1.08 s	6.3 ms	6.2 ms	174
Decrypt	14 ms	2.5 ms	1.84 ms	7.6
Homomorphically Decrypt	17.8 s	1.8 s	1.32 s	13.5
MEDIUM SETTING				
OPERATION	CPU	Tesla C2050	GTX 690	Speedup
Encrypt	10.6 s	37 ms	24 ms	442
Decrypt	70 ms	13 ms	7.2 ms	9.7
Homomorphically Decrypt	96.3 s	12.4 ms	8.4 ms	11.5
LARGE SETTING				
OPERATION	CPU	Tesla C2050	GTX 690	Speedup
Encrypt	82 s	13.5 s	4.1 s	20
Decrypt	295 ms	90 ms	29 ms	10.1
Homomorphically Decrypt	800 s	n/a	235 s	3.4

Table 2.4: Memory usage on Tesla C2050 GPU.

SMALL SETTING	
OPERATION	Memory Used
Encrypt	192 MBytes
Homomorphically Decrypt	668 MBytes
MEDIUM SETTING	
Encrypt	1.5 GBytes
Homomorphically Decrypt	2.8 GBytes
LARGE SETTING	
Encrypt	18 GBytes
Homomorphically Decrypt	20 Gbytes

2.4.1.4 Application Specific Integrated Circuit (ASIC)-based Accelerator

In [37], Wang et al. explored the performance of integer multiplication on ASIC platform. They presented a 768K bits by 768K bits integer multiplier targeting the small settings of the simplified ideal-lattice based scheme in Section 2.4.1.1. The algorithm they adopted is the one presented in Section 2.4.1.2. Specifically, an architecture for 64K-point NTT over the finite ring Z_p , where $p = 2^{64} - 2^{32} + 1$, is proposed. This value of p enables an efficient architecture for modulo p multiplier according to Equation (2.52). The architecture of this multiplier is shown in Figure 2.5, where the inputs are two 64-bit integer and the output is the product of the inputs modulo reduced by p . The proposed hardware architecture for 64K-point NTT is shown in Figure 2.6.

Figure 2.5: Architecture of modular multiplication unit [37].

Figure 2.6: Architecture of radix-16 64K-point NTT processor [37].

The 64K-point NTT was implemented in 4 stages of 16-point NTT as shown in Figure. 2.6. Since a pipelined design requires memory buffers at each NTT stages, it is not suitable for high point NTT. Thus, Wang et al. adopt a memory-based in-place architecture, i.e., to store the immediate results of each stage directly to the same memory as the input data. In addition, they take advantage of the fact that 16-th root of unity in the chosen finite ring is a power-of-2 integer, i.e., $(2^{12})^{16} \equiv 1 \pmod{p}$. Since $2^{192} \equiv 1 \pmod{p}$, a 16-point NTT can be expressed as follows:

$$X(k) = \sum_{n=0}^{15} x(n) 2^{12nk \% 192} \pmod{p} \quad (2.55)$$

$$x(n) = \frac{1}{16} \sum_{k=0}^{15} X(k) 2^{(192-12nk) \% 192} \pmod{p} \quad (2.56)$$

Therefore, 16-point NTT over the chosen finite ring can be efficiently performed simply by shifting and adding (see Figure 2.7). This process can be further sped up by employing carry-save addition technique (see Figure 2.8). The proposed architecture is synthesized on 90nm CMOS technology. It is claimed that 0.206 ms per multiplication performance can be achieved, which is two times faster than their own implementation on a GPU presented in Section 2.4.1.3, with

significantly lower area and power consumptions.

Figure 2.7: Architecture of the hardware module for performing 16-point NTT [37].

Figure 2.8: Architecture of the hardware module for modulo summation of 16 operands [37].

In [38], Doroz et al. made a full custom ASIC implementation of the simplified ideal-lattice based scheme presented in Section 2.4.1.1. They adopted the optimization techniques from the GPU implementation of Wang et al. [25]. The performance of this full custom implementation is comparable to the GPU implementation [25] with a significantly lower silicon cost, despite using a less advanced CMOS technology. The custom ASIC implementation can speed up the computation by a few times, if it is synthesized with the same CMOS technology for the GPU in [25].

2.4.2 Accelerating CNT's Integer-based Scheme

2.4.2.1 Software Performance

In [56], Coron, Naccache, and Tibouchi (CNT) presented a technique to compress the public key in original integer scheme in Section 2.2.2. They also adapted modulus switching technique in ring learning-with-errors based scheme (see Section 2.3.2.2) to reduce the complexity of noise growth from quadratic to linear. The performance of this scheme is given in Table 2.5.

Table 2.5: Performance of Coron-Naccache-Tibouchi integer-based scheme. Computation time is based on a single core of Intel Core2 Duo E8400 processor at 3GHz.

Settings	Key Generation	Encrypt	Decrypt	Homomorphically Decrypt
Toy	0.06 sec	0.05 sec	0.00 sec	0.41 sec
Small	1.3 sec	1.0 sec	0.00 sec	4.5 sec
Medium	28 sec	21 sec	0.01 sec	51 sec
Large	10 min	7.25 min	0.05 sec	11.56 min

From Table 2.2 and Table 2.5, it is clear that with improved noise-management technique, CNT's integer-based scheme achieves a better performance than the original Gentry's integer-based scheme, though it is benchmarked on a less powerful computing platform. Despite that, the performance of CNT's scheme is still considered to be impractical to replace public-key schemes in current practice, such as RSA and elliptic curve cryptography (ECC).

2.4.2.2 Accelerating CNT's Scheme Using Comba Multiplication

Comba multiplication [57] is actually an effective scheduling method to take advantage of the abundant DSP blocks on modern FPGA devices (e.g., Xilinx Virtex-7 series) and reduce the number of expensive memory write accesses compared to textbook method. Suppose the product of two integers, A and B , is to be computed, where each integer is L bits long. If each DSP block can handle inputs with size up to L_{DSP} , then A and B can be viewed as $N_{DSP} = \lceil L / L_{DSP} \rceil$ -word numbers, $A_{N_{DSP}-1}A_{N_{DSP}-2}\dots A_1A_0$ and $B_{N_{DSP}-1}B_{N_{DSP}-2}\dots B_1B_0$, respectively. The classical multiplication method computes the product of A and B based on the accumulation of N_{DSP}^2 products. In order to parallelize this computation on N_{DSP} DSP blocks, the order of inner product computation is rescheduled as shown in Figure 2.9. All of N_{DSP} DSP blocks are configured to operate in MACC (multiply and accumulate) mode to hold the intermediate partial product PP_i .

Figure 2.9: Parallelizing multiplication using Comba method for efficient DSP-based multiplication. This example considers 4-word integers and 4 DSP blocks. The number of partial products PP_i and computation time are reduced compared to serial multiplication.

Comba multiplication has been employed to accelerate very large integer multiplication. The work in [51] focuses on the encryption step of Coron-Naccache-Tibouchi integer based scheme (see Algorithm 2.3 and Table 2.6). Based on Comba method, the authors build a w -bit by w -bit multiplier to take advantage of the rich sources of DSP blocks on the Virtex-7 FPGA XC7VX980T. In their implementation, w is set to 1024, 2048, 2048 and 4096 for toy, small, medium, and large parameter settings. Two architectures are also explored: low-area design using only one w -bit by w -bit Comba multiplier and high-speed design using many w -bit by w -bit multipliers in parallel (see Figure 2.10). The run time of the encryption step employing these two designs are presented in Table 2.7. For comparison purpose, the encryption time of software implementation by the authors of this integer-based scheme is included in the

last row.

Algorithm 2.3: Encryption procedure of Coron-Naccahe-Tibouchi scheme [56]

$$c = m + 2r + 2\sum_{i=1}^n X_i B_i \bmod X_0$$

c denotes the ciphertext, $m \in \{0,1\}$ denotes a single bit message, X_0 is part of the public-key, and each B_i for $i \in \{0,1,2,3,\dots,n\}$ is a randomly chosen integer.

Table 2.6: Different parameter sets in the encryption step of CNT integer-based scheme.

Settings	Bit-length of X_i	Bit-length of B_i	n
Toy	150×10^3	936	158
Small	830×10^3	1476	572
Medium	4.2×10^6	2016	2110
Large	19.35×10^6	2556	7695

Figure 2.10: Low-area (top) and parallel (bottom) architecture design of the encryption step [51].

Table 2.7: Average encryption time of Coron-Naccache-Tibouchi integer-based scheme using Comba multiplication.

Design	Platform	Toy	Small	Medium	Large
Low-area design	Virtex-7 XC7VX980T	0.016s	0.313s	5.773s	81.550s
High-speed design	Virtex-7 XC7VX980T	0.006s	0.114s	2.018s	32.744s
Original scheme [56]	Intel Core2 Duo E8400	0.05s	1.0s	21s	7.25 min

Both the low-area design and high-speed design on FPGA considerably outperform the software implementation on an Intel processor. While the low-area design is about 3-5 times faster, the speedup achieved by the high-speed design is significant, from 8 times in toy settings to more than 13 times in large settings. Since Comba method is basically a standard textbook multiplication, although the speedup seems impressive, its complexity remains quadratic. Very large size integer multiplication can be performed even faster by utilizing other methods with lower complexity.

2.4.2.3 Accelerating CNT's Scheme Using NTT-based Multiplication

In [52], Cao et al. proposed an optimized architecture especially for the encryption step of CNT integer-based scheme (see Algorithm 2.3). Based on the fact that the terms B_i in the encryption equation can be randomly chosen, Cao et al. suggested to set B_i to a low Hamming weight number and thus each term $X_i B_i$ can be computed simply by shifting and addition (see Figure 2.11). To carry out modulo X_0 using Barrett reduction, NTT over Z_p , where $p = 2^{64} - 2^{32} + 1$, was employed to carry out large integer multiplication. Unlike the previously discussed implementation, Cao et al. adopted a fully pipelined NTT design which requires costly buffers at each stage and thus, the number of NTT points they can support is limited. As a consequence, a large integer multiplication cannot be completed in a single NTT multiplication. To address this issue, they instantiated two NTT modules, two point-wise multipliers, two INTT modules and adopted an iterative multiplication accumulation logic as

shown in Figure 2.12. The architecture of large integer multipleir for Barrett reduction is shown in Figure 2.13.

Figure 2.11: Architecture of low Hamming weight multiplier [52].

Figure 2.12: Block accumulation logic used in NTT-based large integer multiplication for Barrett reduction [52].

Figure 2.13: Architecture of NTT-based large integer multiplier for Barrett modulo reduction [52].

In Table 2.8, the performance of the NTT-based design is compared against the performance of the high-speed Comba multiplier described in Section 2.4.2.2. By combining the low Hamming weight parameter and the NTT-based integer multiplication, the encryption step of CNT's integer-based scheme enjoys 10 times speedup on average.

Table 2.8: Average encryption time comparison of NTT-based design and Comba-based design.

Setting	Toy	Small	Medium	Large
NTT [52]	0.597 ms	10.867 ms	198.422ms	3316.696 ms
Comba [51]	6 ms	114 ms	2018 ms	32774 ms

2.4.3 Accelerating Schemes from Ring Learning-with-errors Problem

2.4.3.1 Software Performance

In [32], the performance of FV scheme in Section 2.3.2.1 is benchmarked. The authors remove the support of homomorphic decryption by computing the depth of the targeted application that will be homomorphically evaluated in the cloud in advance. By setting parameters large enough to support this depth, homomorphically evaluating the decryption circuit is not necessary. The performance of FV scheme is shown in Table 2.9.

Table 2.9: Timing of FV's primitives on Intel Core i7-2600 @ 3.4 GHz with no turbo-boost and hyper-threading.

Key generation	0.2 sec
Encryption	34 msec
Decryption	16 msec
Homomorphic addition	1.4 msec
Homomorphic multiplication	148 msec

As can be seen from Table 2.9 and Table 2.5, removing the support for homomorphic decryption can significantly improve the performance of homomorphic cryptosystems. However, it takes a considerable amount of time to homomorphically evaluate a real-world application, for example evaluating lightweight block cipher SIMON requires up to more than one hour, even with the support of turbo-boost and hyper-threading [32].

2.4.3.2 Hardware Accelerators

The homomorphic operations, namely homomorphic addition and homomorphic multiplication, of schemes from ring learning-with-errors problem are computed in the polynomial ring $Z_p[x]/(x^n + 1)$. Homomorphic addition is easy to perform since it is nothing but polynomial addition in the ring $Z_p[x]/(x^n + 1)$, all necessary computations are merely coefficient-by-coefficient modular p addition. On the other hand, homomorphic multiplication is more involved. Indeed, this has become the main target of all existing hardware accelerators for the schemes based on ring learning-with-errors problem [34, 36, 54].

For the polynomial multiplication in the polynomial ring $Z_p[x]/(x^n + 1)$, each element in this ring can be viewed as a polynomial of degree $n - 1$ and the degree of the product of two elements will be $2n - 2$. The smallest power of two numbers $N > 2n - 2$ is chosen to compute the product of two polynomials by using the NTT over Z_p . Unlike NTT-based integer multiplication algorithm described in Section 2.4.1.2, where an integer must be transformed into a polynomial, the two polynomials are directly applied as inputs and thus the carry resolver step is not necessary. The only condition that p should satisfy is $p \equiv 1 \pmod{N}$ to guarantee the existence of N -th root of unity. Similar to NTT-based integer multiplication, good candidates for p are Solinas prime numbers, which enable fast modulo reduction. Solinas prime number, named after Jerome Solinas, is a prime number of the form $2^a \pm 2^b \pm 1$, where $0 < b < a$. For example is $p = 2^{64} - 2^{32} + 1$ in Section 2.4.1.2 is a Solinas prime number. In [36], Popplemann et al. followed this approach to design a coprocessor for homomorphic addition and multiplication of YASHE [59] - a scheme similar to FV scheme described in Section 2.3.2.1.

While NTT algorithm is efficient for multiplication of high degree polynomials, its performance is limited by the size of the polynomial coefficients. Indeed, in [36], the authors can perform multiplication of polynomials with degree up to 16384 but the size of the polynomial coefficients is limited to 512. Thus, they cannot

support larger parameter sets. This problem can be solved by using residue number system (RNS) to break large size coefficient arithmetic into many small residue operations to be executed in parallel and independently. Before homomorphic encryption schemes, RNS has been also applied to accelerate common public key encryption schemes such as RSA and ECC [60].

An RNS is defined by a set of K pairwise prime moduli $\{m_1, m_2, \dots, m_K\}$, i.e., the greatest common divisor of m_i and m_j is one for $i \neq j$. The dynamic range of RNS is defined as $M = \prod_{i=1}^K m_i$. An integer $X \in [0, M)$ can be uniquely represented by a K -tuple $(x_1, x_2, \dots, x_K)$ in RNS, where each residue x_i is generated by $|X|_{m_i} = X \bmod m_i$. Let Z be the result of an arithmetic operation $\circ$ on X and Y , where $\circ \in (+, -, \times)$. $\circ$ can be performed on the residue digits of X and Y in parallel to retrieve the residue representation of Z by

$$(z_1, z_2, \dots, z_K) = \begin{pmatrix} x_1 \circ y_1 \bmod m_1, \\ x_2 \circ y_2 \bmod m_2, \\ \vdots \\ x_K \circ y_K \bmod m_K \end{pmatrix} \quad (2.57)$$

The NTT-RNS polynomial multiplication algorithm is illustrated in Figure 2.14.

Figure 2.14: NTT-RNS polynomial multiplication algorithm.

Consider two very high degree and very large coefficient polynomials, $A(x)$ and $B(x)$. $C(x) = A(x) \times B(x)$ can be calculated by the NTT-RNS algorithm as follows:

- Firstly, $A(x)$ and $B(x)$ are converted into their equivalent RNS form. For each of them, a set of K high degree polynomial but relatively small coefficient $a_i(x)$ and $b_i(x)$ are generated.
- Secondly, each $a_i(x)$ is multiplied by the corresponding $b_i(x)$ to obtain $c_i(x)$ by NTT algorithm
- Lastly, $C(x)$ is recovered from the set of $c_i(x)$ by residue-to-binary conversion

In [34], the algorithm in Figure 2.14 is carried out by having multiple parallel processors. In Figure 2.15, each processor contains two units: a polynomial arithmetic unit (PAU) for handling a single multiplication $c_i(x) = a_i(x) \times b_i(x)$ and a CRT unit (CRTU) for converting polynomial coefficients from RNS form to binary form. Division-and-rounding units (DRU) are dedicated to scaling and

rounding polynomial coefficients by t/q for homomorphic multiplication (see Equation (2.39)). In order to fit multiple processors into a large FPGA board, Roy et al. [34] adopted an in-place NTT architecture for the parallel processors. In [58], Doroz et al built a fully pipelined NTT processor. However, due to the large silicon cost of fully pipelined design, only one NTT processor can be synthesized to the target FPGA board. Thus, each $c_i(x) = a_i(x) \times b_i(x)$ must be computed in sequence.

Figure 2.15: Parallel hardware architecture for in-place NTT-RNS polynomial multiplication algorithm [34].

2.5 Conclusion

This chapter has briefly explained and provided a comprehensive review of various partially homomorphic encryption schemes, Gentry's fully homomorphic encryption scheme, and the important major developments following Gentry's scheme. Besides, algorithms and architectures for hardware accelerators have also been discussed. From the discussion, it can be concluded that:

- Schemes from ring learning-with-errors are more efficient than Gentry's lattice-based and integer-based schemes.
- Hardware accelerators can significantly improve timing-critical components

and supplement general-purpose computing platform to enhance performance of homomorphic cryptosystem. More works need to be done in this research direction.

- It is very challenging but highly paid off to eliminate noise growth, although it has already been reduced from quadratic to linear in schemes from ring learning-with-errors problem.

Chapter 3

Accelerating Conversion from Residue to Binary Representatio of Very High Cardinality Arbitrary Moduli Set for Homomorphic Encryption

3.1 Introduction

Due to the effect of noise growth, state-of-the-art homomorphic encryption schemes involves addition and multiplication of very high degree polynomials, from 27000 to 32768, and the bitwidth of the coefficients ranges from 517 bits to 1225 bits, depending on the security settings [32]. As a consequence, software implementation of state-of-the-art homomorphic encryption schemes consume considerable large amount of time for homomorphic evaluation of rather simple function. For SIMON-64/128, a popular block cipher which is significantly lighter than AES, takes 36.7 minutes, 1.3 minutes and about 1 hours 10 minutes on a 4-core Intel Core i7 processor for key generation, encryption and homomorphic function evaluation, respectively [32].

Accelerating performance-critical cryptographic computations by using hardware platform such as ASIC, FPGA, GPU has been a common practice for long. The reported results showed that hardware acceleration could bring homomorphic encryption technology closer to practice [28], [33], [34], [35], [36], [37], [38], [39]. The key success of implementing homomorphic encryption schemes on hardware platforms lies on techniques to parallelize computations. Due to its ability to break large calculation into parallel, independent, and smaller calculation, residue number system (RNS) is efficient in arithmetic operations of very large numbers. Very high cardinality RNS sets were used to accelerate homomorphic computation in recent implementation. In [33], Wang et al used an RNS set of thirty-two 32-bit wide moduli to accelerate matrix-vector multiplication on GPU. In [34], Roy et al. employed an RNS set of eighty four

30-bit wide moduli for homomorphic evaluation of SIMON-64/128 on FPGA. In [35], Doroz et al. used an RNS set of forty one 32-bit wide moduli for homomorphic evaluation of AES on FPGA. In [39], Dai et al. used RNS for homomorphic evaluation of AES on GPU. However, none of them reports how many RNS moduli have been used.

While parallelizing operations such as polynomial multiplication and large number arithmetic have been reported in RNS domain, current hardware implementation of homomorphic encryption schemes do not have technique to parallelize the residue-to-binary conversion of very high cardinality set. Thus, they are unable to leverage cloud computing power. As a result, a stand-alone residue-to-binary converter is much more costly than other homomorphic operations and contributes a significant part in the total delay. In [33], a residue-to-binary conversion occupies 16.75% of the total delay for a single matrix multiplication. In [34], it contributes 15.82% to the total execution time of homomorphic SIMON evaluation. In [35], 28.48% of total execution time for homomorphic AES evaluation is spent on residue-to-binary conversion. Even worse, in [39], one residue-to-binary conversion itself costs more than 50% of the total time in polynomial multiplications for homomorphic evaluation of AES.

To overcome the performance bottleneck of RNS-based accelerators for homomorphic encryption schemes, a new method to accelerate residue-to-binary conversion for very high cardinality arbitrary moduli set is proposed in this chapter. By using base extension to reduce the cardinality of the arbitrary moduli set to the popular three moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, the most efficient residue-to-binary conversion that leverages the number theoretic properties of these special moduli can be used to reduce its computational complexity.

The rest of this chapter is organized as follows: Section 3.2 provides the RNS background, presents a short review of residue-to-binary conversion methods used in state-of-art implementation of homomorphic encryption schemes. In Section 3.3, the proposed algorithm and its hardware architecture are described. In Section 3.4, the theoretical and experimental delay of the

proposed method are compared with previous works. Finally, Section 3.5 concludes the chapter.

3.2 Background

An RNS is defined by a set of N coprime integer numbers $\{m_1, m_2, m_3, \dots, m_N\}$, i.e., the greatest common divisor of any pair of integers belong to this set is equal to one. Let the dynamic range M be the product of all integers in the set $\{m_1, m_2, m_3, \dots, m_N\}$. An integer X in the range $[0, M)$ can be uniquely represented by an N tuple $(x_1, x_2, x_3, \dots, x_N)$, where each residue x_i can be determined by $X \bmod m_i$, i.e., the remainder of dividing X by the modulus m_i . $(x_1, x_2, x_3, \dots, x_N)$ is called the residue representation of X with respect to the base $\{m_1, m_2, m_3, \dots, m_N\}$.

Let Z be the result of a modulo M arithmetic operation $\diamond$ on X and Y , where $\diamond$ can be an integer addition, a subtraction, or a multiplication. The residue representation of X , Y and Z satisfy the following set of equations:

$$\begin{aligned} z_1 &= x_1 \diamond y_1 \bmod m_1 \\ z_2 &= x_2 \diamond y_2 \bmod m_2 \\ z_3 &= x_3 \diamond y_3 \bmod m_3 \\ &\vdots \\ z_{N-1} &= x_{N-1} \diamond y_{N-1} \bmod m_{N-1} \\ z_N &= x_N \diamond y_N \bmod m_N \end{aligned} \tag{3.1}$$

For example, let $N=3$ and the moduli set $\{m_1, m_2, m_3\} = \{63, 64, 65\}$. Let $X = 40633$ and $Y = 26967$. The residue representation of X and Y with respect to the base $\{63, 64, 65\}$ are computed as:

$$(x_1, x_2, x_3) = (X \bmod 63, X \bmod 64, X \bmod 65) = (61, 57, 8) \tag{3.2}$$

$$(y_1, y_2, y_3) = (Y \bmod 63, Y \bmod 64, Y \bmod 65) = (3, 23, 57) \tag{3.3}$$

Let $\diamond$ be the integer addition. The residue representation of $Z = X + Y = 67600$

with respect to the base $\{63,64,65\}$ are computed from the residue representation of X and Y as follows:

$$\begin{aligned}(z_1, z_2, z_3) &= (x_1 + y_1 \bmod 63, x_2 + y_2 \bmod 64, x_3 + y_3 \bmod 65) \\ &= (1, 16, 0)\end{aligned}\quad (3.4)$$

It can be easy to verify that (z_1, z_2, z_3) is the correct representation of Z with respect to $\{63,64,65\}$ since:

$$(67600 \bmod 63, 67600 \bmod 64, 67600 \bmod 65) = (1, 16, 0) \quad (3.5)$$

Given an arbitrary RNS base $\{m_1, m_2, m_3, \dots, m_N\}$ and a residue representation $(x_1, x_2, x_3, \dots, x_N)$, the binary representation of the integer X corresponding to this base can be computed by the Mixed Radix Conversion (MRC) or the Chinese Remainder Theorem (CRT) [61].

MRC expresses the integer X as:

$$X = x_1 + a_1 m_1 + a_2 m_1 m_2 + a_3 m_1 m_2 m_3 + \dots + a_{N-1} m_1 m_2 m_3 \dots m_{N-1} \quad (3.6)$$

where $a_1 \in [0, m_2)$, $a_2 \in [0, m_3)$, $a_3 \in [0, m_4)$, ..., $a_{N-1} \in [0, m_N)$.

The values of the mixed radix coefficients $a_1, a_2, a_3, \dots, a_{N-1}$ can be successively computed as follows. First, since $X - x_1$ is divisible by m_1 , $X - x_1$ can be determined by subtracting x_1 from all other residues. To obtain the division by m_1 , x_1 is subtracted from each residue x_i , $i \neq 1$ of X . The residue representation of the difference $X - x_1$ is then multiplied with the multiplicative inverse of m_1 with respect to the corresponding modulus x_i . We have:

$$\frac{X - x_1}{m_1} = a_1 + a_2 m_2 + a_3 m_2 m_3 + \dots + a_{N-1} m_2 m_3 \dots m_{N-1} \quad (3.7)$$

Thus, a_1 can be found in the residue corresponding to the modulus m_2 . This procedure sequentially determines $a_1, a_2, a_3, \dots$ and terminates when a_{N-1} is found. In Figure 3.1, an example with $N = 4$ is used to illustrate this method. In

this example, $|a|_m$ denotes $a \bmod m$, and $|m_i^{-1}|_{m_j}$ denotes the multiplicative inverse of m_i modulo m_j , i.e., the product of this number and m_i is equal to one modulo m_j .

m_1	m_2	m_3	m_4
x_1	x_2	x_3	x_4
	$-x_1$	$-x_1$	$-x_1$
	$ x_2 - x_1 _{m_2}$	$ x_3 - x_1 _{m_3}$	$ x_4 - x_1 _{m_4}$
	$\times m_1^{-1} _{m_2}$	$\times m_1^{-1} _{m_3}$	$\times m_1^{-1} _{m_4}$
	a_1	p	q
		$-a_1$	$-a_1$
		$ p - a_1 _{m_3}$	$ q - a_1 _{m_4}$
		$\times m_2^{-1} _{m_3}$	$\times m_2^{-1} _{m_4}$
		a_2	r
			$-a_2$
			$ r - a_2 _{m_4}$
			$\times m_3^{-1} _{m_4}$
			a_3

Figure 3.1: An example to illustrate Mixed Radix Conversion method

Due to the sequential generation of mixed radix coefficients, MRC-based methods are not suitable for high cardinality moduli set, especially in recent hardware implementation of homomorphic encryption schemes, where the cardinality of chosen RNS base is very high. On the other hand, CRT-based methods are widely used for the residue-to-binary conversion in the state-of-the-art hardware implementation of homomorphic encryption due to the logarithmic complexity with respect to the cardinality of chosen moduli set. To recover the binary representation of X from its RNS representation $(x_1, x_2, x_3, \dots, x_N)$ with respect to the base $\{m_1, m_2, m_3, \dots, m_N\}$, one of the following

equations can be used:

$$X = \left| \sum_{i=1}^K M_i \left| M_i^{-1} \right|_{m_i} x_i \right|_M \quad (3.8)$$

$$X = \left| \sum_{i=1}^K M_i \left| M_i^{-1} x_i \right|_{m_i} \right|_M \quad (3.9)$$

where $|a|_m$ denotes $a \bmod m$, $M_i = \frac{M}{m_i}$, and $|M_i^{-1}|_{m_i}$ is the multiplicative inverse of $M_i \bmod m_i$.

The difference between Equation (3.8) and Equation (3.9) lies on the number of subtractions of M from the weighted sum to carry out the modulo M operation. In Equation (3.9), since $|M_i^{-1} x_i|_{m_i} < m_i$, $M_i |M_i^{-1} x_i|_{m_i} < M$, the weighted sum is about K times greater than M . As a consequence, its modulo M reduction requires at most K subtractions of M . Equation (3.8), in contrast, needs much more subtractions of M from the weighted sum to complete the modulo M operation in the worst case. However, unlike Equation (3.9), it requires no modulo multiplication to evaluate the weighted sum.

Consider the base $\{m_1, m_2, m_3\} = \{15, 16, 17\}$ as an example. Let $(x_1, x_2, x_3) = (9, 8, 7)$. Thus, $M = 4080$, $M_1 = 272$, $M_2 = 255$, $M_3 = 240$, $|M_1^{-1}|_{15} = 8$, $|M_2^{-1}|_{16} = 15$, $|M_3^{-1}|_{17} = 9$. Applying Equation (3.8) gives:

$$\begin{aligned} X &= |272 \times 8 \times 9 + 255 \times 15 \times 8 + 240 \times 9 \times 7|_{4080} \\ &= |65304|_{4080} \\ &= 24 \end{aligned} \quad (3.10)$$

Applying Equation 3.9 gives:

$$\begin{aligned} X &= |272 \times |8 \times 9|_{15} + 255 \times |15 \times 8|_{16} + 240 \times |9 \times 7|_{17}|_{4080} \\ &= |272 \times 12 + 255 \times 8 + 240 \times 12|_{4080} \\ &= |8184|_{4080} \\ &= 24 \end{aligned} \quad (3.11)$$

Equation (3.8) is used in [33] and [34] whereas Equation (3.9) is used in [35] and [39] for residue-to-binary conversion. Because of the large number of arbitrary moduli used in the RNS-based hardware implementation of homomorphic encryption, large and computationally intensive modulo arithmetic operations are incurred in the CRT-based residue-to-binary conversion. Wang et al [33] addressed the modulo reduction problem by applying Barrett modulo reduction techniques. Barrett reduction basically replaces the division by using the precomputed reciprocal of the modulus, simple shift and multiplication operations. After that, at most three subtractions are needed to obtain the correct result. The pseudo code of Barrett modulo reduction is given in Algorithm 3.1.

Algorithm 3.1 Barrett modulo reduction algorithm

Input: Z, M where $0 \leq Z < M^2$, $2^{k-1} \leq M < 2^k$

Precomputation: $\mu = \lfloor 2^{2k}/M \rfloor$

Output: $r = Z \bmod M$

Procedure: $q = \lfloor \lfloor Z/2^k \rfloor (\mu/2^k) \rfloor$

$r = Z - qM$

while $(r \geq M)$ $r = r - M$;

return r

Another method to overcome the large modulo operations is to perform conditional subtraction by checking the summation of the terms in each iteration of Equation (3.9). This iterative method is implemented in [35] and [39]. In [34], large modulo reduction is performed by a windowed subtraction method, which resembles the classical shift-and-subtract binary division algorithm. The left most digits in the dividend and divisor are first aligned to a window section. If the windowed section of the dividend is greater than or equal to the divisor, the divisor is subtracted from the windowed section of the dividend. The divisor is then shifted one bit right to repeat the windowed subtraction process until the dividend is less than the divisor.

3.3 Proposed Algorithm and Hardware Architecture

Since the inefficiency of CRT-based residue-to-binary conversion originates

from the high cardinality base and large modulo reduction, a fast residue-to-binary conversion method will tackle both problems. High cardinality base can be eliminated simply by converting the residue representation to a low cardinality base. However, if the target low cardinality base is an arbitrary base, the large modulo reduction remains as a major problem. Fortunately, this problem can be solved by using a modular arithmetic friendly base as the low cardinality base. The proposed algorithm for residue-to-binary conversion of very high cardinality arbitrary moduli set consists of two stages. The first stage reduces the large number of residues to three and modifies the corresponding large number of arbitrary moduli to only three arithmetic friendly special moduli. In the second stage, the residue-to-binary conversion is accelerated by performing the CRT in the special moduli set.

Given an integer X in its RNS representation $(x_1, x_2, \dots, x_N)$ with respect to the base $\{m_1, m_2, \dots, m_N\}$, base extension is a procedure to find out its RNS representation $(x'_1, x'_2, \dots, x'_K)$ corresponding to a new base $\{m'_1, m'_2, \dots, m'_K\}$. To maintain the uniqueness, the new base $\{m'_1, m'_2, \dots, m'_K\}$ is chosen such that its dynamic range is greater than the dynamic range of the old base $\prod_{j=1}^K m'_j \geq \prod_{i=1}^N m_i$. For each $j \in [1, K]$, x'_j can be computed by employing Equation (3.9) as follows:

$$\begin{aligned}
 x'_j &= |X|_{m'_j} \\
 &= \left\| \sum_{i=1}^N M_i |M_i^{-1} x_i|_{m_i} \right\|_{M|_{m'_j}} \\
 &= \left| \sum_{i=1}^N M_i |M_i^{-1} x_i|_{m_i} - rM \right|_{m'_j} \\
 &= \left| \sum_{i=1}^N |M_i|_{m'_j} |M_i^{-1} x_i|_{m_i} - r|M|_{m'_j} \right|_{m'_j}
 \end{aligned} \tag{3.12}$$

where r is an integer such that $0 \leq r < N$.

r can be evaluated using the fast base extension method of Shenoy and Kumaresan [62] as follows:

$$r = \left\| M^{-1} \right\|_{m_r} \left(\sum_{i=1}^N \left\| M_i \right\|_{m_r} \left\| M_i^{-1} x_i \right\|_{m_i} \right) - x_r \Bigg\|_{m_r} \quad (3.13)$$

where m_r is a redundant modulus such that $m_r \geq N$ and $x_r = \left\| X \right\|_{m_r}$.

Since the constants, $\left\| M_i \right\|_{m'_j}$ and $\left\| M \right\|_{m'_j}$, in Equation (3.12) can be precomputed and their bit width are limited by the bit width of m'_j , the large arithmetic operations in Equation (3.8) and Equation (3.9) are avoided. Moreover, the computation of the residue x'_i of the new moduli set can be performed independently from the residue x'_j for all $i \neq j$, resulting in shorter delay than the large non-parallelizable modulo operations in Equation (3.8) and Equation (3.9). The hardware architecture of this base extension operation of Equation (3.12) is shown in Figure 3.2. Equation (3.13) has similar expression as Equation (3.12), except that all its modulo operations are much smaller. Therefore, the integer r can be computed from the redundant modulus in parallel with a much shorter delay before the final modulo subtraction in the left branch of Figure 3.2. The hardware architecture to determine r is shown in Figure 3.3.

Figure 3.2: Architecture of base extension operation in Stage 1.

Figure 3.3: Architecture to determine r for base extension operation.

As the new moduli set has the same or larger dynamic range than M , large modular arithmetic operations are inevitable in its residue-to-binary conversion. To simplify the complexity of residue-to-binary conversion in the second stage, the new moduli set $\{m'_1, m'_2, \dots, m'_K\}$ is selected to be the three moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. This three moduli set is well known for its efficient fully combinational adder-based implementation of residue-to-binary conversion due to the unique number theoretic properties of modulo 2^n and $2^n \pm 1$ operations. The efficient residue-to-binary converter II described in [63] is directly adopted in this stage. It uses four n -bit carry lookahead adders, $2n$ one-bit full adders, two n -bit 2-1 multiplexer, and some simple logic gates. The conversion procedure is briefly described below.

After the first stage, the resulting integer can be represented as a 3-tuple (X_1, X_2, X_3) , which is denoted as:

$$X_1 = |X|_{2^n - 1} = X_{1(n-1)} X_{1(n-2)} \dots X_{10} \quad (3.14)$$

$$X_2 = |X|_{2^n} = X_{2(n-1)} X_{2(n-2)} \dots X_{20} \quad (3.15)$$

$$X_3 = |X|_{2^n + 1} = X_{3n} X_{3(n-1)} X_{3(n-2)} \dots X_{30} \quad (3.16)$$

The binary integer can be recovered from (X_1, X_2, X_3) by concatenation using the following equation:

$$X = X_2 + 2^n (Y_1 + 2^n Y_2) \quad (3.17)$$

where X_2 , Y_1 , and Y_2 are n -bit numbers.

Y_1 and Y_2 can be computed using the architecture in Figure 3.4. In Figure 3.4, “CLA1”, “CLA2”, “CLA3”, and “CLA4” are n -bit carry look ahead adders. The architectures of the blocks “nFA1” and “nFA2” are depicted in Figure 3.5.

Figure 3.4: Residue-to-binary conversion architecture for $\{2^n-1, 2^n, 2^n+1\}$ [63].

Figure 3.5: Blocks "nFA1" and "nFA2" [63]

The following numerical example illustrates the base extension method to convert the residue representation $(1, 0, 4, 1, 3)$ of an integer $X = 10000$ in the moduli set $\{3, 5, 7, 11, 13\}$ to the residues (x'_1, x'_2, x'_3) in the three moduli set $\{2^5 - 1, 2^5, 2^5 + 1\}$. To compute r , the smallest integer larger than $N = 5$ and coprime to the source moduli is chosen to be the redundant modulus $m_r = 8$. Since $M = 15015$, $M_1 = 5005$, $M_2 = 3003$, $M_3 = 2145$, $M_4 = 1365$ and $M_5 = 1155$. Thus, $|M_i^{-1}x_i|_{m_i} = 1, 0, 6, 1$ and 5 for $i = 1, 2, 3, 4$ and 5 , respectively. From Equation (3.13), we have

$$r = \left| 15015^{-1} \left(\sum_{i=1}^5 |M_i|_8 |M_i^{-1}x_i|_{m_i} - 0 \right) \right|_8 = |7 \times (7 - 0)|_8 = 1 \quad (3.18)$$

According to Equation (3.12):

$$\begin{aligned} x'_1 &= \left| \sum_{i=1}^5 |M_i|_{31} |M_i^{-1}x_i|_{m_i} - 1 \times |15015|_{31} \right|_{31} = 18 \\ x'_2 &= \left| \sum_{i=1}^5 |M_i|_{32} |M_i^{-1}x_i|_{m_i} - 1 \times |15015|_{32} \right|_{32} = 16 \\ x'_3 &= \left| \sum_{i=1}^5 |M_i|_{33} |M_i^{-1}x_i|_{m_i} - 1 \times |15015|_{33} \right|_{33} = 1 \end{aligned} \quad (3.19)$$

3.4 Performance Analysis and Comparison

3.4.1 Delay of the Proposed Method

To compare the performance of the proposed residue-to-binary converter with existing converters that use different number of moduli N and average bit width per modulus w for RNS-based hardware implementation of homomorphic encryption schemes, the time complexity of the proposed converter is analytically evaluated independent of the moduli set first. Based on the prime number theorem, the number of primes not exceeding q is asymptotic to $q/\ln(q)$ and is more accurately approximated by $q/(\ln(q)-1)$. There are more than enough primes with bit widths differ by no more than one bit to be selected as the moduli for the implementation of homomorphic encryption schemes. Hence, the bit widths of all the moduli and residues can be assumed to be uniform and

equal to w bits. For an N -moduli set, the dynamic range M of the chosen RNS can be approximated by Nw bits. The first stage of the proposed method maps the N residues in this moduli set to three residues in the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, which has a dynamic range $M' \geq M$. Since $\{2^n - 1, 2^n, 2^n + 1\}$ is a balanced moduli set, each of its moduli and residues can be represented by approximately $\lceil Nw/3 \rceil$ bits.

From Figure 3.2, the critical path of the base extension operation in the first stage is contributed by the delays of one modular multiplier, one binary multiplier and $\lceil \log_2 N \rceil + 3$ binary adders. This is elucidated as follows. A $(w \times w)$ -bit modular multiplier is needed to compute $|M_i^{-1}x_i|_{m_i}$. The w -bit output is then multiplied by the constant $|M_i|_{m_j}$ of approximately $\lceil Nw/3 \rceil$ bits, which incurs a delay of a $w \times \lceil Nw/3 \rceil$ -bit binary multiplier. The $(\lceil Nw/3 \rceil + w)$ -bit outputs of N parallel multipliers are summed in an adder tree before the term $r|M|_{m_j}$ is subtracted. As $r|M|_{m_j}$ can be computed in a much shorter time independently, it does not constitute to the critical path delay. The total delay of the adder tree and the subtractor can be approximated by $(\lceil \log_2 N \rceil + 1)$ times the delay of a $(\lceil Nw/3 \rceil + w)$ -bit binary adder by ignoring the slight expansion in bit width. The result is modulo reduced by m'_j to produce three final residues x'_j . The modulo reduction by 2^n can be done at zero delay by simply keeping the n least significant bits. The modulo $2^n \pm 1$ reduction can be done at the cost of two binary additions [64]. Thus, the delay of the modulo reduction can be approximated by two times the delay of a $\lceil Nw/3 \rceil$ -bit binary adder. Omitting the negligible delay of the 2-1 multiplexer, inverters, and one-bit full adder of the residue-to-binary converter for $\{2^n - 1, 2^n, 2^n + 1\}$, the delay in the second stage is dominated by the delay of a $\lceil Nw/3 \rceil$ -bit binary adder, according to Figure 3.4 and Figure 3.5.

3.4.2 Delay of Wang's Method

Wang et al. [33] rely on Equation (3.8) and Barrett modulo reduction algorithm for their residue-to-binary conversion. To analyze the delay, their method is divided into three steps. In the first step, $M_i \left| M_i^{-1} \right|_{m_i} x_i$ is evaluated for each $i = 1, 2, 3, \dots, N$. Using the same bit-width analysis method in Section 3.4.1, by having N parallel $w \times Nw$ -bit multiplier, the delay of this step can be approximated by the delay of a $w \times Nw$ -bit multiplier, whose output is considered to be $(N+1)w$ bits wide. In the second step, the summation of $M_i \left| M_i^{-1} \right|_{m_i} x_i$ for $i = 1, 2, 3, \dots, N$ is evaluated by a binary adder tree. The delay of this step can be approximated by $\lceil \log_2 N \rceil$ times the delay of a $(N+1)w$ -bit adder. The result of this step can be considered to be $((N+1)w + \lceil \log_2 N \rceil)$ bits wide. In the third step, the result of the previous step is reduced modulo M using the Barrett modulo reduction algorithm. Ignoring the addition, the delay of this step can be approximated by two times the delay of a $(\lceil \log_2 N \rceil + w) \times Nw$ -bit multiplier.

3.4.3 Delay of Roy's Method

Roy et al. [34] rely on Equation (3.8) and the shift-and-subtract division for their residue-to-binary conversion. To analyze the delay, their method is divided into three steps. In the first step, $M_i \left| M_i^{-1} \right|_{m_i} x_i$ is evaluated for each $i = 1, 2, 3, \dots, N$. By having N parallel $w \times Nw$ -bit multiplier, the delay of this step can be approximated by the delay of a $w \times Nw$ -bit multiplier. In the second step, the summation of $M_i \left| M_i^{-1} \right|_{m_i} x_i$ for $i = 1, 2, 3, \dots, N$ is evaluated by a binary adder tree. The delay of this step can be approximated by $\lceil \log_2 N \rceil$ times the delay of a $(N+1)w$ -bit adder. In the third step, the result of the second step is reduced modulo M using the shift-and-subtract division method. In shift-and-subtract division, the necessary number of subtractions depends on the difference in the bit widths of the modulus and the number that need to be modulo reduced. Since the sum from the second step is $((N+1)w + \lceil \log_2 N \rceil)$ bits wide and M is Nw , the

bit width difference is determined to be $w + \lceil \log_2 N \rceil$. In this step, the number of necessary Nw -bit subtractions is estimated to be $(w + \lceil \log_2 N \rceil)/2$ on average.

3.4.4 Delay of Doroz's Method

Doroz et al. [35] rely on Equation (3.9) and the iterative method for their residue-to-binary conversion. To analyze the delay, their method is divided into four steps. In the first step, $|M_i^{-1}x_i|_{m_i}$ is evaluated for each $i = 1, 2, 3, \dots, N$. By having N parallel $w \times w$ -bit modular multiplier, the delay of this step can be approximated by the delay of a $w \times w$ -bit modular multiplier. In the second step, the output of each modular multiplier in the first step is multiplied with the corresponding M_i . By having N parallel $w \times (N-1)w$ -bit multiplier, the delay of this step can be approximated by the delay of a $w \times (N-1)w$ -bit multiplier. The output of each multiplier is considered to be Nw bits wide. In the third step, the summation of the outputs of all multipliers in the second step is evaluated by a binary adder tree. The delay of the third step can be approximated by $\lceil \log_2 N \rceil$ times the delay of a Nw -bit adder. In the fourth step, the sum from the third step is subtracted by M successively until it is less than M . In this step, the number of necessary Nw -bit subtractions is estimated to be $N/2$ on average.

3.4.5 Comparison

The analytical delays of the residue-to-binary converters from [33], [34] and [35] are compared with the proposed converter in Table 3.1. The delays are expressed in terms of the delay of the equivalent number of binary multiplications Δ_M , modular multiplications Δ_{MM} and binary additions Δ_A , with the size (bit width) of each type of operations expressed in terms of the number of moduli N and average bit width w of each modulus. The delay of a binary subtraction is considered to be the same as the delay of a binary addition. The symbols in Table 3.1 can be interpreted as follows. $\Delta_M \left\langle w \times \left\lceil \frac{Nw}{3} \right\rceil \right\rangle$ is the delay

of a w bits by $\left\lceil \frac{Nw}{3} \right\rceil$ bits binary multiplier. $\Delta_A \left\langle \left\lceil \frac{Nw}{3} \right\rceil + w \right\rangle$ is the delay of a $\left(\left\lceil \frac{Nw}{3} \right\rceil + w \right)$ bits binary adder. $\Delta_A \left\langle \left\lceil \frac{Nw}{3} \right\rceil \right\rangle$ is the delay of a $\left\lceil \frac{Nw}{3} \right\rceil$ bits binary adder. $\Delta_{MM} \langle w \times w \rangle$ is the delay of a w bits by w bits modulo multiplier. The remained symbols can be interpreted in a similar manner.

Table 3.1: Performance comparison of residue-to-binary converters.

Design	Delay
This work	$\Delta_M \langle w \times \left\lceil \frac{Nw}{3} \right\rceil \rangle + (\lceil \log N \rceil + 1) \Delta_A \left\langle \left\lceil \frac{Nw}{3} \right\rceil + w \right\rangle + 3 \Delta_A \left\langle \left\lceil \frac{Nw}{3} \right\rceil \right\rangle + \Delta_{MM} \langle w \times w \rangle$
[33]	$\Delta_M \langle w \times Nw \rangle + (\lceil \log N \rceil) \Delta_A \langle Nw + w \rangle + 2 \Delta_M \langle (\lceil \log N \rceil + w) \times Nw \rangle$
[34]	$\Delta_M \langle w \times Nw \rangle + (\lceil \log N \rceil) \Delta_A \langle Nw + w \rangle + \left(\frac{\lceil \log N \rceil + w}{2} \right) \Delta_A \langle Nw \rangle$
[35]	$\Delta_M \langle w \times (N - 1)w \rangle + \left(\lceil \log N \rceil + \frac{N}{2} \right) \Delta_A \langle Nw \rangle + \Delta_{MM} \langle w \times w \rangle$

All the residue-to-binary converter designs are also described in Verilog, synthesized and mapped to the STM CMOS 40nm standard cell library by Synopsys Design Compiler. For a fair comparison, the proposed residue-to-binary converter is designed for three different combinations of N and w reported separately in [33], [34] and [35]. The same type of binary and modular multipliers and binary adder are adopted for all the designs in comparison. The binary adders and multipliers are implemented using the industry proven IPs available in Synopsys DesignWare. The improved Barrett modular multiplication algorithm [65] is used for the implementation of the modular multiplier. The improved algorithm needs at most one subtraction as opposed to multiple subtractions in the Barrett modulo reduction algorithm. The synthesized delays and areas of different residue-to-binary converters are shown in Table 3.2.

Table 3.2: Synthesis results of different residue-to-binary converters

	$N = 32, w = 32$		$N = 84, w = 30$		$N = 41, w = 32$	
	This work	[33]	This work	[34]	This work	[35]
Area (mm^2)	3.88	3.75	23.08	22.36	6.24	6.14
Delay (ns)	487.6	1095.77	1345.38	6855.65	671.24	4565.09

The results show that the proposed residue-to-binary converter is faster than all the other designs in comparison for the same moduli set. From the moduli set independent analysis in Table 3.1, the reason for this speed gain can be attributed to the generally smaller operands used in the proposed method for the additions and the time-critical multiplications. Specifically, with $N = 84$ and $w = 30$, while the delay of the work in [34] is the sum of the delay of a 30-bit by 2520-bit binary multiplier plus 7 times the delay of a 2550-bit binary adder plus 18 times the delay of a 2520-bit adder, the delay of the proposed converter is the sum of the delay of a 30-bit by 840-bit binary multiplier plus 8 times the delay of a 870-bit binary adder plus 3 times the delay of a 840-bit binary adder plus the delay of a 30-bit by 30-bit modulo multiplier. It is clear that the delay of the proposed converter should be much less than the delay of the work in [34]. Similar arguments can be made for the case of $N = 32$ and $w = 32$ and the case of $N = 41$ and $w = 32$. From Table 3.2, the proposed residue-to-binary converter design is 2.25, 5.1, and 6.8 times faster than the designs of [33], [34] and [35], respectively. While the design of [33] uses Barrett algorithm for large modulo reduction (see Algorithm 3.1), the designs of [34] and [35] rely on iteration or binary division, which adds more adder delay into their critical paths. The improved residue-to-binary performance can be extrapolated to an acceleration of hardware implementation of homomorphic encryption schemes by 10.2% to 32.08% if the existing residue-to-binary converters are replaced by the proposed design. The significant speed improvement is achieved at an insignificant area overhead of no more than 3.5% over the designs of [33], [34], and [35]. The excess area is mainly incurred by the redundant modulus for fast base

extension, and represents a negligibly small fraction of the total footprint of the hardware implementation of homomorphic encryption schemes. The trade-off is acceptable since speed is presently the stumbling block to the widespread application of the state-of-the-art homomorphic encryption schemes.

3.5 Summary

This chapter presents a fast residue-to-binary conversion method for high cardinality arbitrary moduli set by transforming a large number of residues of arbitrary moduli into only three residues before the conversion. When recent converters used in RNS-based accelerators of homomorphic encryption suffer from very large operand arithmetic and inefficient large modulo reduction, the proposed converter avoids them by exploiting the smaller and parallel operations of base extension for the mapping to $\{2^n - 1, 2^n, 2^n + 1\}$ and the highly efficient adder-based residue-to-binary converter of the latter to achieve 2.25 ~ 6.8 times speedup. This speedup of residue-to-binary conversion has translated into approximately 10.2% to 32.08% of overall speed improvement for the hardware implementations of the respective homomorphic encryption schemes.

Chapter 4

Exploring Number Theoretic Transform on ASIC for Ring Learning-with-Error Based Cryptography

4.1 Introduction

Besides the application in building the most efficient homomorphic encryption schemes [66], ring learning-with-errors problem also find its usage in designing quantum-resistant cryptography. Nowadays, a large majority of digital devices and applications are being secured under the hardness of classical hard problems, such as integer factorization (for RSA) and discrete logarithm (for ECC). However, recent development in quantum technology has posed serious threats to systems whose security are guaranteed by RSA and ECC, since they can be solved in polynomial time by using Shor algorithm [67]. Ring learning-with-error based cryptography, proven to be immune from quantum attacks [43], is being considered a promising candidate to replace the vulnerable cryptosystems. Since its introduction, many cryptosystems have been proposed and experimented for commercial products, such as key exchange protocol [68] and digital signature [69].

Several implementations of ring learning-with-error based cryptography have been proposed, targeting various software and hardware platforms [70], [71], [72], [73]. However, hardware architectures targeting ring learning-with-error based cryptography have been reported only on reconfigurable devices, leaving ASIC-based designs unexplored at the time of this study. In the case of homomorphic encryption schemes, their parameters are often very large [32], making it impossible to validate the design on ASIC platform. In the case of quantum-resistant cryptography, the parameter selection is still an open problem which deters worthy investigation of custom ASIC-based designs. Motivated by the recent governmental warnings of quantum technology and

calling for the standardization of post-quantum cryptography [74], as well as the rapidly increasing concern about the privacy preservation of user's sensitive information [75], it is crucial to start exploring and evaluating the designs of ring learning-with-error based cryptography on ASIC platform.

In this chapter, the ASIC design of ring learning-with-error based cryptography is explored with a custom implementation of the Number Theoretic Transform (NTT) for polynomial multiplication - the most critical and computationally intensive operation in ring learning-with-error based cryptography. The custom design makes use of the algorithmic properties of either Barrett or Montgomery multiplication algorithm for two simplified modulo multipliers. Besides, it employs two parallel half-sized dual-port memories to further increase the throughput.

The remainder of this chapter is organized as follows. Section 4.2 gives a brief review of the state-of-the-art algorithm and hardware architectures of Number Theoretic Transform. The proposed custom architecture is presented in Section 4.3. The proposed design is then compared with other works in Section 4.4. The chapter is concluded in Section 4.5.

4.2 Number Theoretic Transform

Ring learning-with-errors based cryptosystems are built around the polynomial ring $R = \mathbb{Z}_p[x]/f(x)$, where p is a prime number and $f(x)$ is an irreducible polynomial. For practical reasons, $f(x)$ is commonly chosen as $(x^N + 1)$, where N is a power of 2. When $p \equiv 1 \pmod{2N}$, there exists a $2N$ -th root of unity $\psi_{2N} \in \mathbb{Z}_p$ such that $\psi_{2N}^{2N} \equiv 1 \pmod{p}$. Such root of unity reduces the complexity of polynomial multiplication, which is the most crucial and computationally intensive operation in R , by using NTT algorithm (see Chapter 2, Section 2.4.3.2).

Since the product of two elements belongs to $R = \mathbb{Z}_p[x]/(x^N + 1)$ is a polynomial with degree up to $(2N - 2)$, to carry out the multiplication by NTT algorithm, the

input polynomials need to be padded with N zeros, i.e., the inputs are treated as polynomials with degree $(2N-1)$, and $2N$ -point NTT should be applied. Let

$a(x) = \sum_{i=0}^{N-1} a_i x^i$ be an element belong to $R = \mathbb{Z}_p[x]/(x^N + 1)$. Let the sequence $A = NTT(a)_{\psi_{2N}}^{2N} = \{A_0, A_1, A_2, \dots, A_{2N-1}\}$ be the $2N$ -point NTT representation of $a(x)$. A is computed as:

$$A_j = \sum_{i=0}^{2N-1} a_i (\psi_{2N})^{ij} \pmod{p} \quad (4.1)$$

where $a_j = 0$ for $j = N, N+1, N+2, \dots, 2N-1$.

To recover $a(x) = INTT_{\psi_{2N}}^{2N}(A)$, the coefficients are computed by:

$$a_i = \frac{1}{2N} \sum_{j=0}^{2N-1} A_j (\psi_{2N})^{-ij} \pmod{p} \quad (4.2)$$

In [72], it is proposed to take advantage of the special arithmetic property of $(x^N + 1)$ to reduce the necessary number of NTT points from $2N$ to N , thus

removing the need for zero padding. Let $b(x) = \sum_{i=0}^{N-1} b_i x^i$ and $c(x) = \sum_{i=0}^{N-1} c_i x^i$ be two polynomials such that $c(x)$ is the product of $a(x)$ and $b(x)$ in the ring R .

Furthermore, let $w \equiv \psi_{2N}^2 \pmod{p}$, $a'(x) = \sum_{i=0}^{N-1} a_i (\psi_{2N})^i x^i$,

$b'(x) = \sum_{i=0}^{N-1} b_i (\psi_{2N})^i x^i$ and $c'(x) = \sum_{i=0}^{N-1} c_i (\psi_{2N})^i x^i$. The negative wrapped

convolution [72] states that:

$$NTT(c')_w^N = NTT(a')_w^N \odot NTT(b')_w^N \quad (4.3)$$

where $\odot$ denotes the point-wise multiplication.

Although the negative wrapped convolution can reduce the number of NTT points by half, it requires to scale the input polynomials by powers of ψ_{2N} . In [76], it is noted that $a'(x) = a(\psi_{2N}x)$. Thus, scaling the input polynomials by powers of ψ_{2N} can be merged into the computation process of NTT by scaling

the twiddle factor by $\psi_{2N}^{N/2}$ in the first stage, $\psi_{2N}^{N/4}$ in the second stage, $\psi_{2N}^{N/8}$ in the third stage, and so on, using the Cooley-Tukey radix-2 decimation-in-time approach [77]. In [70], the techniques from [72] and [76] are incorporated to propose the most optimized algorithm. Its pseudo-code is presented in Algorithm 4.1.

Algorithm 4.1 Optimized NTT Algorithm [70]

Precondition: Store N powers of ψ_{2N} in bit-reversed order vector

S

Input: Coefficients $a_0, a_1, a_2, \dots, a_{N-1}$ of polynomial $a(x)$

Procedure:

```

     $m = 1;$ 
     $k = N/2;$ 
    while  $m < N$  do
        for  $i = 0$  to  $m - 1$  do
             $j\_first = 2 \times i \times k;$ 
             $j\_last = j\_first + k - 1;$ 
             $\Psi_i = S_{m+i};$ 
            for  $j = j\_first$  to  $j\_last$  do
                 $l = j + k;$ 
                 $a_j = a_j + a_l \times \Psi_i \bmod p;$ 
                 $a_l = a_l - a_j \times \Psi_i \bmod p;$ 
            end for
        end for
         $m = 2 \times m;$ 
         $k = k/2;$ 
    end while
    return  $a(x);$ 

```

While NTT is heavily optimized at algorithmic level, its existing hardware implementations on FPGAs are rather straightforward. A popular high-level architecture is shown in Figure 4.1, where the Cooley-Tukey radix-2 decimation-in-time is illustrated and the inputs/outputs are read/written to memories. The inputs to the butterfly circuits in Figure 4.1 are two polynomials coefficients, a and b , and $pow(w)$ is a precomputed twiddle factor.

Figure 4.1: Cooley-Tukey radix-2 Decimation-in-time butterfly.

From Figure 4.1, it is clear that the performance of the multiplier and the modulo p reduction block will determine the performance of NTT computation. On FPGA platform, the multiplier can be designed using multiple high-speed DSP blocks. For the modulo p reduction, several works have been proposed to choose p as a number that possess special arithmetic properties to accelerate the modulo reduction. [78], [79]. However, this causes difficulties to have a fixed architecture supporting multiple values of p . As a consequence, the works in [72], [73] and [76] propose to employ the shift-and-subtract division to implement the modulo p reduction block. For the same reason, the proposed architecture will explore algorithms that can support arbitrary value of p for the modulo p reduction block.

4.3 Proposed Architecture

The proposed architecture will employ Algorithm 4.1 since it is the most optimized algorithm for NTT, to the best of the author's knowledge. To realize Algorithm 4.1, two important design blocks are explored: modulo multiplier and modulo adder/subtractor. For the modulo multiplier with an arbitrary modulus p , besides the shift-and-subtract division, Barrett's method (see Algorithm 4.2) or Montgomery's method (see Algorithm 4.3) can be employed.

Algorithm 4.2 Barrett Modulo Multiplication

Input: A, B, p where $A, B < p, 2^{n-1} \leq p < 2^n$

Precomputation: $C = \lfloor 2^{2n+3}/p \rfloor$

Output: $R = AB \bmod p$

Procedure:

1. $P = A \times B;$
 2. $T_0 = \lfloor P / 2^{n-2} \rfloor;$
 3. $T_1 = T_0 \times C;$
 4. $Q = \lfloor T_1 / 2^{n+5} \rfloor;$
 5. $T_2 = Q \times p;$
 6. $R = P - T_2;$
 7. if $R \geq p$ then $R = R - p;$
 8. return $R;$
-

Algorithm 4.3 Montgomery Modulo Multiplication

Input: A, B, M where $A, B < p, 2^{n-1} \leq p < 2^n$

Precomputation: $C = -p^{-1} \bmod 2^n$

Output: $R = AB2^{-n} \bmod p$

Procedure:

1. $P = A \times B;$
 2. $T_0 = P \bmod 2^n;$
 3. $T_1 = T_0 \times C;$
 4. $Q = T_1 \bmod 2^n;$
 5. $T_2 = Q \times p;$
 6. $R = (P + T_2)/2^n;$
 7. if $R \geq p$ then $R = R - p;$
 8. return $R;$
-

4.3.1 Simplified Modulo Multiplier

The integer multiplications (Steps 1, 3 and 5 of Algorithms 4.2 and 4.3) of Barrett and Montgomery algorithms are usually implemented using the fast DSP multipliers on FPGAs. In most cases, the bit-width of the required multiplication is not the same as the bit-width of the DSP multipliers, which leads to inefficient computations. The algorithmic properties of either Barrett's or Montgomery's method can be more efficiently exploited to optimize the design in ASIC implementation.

Considering the computation of T_0 and Q in Barrett's method (see Algorithm 4.2). Since both T_0 and Q are obtained from a rounded down division by a power of 2, a truncated multiplier design can be used instead of a slower, larger and more power-hungry full-width multipliers. However, as they introduce errors into T_0 and Q , a greater number of subtractions by p will be required at Step 7 to bring R into the range $[0, p)$. Therefore, in the proposed design, T_0 and Q are implemented by discarding $(n-2)$ and $(n+5)$ least significant bits (LSBs), respectively from the output of a full-width multiplier. The arithmetic units for Steps 5 and 6 can be simplified by considering the fact that after Step 6, the maximum value of R is $2p - 1$, which is an $(n+1)$ -bit number. For Step 6, instead of a $2n$ -bit subtractor, an $(n+1)$ -bit subtractor is implemented. The inputs to the $(n+1)$ -bit subtractor are the $n+1$ LSBs of P and T_2 . Thus, for Step 5, the logic required for the computation of the most significant bits (MSBs) of T_2 can be saved since they are not needed. This results in the proposed "reduced" Barrett's multiplier shown in Figure 4.2 without incurring additional subtractions of p in Step 7.

Figure 4.2: Hardware architecture of simplified Barrett multiplier.

Considering the computation of T_0 in Montgomery's method (see Algorithm 4.3). It is not possible to reduce the multiplier design by discarding the n MSBs of P since those MSBs are required at Step 6. On the other hand, Q can be computed by a reduced multiplier since the n MSBs of T_1 are not required for the subsequent steps. Therefore, Steps 3 and 4 can both be realized in the reduced multiplier as shown in Figure 4.3 without a T_1 bus. Step 6 is normally

implemented by a $2n$ -bit adder. However, by taking advantage of the fact that the division in Step 6 is exact [50], this step can be sped up as follows. Since the n LSBs of the sum of P and T_2 are all zeros, R is either equal to $P/2^n + T_2/2^n + 1$ or $P/2^n + T_2/2^n$, depending on the carry generated by the sum $P \bmod 2^n + T_2 \bmod 2^n$. In the proposed simplified design, both sums $P/2^n + T_2/2^n + 1$ and $P/2^n + T_2/2^n$ are evaluated in parallel and the correct value of R in Step 6 is determined by the output from the carry generation logic (CGL) block. The delay for this step is reduced from the delay of a $2n$ -bit adder to the delay of an n -bit adder and a multiplexer. This results in the proposed “reduced” Montgomery multiplier shown in Figure 4.3.

Figure 4.3: Hardware architecture of simplified Montgomery multiplier.

4.3.2 Modulo Adder and Subtractor

Figure 4.4(a) and 4.4(b) show the implementation of the modulo p adder and modulo p subtractor, respectively. The inputs of the adder are three n -bit numbers, A , B and p , such that $A, B < p$. Since $0 \leq A + B \leq 2p - 2$, only a single subtraction by p is sufficient to keep the result in the range $[0, p)$. The inputs of the subtractor are three n -bit numbers, A , B and p , such that $A, B < p$. Since $-p + 1 \leq A - B \leq p - 1$, only a single addition by p is sufficient to keep the result in the range $[0, p)$.

Figure 4.4: Hardware architecture of (a) modulo adder and (b) modulo subtractor.

4.3.3 Overall Architecture

Figure 4.5 shows the proposed hardware architecture for NTT, which is based on Algorithm 4.1. The core components include a butterfly circuit built from the modulo arithmetic units presented in previous sections (Section 4.3.1, Section 4.3.2, and Section 4.3.3), a ROM (S) to hold N precomputed powers of ψ_{2N} , where each power is $\lceil \log(p) \rceil$ bits wide, a controller and two RAMs, RAM_0 and RAM_1 . Each RAM holds $N/2$ coefficients where each coefficient is $\lceil \log(p) \rceil$ bits wide. The butterfly circuit receives two coefficients and generates two new values per clock cycle. Storing all N coefficients in a single RAM will lead to idle cycles in the butterfly circuit. Modern RAMs usually support dual-port access, which means that two values can be read/written in one clock cycle. Since the indices of two coefficients entering the butterfly always have different parities [80], by having two dual-port RAMs working in parallel, the butterfly circuit can be executed continuously with no idle cycle. Moreover, two parallel RAMs, each of half the size of a single RAM, can operate at a higher operating frequency than a single RAM.

Figure 4.5: Hardware architecture for the computation of NTT.

Note that in Figure 4.5, for the butterfly circuit built with Montgomery multiplier, every constant stored in S is replaced by its value scaled by $2^n \bmod p$. Since the result of Montgomery's method is not $AB \bmod p$ but $AB2^{-n} \bmod p$, forward transformation into the Montgomery domain at the start of NTT and backward transformation at the end of NTT are necessary [71]. In the proposed architecture, this transformation is eliminated by modifying the N powers of ψ_{2N} stored in the bit-reversed order vector S . It is observed that the inputs of every modulo multiplication within the NTT are a coefficient and a precomputed value loaded from S . Let b be a coefficient and $\Psi' = \Psi 2^n \bmod p$ be a value loaded from the scaled S . The application of Montgomery multiplication on b and Ψ' produces the correct result of $b\Psi \bmod p$ because:

$$b\Psi'2^{-n} \bmod p = b\Psi 2^n 2^{-n} \bmod p = b\Psi \bmod p \quad (4.4)$$

4.4 Evaluation and Comparison

From Figure 4.2 and Figure 4.3, the simplified Barrett multiplier costs approximately two full n -bit by n -bit multipliers, one reduced n -bit by n -bit multiplier, two n -bit subtractors and one multiplexer while the area cost of the simplified Montgomery multiplier is approximately two full n -bit by n -bit multipliers, one reduced n -bit by n -bit multiplier, three n -bit subtractors (as the cost of an adder and the cost of a subtractor are approximately equal), two multiplexers and a CGL block. The delay of the simplified Barrett multiplier can be approximated by the sum of the delays of two full n -bit by n -bit multipliers, one reduced n -bit by n -bit multiplier, two n -bit subtractors and one multiplexer. According to Algorithm 4.3, since T_0 is obtained from the n LSBs of P , the total delay of the Montgomery multiplier can be estimated by the sum of the delays of one full n -bit by n -bit multiplier, two reduced n -bit by n -bit multipliers, two n -bit subtractors and two multiplexers. Based on gate-level component estimation, the butterfly circuit built from the simplified Barrett multiplier is smaller and slower than the butterfly circuit built from the simplified Montgomery multiplier.

The butterfly circuits based on the simplified Barrett and Montgomery multipliers are modeled by Verilog codes, synthesized and mapped to the TSMC 40nm CMOS process technology using Synopsys Design Compiler. The silicon area and critical path delay are shown in Table 4.1. The synthesis results validate their relative cost and performance estimated by the above technology-independent complexity analysis at component-level.

Table 4.1: Delay (D) and area (A) of the butterfly circuit constructed with either the simplified Barrett or Montgomery multiplier.

N	p	Butterfly Barrett		Butterfly Montgomery	
		D (ns)	A (μm^2)	D (ns)	A (μm^2)
256	7681	3.81	9864.88	3.27	12506.89
256	1049089	4.68	20952.29	4.00	24359.75
512	12289	4.00	10981.44	3.43	13262.76
512	5941249	4.87	22098.86	4.28	23912.87
1024	1061093377	5.38	32578.19	4.71	34969.64
2048	$2^{57}+25 \times 2^{13}+1$	6.79	108821.03	6.05	112465.01

The RAMs and ROM in Figure 4.5 of the proposed implementations are generated by the ARM memory compiler with TSMC 40nm technology. Pipelined flip-flops are then inserted into the synthesized butterfly circuit according to the maximum operating frequency reported by the memory compiler. Therefore, an optimal number of pipeline stages is achieved, which in turn helps maximizing the throughput. The performance of the proposed design is then compared with the performances of the architectures presented in [72] and [73] under different parametric combinations of N and p . The results are shown in Figure 4.6, Figure 4.7, Figure 4.8 and Figure 4.9.

Figure 4.6: Comparison of normalized synthesized area of ASIC re-implementation of FPGA designs and the proposed designs with either Barrett or Montgomery multiplier under different combination of (N, p) .

Figure 4.7: Comparison of frequency of designs on FPGA, their ASIC re-implementation, and the proposed designs with either Barrett or Montgomery multiplier under different combination of (N, p) .

Figure 4.8: Comparisons of required cycles per NTT of other designs and the proposed designs with either Barrett or Montgomery multiplier under different combination of (N, p) .

Figure 4.9: Comparisons of normalized NTT operation per second of designs on FPGA, their ASIC reimplementation, and the proposed designs with either Barrett or Montgomery multiplier under different combination of (N, p) .

As can be seen from Table 4.1, the butterfly circuit with Montgomery multiplier is faster than that with Barrett multiplier. However, this difference does not make a noticeable distinction in the cycle counts in Figure 4.8 since the rounded ratio between the delay of the butterfly circuit and the minimum clock period of the memories is the same regardless of the type of multiplier. Thus, at this point, it can be concluded that NTT with Barrett multiplier is more efficient while NTT with Montgomery multiplier may offer better performance if the minimum clock period of the memories is small enough to create a difference between the two types of multiplier. From Figure 4.7, compared to FPGA implementation, the ASIC re-implementation of the designs in [72] and [73] has increased the operating frequency and led to significant improvement in throughput, as shown in Figure 4.9. From Figure 4.6, Figure 4.7 and Figure 4.8, compared to the ASIC re-implementation of the designs in [72] and [73], in general, the proposed custom architecture has lower latency and higher operating frequency at the cost of higher silicon area. For small (N, p) such as $(256, 7681)$, compared to the ASIC re-implementation of the FPGA design, the proposed custom designs achieve 1.26% throughput improvement at the cost of at least 7% more silicon

area. For large (N, p) such as the last three pairs, the proposed custom designs are clearly preferred to the ASIC re-implementation of FPGA design. The throughput has been increased from 28.38% to 54.85% at the cost of at most 7.5% hardware overhead. Therefore, the designs in [72] and [73] are more suitable for small parameter sets while the proposed custom designs are more efficient for large parameter sets that are frequently encountered in time-critical homomorphic encryption.

4.5 Conclusion

In this chapter, the performance of Number Theoretic Transform on ASIC is explored. A custom design is proposed by optimizing the modulo multiplier designs of Barrett and Montgomery multiplication algorithm. The simplified Barrett multiplier exploits truncation without incurring extra modulo reduction operation while the simplified Montgomery multiplier eliminates the conversion overheads. The throughput is enhanced by having two dual-port RAMs working in parallel and pipeline registers inserted according to the access time of RAMs. Experimental results show that the custom design is more suited to high-performance applications.

Chapter 5

Privacy Preserving Polynomial Evaluation via Residue Number System and Fuzzy Vault

5.1 Introduction

In Chapter 3 and Chapter 4, algorithm and architecture to further accelerate the performance of multiplication of polynomials that have very high degree and very large coefficient have been explored on ASIC platform. Though non-trivial improvements have been achieved, they are still not sufficient to make homomorphic encryption schemes practical for many applications. When compared to other works, the residue-to-binary converter in Chapter 3 shortens the conversion delay by 2.2 to 6.8 times, and the custom architecture in Chapter 4 can improve the NTT throughput by 1.28 to 1.54 times. To make homomorphic encryption schemes practical in most applications, order of magnitude improvement is needed.

The reason behind this is the noise growth in the state-of-the-art homomorphic encryption schemes. Despite the effort to minimize it, noise growth is still a persistent problem in homomorphic encryption, where it imposes large ciphertext size and slows down performance substantially. In this chapter, a method that frees us from the concern of noise growth and achieve small ciphertext size, as well as a better performance, is proposed. The proposed method allows users to outsource a polynomial computation to another party without disclosing user's inputs. This is achieved via a combination of residue number system and fuzzy vault. While fuzzy vault helps to secure user's information, residue number system enables homomorphism while separating noise term from the message.

The rest of this chapter is organized as follows. Section 5.2 provides a brief background of residue number system, fuzzy vault and existed homomorphic

encryption schemes. Section 5.3 introduces the system model and privacy requirements. Section 5.4 describes the construction of the proposed method . Section 5.5 presents its security proof. In Section 5.6, the use of the proposed scheme for several privacy-sensitive applications is discussed. The implementation result of the proposed method is reported, discussed, and compared with other works in Section 5.7. The chapter is concluded in Section 5.8.

5.2 Background

5.2.1 Residue Number System

As defined and introduced in Section 3.2, an RNS is defined by a set of n coprime integer numbers $\{m_1, m_2, m_3, \dots, m_n\}$ known as the base of the RNS. The dynamic range M of an RNS is defined be the product of all integers in the base $\{m_1, m_2, m_3, \dots, m_n\}$. An integer X in the range $[0, M)$ can be uniquely represented by a residue representation, which is n -tuple $(x_1, x_2, x_3, \dots, x_n)$, where each residue x_i can be determined by $X \bmod m_i$. Given the set $P = \{(m_i, x_i)\}_{i=1}^n$, X can be recovered by applying Chinese remainder theorem on P .

$$X = CRT(P) = \left(\sum_{i=1}^n M_i (M_i^{-1} x_i \bmod m_i) \right) \bmod M \quad (5.1)$$

where M_i is defined as M/m_i and M_i^{-1} is the multiplicative inverse of $M_i \bmod m_i$, i.e., $M_i^{-1} M_i = 1 \bmod m_i$.

Let Z be the result of a modulo M arithmetic operation $\diamond$ on X and Y , where $\diamond \in (+, -, \times)$. The residue representation of X , Y , and Z satisfy the following equations:

$$\begin{aligned} z_1 &= x_1 \diamond y_1 \bmod m_1 \\ z_2 &= x_2 \diamond y_2 \bmod m_2 \\ &\vdots \\ z_n &= x_n \diamond y_n \bmod m_n \end{aligned} \quad (5.2)$$

Thus, residue number system is said to express homomorphic properties with respect to addition, subtraction, and multiplication.

5.2.2 Fuzzy Vault

Fuzzy vault is a cryptographic construction that has been widely studied for authentication or securing secret information using biometric features [81], [82], [83]. Fuzzy vault works as follows. Supposed that a user would like to lock his/her secret data under a set S and the secret data can be encoded into a univariate polynomial $p(x)$. Considering the elements of S as distinct values of the variable x , the user computes the evaluation of $p(x)$ on the elements of S . This process can be viewed as a mapping of the elements of S onto points lying on the polynomial $p(x)$. After that, a number of chaff points. i.e., random points that do not lie on $p(x)$, are generated. The collection of points, those belong to $p(x)$ and those do not belong to $p(x)$, is called the fuzzy vault and can be made public. Given the fuzzy vault, the set S helps to identify points that are not chaff points, which in turn can be used to reconstruct the polynomial $p(x)$ and the secret data. On the other hand, without the set S , attackers cannot distinguish between chaff points and points lying on $p(x)$. This guarantees the security of the fuzzy vault.

5.2.3 Homomorphic Encryption with Noise

Early homomorphic encryption schemes [17], [18], [19], [20], [21] have their ciphertext in the form of b^e where b and e are the base and the exponent respectively. When the message lies on the exponent, homomorphism with respect to addition is achieved. When the message lies on the base, homomorphism with respect to multiplication is achieved. However, homomorphism with respect to addition and multiplication cannot be achieved simultaneously (see Chapter 2). State-of-the-art homomorphic encryption schemes overcome this drawback by having the ciphertext as a linear function of the message. For security reasons, a plaintext m is marked with a small noise

term to conceal it from attackers. Let t , q , c , e , and sk be a plaintext modulus, a ciphertext modulus, a ciphertext, a small noise term, and a secret key respectively. In general, the decryption structures of these noisy homomorphic cryptosystems are in the form of the dot product of the ciphertext and the secret key $\langle c, sk \rangle$, and can be divided into two categories: Brakerski, Gentry and Vaikuntanathan fully homomorphic encryption without bootstrapping (BGV) style [29] and Fan and Vercauteren (FV) somewhat practical fully homomorphic encryption (FV) style [47]. In BGV, $\langle c, sk \rangle = m + te \pmod{q}$, and the message m can be recovered by a modulo t operation as long as the term te is small enough not to wrap around modulo q . In FV, $\langle c, sk \rangle = (q/t)m + e$, and the message m can be recovered by scaling the dot product by t/q and rounding the result to the nearest integer. For this operation to be successful, the noise term e should not be greater than $q/2t$. Although the approach with noise can enable both homomorphism with respect to addition and multiplication, unlike early schemes where infinite homomorphic multiplication or addition can be carried out, only a limited number of continuous homomorphic operations can be allowed to bound the magnitude of the noise term. Theoretically, this limitation can be removed via the process of bootstrapping, i.e., express the decryption process as a circuit of addition and multiplication and then homomorphically evaluate it [23]. However, the cost of this bootstrapping is so expensive that in practice, given a function that is to be homomorphically evaluated, q is usually chosen large enough to cover the largest possible noise [84], [85], [86], [87].

5.3 System and Privacy Models

5.3.1 System Model

The objective of this work is to securely outsource a polynomial computation from one party to another without leaking its private data. The system model is shown in Figure 5.1. Two entities, party A and party B , are involved in the exchange of information.

Figure 5.1: System model under consideration.

In Figure 5.1, party A is an entity that wish to use the computational resources provided by party B to evaluate a polynomial function on its own data while keeping its data private. Party A is responsible for encrypting its data before sending them to party B , instructing party B to carry out the desired polynomial function over the encrypted data, and decrypting the result computed by party B . Party B is assumed to have ‘infinite’ computational resources and is tasked to perform operations over party A ’s encrypted data according to the instructions sent by party A .

5.3.2 Privacy Model

In the privacy model of our system, party B is considered to be a curious-but-honest party, i.e., party B will strictly follow the instruction from party A but is interested to know the private data of party A . In addition, an active adversary A^* is introduced into the model. The ultimate purpose of adversary A^* is to obtain the plaintexts from the ciphertexts generated by party A . Adversary A^* is assumed to have the capability to eavesdrop the communication link between party A and party B or to compromise party B to obtain the ciphertexts of party A ’s data. However, it is restricted from compromising party A .

5.4 Proposed Scheme

5.4.1 Intuition

Adding a noise term to the message in the homomorphic encryption scheme mentioned in Section 5.2.3 can be considered as mixing a precious chemical

substance P with a junk substance E in a barrel. The secret key can be viewed as a special substance S that can eliminate the junk substance from the precious substance, and homomorphic operations can be viewed as heating the barrel, thus increasing the volume of the junk substance. If the volume of the junk substance reaches a certain threshold, the barrel is not large enough to contain it. As a consequence, the exceeded volume of substances - a mix of precious substance and junk substance - get out of the barrel and evaporate. Therefore, the special substance S cannot be used to recover the original precious substance P since parts of it has evaporated. This phenomenon is similar to the noise growth in homomorphic encryption schemes in Section 5.2.3. Correct message cannot be recovered via decryption if the noise term reaches a specific level (q in BGV and $q/2t$ in FV). Therefore, q should be chosen as a very large integer, and similarly the volume of the barrel should be very big to prevent substances leakage.

Instead of adding junk, substance P can be divided into many parts and stored in identical barrels. To hide the barrels containing substance P , the junk substance E is also divided into many parts and stored in identical barrels. Successful recovery of substance P requires combining all the barrels containing it and the ability to distinguish barrels containing parts of P from barrels containing parts of E . Failing to do so will result in combining junk barrels with barrels of precious substances. The more parts P and E can be divided, the harder it is to recover all parts belong to P . Since junk substance is separated from precious substance, it is not necessary to worry about junk evaporation due to heating that can carry away the precious substance. In this way, it is not necessary to keep a big barrel. The same concept can be applied to homomorphic encryption if there is a way to divide the message into many parts and hide them with many other noisy parts. The next section shows how this concept can be achieved via residue number system and fuzzy vault.

5.4.2 Construction

The proposed scheme is a modified version of the original fuzzy vault presented in [88]. Firstly, the algorithm ENCRYPT is presented. The basic idea here is to

encode the secret message as a residue representation with respect to a secret set. This encoding method helps to achieve the concept of “dividing the message into many small parts” and allows homomorphism due to the mathematical property of residue number system. The residue representation of the secret message can be viewed as a set of tuples. Each tuple is a pair of modulus and its corresponding residue (Lines 2-5 of the algorithm ENCRYPT). To conceal the secret message, chaff pairs, which are pairs of random modulus a and residue b such that b differs from the secret message modulo a , are generated and added into the set of pairs representing the secret message (Lines 6-10 of the algorithm ENCRYPT).

Algorithm: ENCRYPT

Public parameters: A set of r coprime integers U

Input: Parameters n such that $n \leq r$. A secret message m .

A secret set $S = \{s_i\}_{i=1}^n$ consisting of n distinct elements uniformly chosen from U .

Output: $R = \text{ENCRYPT}(m)$ - a set of r pairs $\{(a_i, b_i)\}_{i=1}^r$ such that $b_i < a_i$ and a_i 's are distinct.

Procedure:

1. $A, R \leftarrow \emptyset$;
 2. for $i = 1$ to n do
 3. $(a_i, b_i) \leftarrow (s_i, m \bmod s_i)$;
 4. $A \leftarrow A \cup \{a_i\}$;
 5. $R \leftarrow R \cup \{(a_i, b_i)\}$;
 6. for $i = n+1$ to r do
 7. $a_i \in_u U - A$;
 8. $A \leftarrow A \cup \{a_i\}$;
 9. $b_i \in_u \mathbb{Z}_{a_i} - \{m \bmod a_i\}$;
 10. $R \leftarrow R \cup \{(a_i, b_i)\}$;
 11. return R ;
-
-

In the algorithm ENCRYPT, symbols $\emptyset$, $\cup$, and $-$ stand for empty set, set union, and set difference respectively. $\in_u$ implies uniformly random selection

from a set. Z_a denotes the finite ring of integers modulo a . To protect the information about the order in which a_i are chosen, the set R is randomly permuted before making it public.

The ciphertext can be decrypted as follows. The ciphertext, which is the set R generated by the algorithm ENCRYPT, is a collection of pairs of modulus and residue. Some of these pairs are genuine, i.e., they represent the secret message. The chaff pairs, which are pairs of random modulus and residue, are generated in such a way that they do not intersect with the secret set S and the residues of the secret message. To successfully recover the secret message, the secret S must be required to distinguish the genuine modulus – residue pairs from the chaff pairs. The procedure to decrypt the ciphertext is shown in the algorithm DECRYPT. Line 3 of DECRYPT assigns a unique pair that has the modulus s_i from R to (m_i, x_i) . The recovered set of genuine pairs

$$P = \{(s_i, m \bmod s_i)\}_{i=1}^n \subset R.$$

Algorithm: DECRYPT

Input: Secret set $S = \{s_i\}_{i=1}^n$, ciphertext R consisting of r modulus-residue pairs $\{(a_i, b_i)\}_{i=1}^r$ such that $b_i < a_i$ and a_i 's are distinct.

Output: A message $m = \text{DECRYPT}(R)$ such that

$$\{(s_i, m \bmod s_i)\}_{i=1}^n \subseteq R$$

Procedure:

1. $P \leftarrow \emptyset$;
 2. for $i = 1$ to n do
 3. $(m_i, x_i) \xleftarrow{s_i} R$;
 4. $P \leftarrow P \cup \{(m_i, x_i)\}$;
 5. $m = \text{CRT}(P)$;
 6. Output m ;
-
-

5.4.3 Homomorphic Operations

Let R_1 and R_2 be the output ciphertexts when applying the algorithm ENCRYPT on the messages, m_1 and m_2 , respectively. The encryption of $m_1 \diamond m_2$, where $\diamond \in (+, -, \times)$, can be computed by manipulating components of the ciphertexts, R_1 and R_2 , as given in the following algorithm.

Algorithm: COMP

Input: Operation $\diamond$, ciphertexts $R_1 = \{(a_{1i}, b_{1i})\}_{i=1}^r$ and $R_2 = \{(a_{2i}, b_{2i})\}_{i=1}^r$ such that $R_1 = \text{ENCRYPT}(m_1)$ and $R_2 = \text{ENCRYPT}(m_2)$

Output: Ciphertext $R_3 = \{(a_{3i}, b_{3i})\}_{i=1}^r$ such that $R_3 = \text{ENCRYPT}(m_1 \diamond m_2)$

Procedure:

1. $R_3 \leftarrow \emptyset$;
 2. for $i = 1$ to r do
 3. $(m_{temp}, x_{temp}) \leftarrow \frac{a_{1i}}{a_{2i}} R_2$;
 4. $(a_{3i}, b_{3i}) \leftarrow (m_{temp}, b_{1i} \diamond x_{temp} \bmod m_{temp})$;
 5. $R_3 \leftarrow R_3 \cup \{(a_{3i}, b_{3i})\}$;
 6. return R_3 ;
-
-

The correctness of this algorithm is straightforward. In the algorithm COMP, for each modulus-residue pair of R_1 , its modulus is used as a key to look for a pair sharing the same modulus in R_2 (Line 3). Since the ciphertexts, R_1 and R_2 , are generated under the public parameter U , the set of moduli collected from modulus-residue pairs of R_1 must consists of r distinct coprime integers, and are equal to the one generated from R_2 . Thus, the pair-matching from the ciphertext R_1 to the ciphertext R_2 can be considered as a one-to-one operation. With this one-to-one matching, the genuine pairs representing m_1 are mapped to the genuine pairs representing m_2 while the chaff pairs in R_1 are matched to the

chaff pairs in R_2 . For each matching, operation $\diamond$ is applied on the residues with the matched modulus (Line 4). Thus, in this process, there will eventually be n generated genuine modulus-residue pairs and $(r - n)$ generated chaff pairs. By the homomorphism property of RNS (Equation (5.2)), the generated genuine modulus-residue pairs can form the residue representation of $m_1 \diamond m_2$ with respect to the secret set S . To recover $m_1 \diamond m_2$ from R_3 , one can rely on the secret set S to filter out the genuine pairs, and then applying the Chinese Remainder Theorem (Equation (5.1)) on them.

For the purpose of illustration, an example is given to show how the algorithms ENCRYPT, DECRYPT, and COMP work. Let $r = 8$, $n = 4$, $U = \{211, 223, 227, 229, 233, 239, 241, 251\}$, and $S = \{211, 227, 233, 241\}$. Let $m_1 = 12345$. Applying the algorithm ENCRYPT on m_1 , the elements of S are used to generate $n = 4$ genuine pairs:

$$\begin{aligned} (211, m_1 \bmod 211) &= (211, 107) \\ (227, m_1 \bmod 227) &= (227, 87) \\ (233, m_1 \bmod 233) &= (233, 229) \\ (241, m_1 \bmod 241) &= (241, 54) \end{aligned} \tag{5.3}$$

Next, the elements of U that do not belong to S , which are $\{223, 229, 239, 251\}$, are used to generate the chaff modulus-residue pairs. Each residue of these chaff pairs (a_i, b_i) is uniformly selected from the elements of the finite ring Z_{a_i} that are different from the modulo reduction of m_1 by the modulus of the chaff pair. Since $m_1 \bmod 223 = 80$, $m_1 \bmod 229 = 208$, $m_1 \bmod 239 = 156$ and $m_1 \bmod 251 = 46$, b_1 can be randomly selected from $Z_{223} - \{80\}$, b_2 can be randomly selected from $Z_{229} - \{208\}$, b_3 can be randomly selected from $Z_{239} - \{156\}$, and b_4 can be randomly selected from $Z_{251} - \{46\}$. Suppose b_1, b_2, b_3 , and b_4 are selected to be 81, 205, 161 and 39, respectively. By adding these chaff pairs, $(223, 81)$, $(229, 205)$, $(239, 161)$ and $(251, 39)$, into the genuine pairs and then randomly permuting the elements, the set R_1 representing the

encryption of m_1 is formed.

$$R_1 = \left\{ \begin{array}{l} (211,107), (227,87), (223,81), (251,45), \\ (239,161), (241,54), (233,229), (229,205) \end{array} \right\} \quad (5.4)$$

Let $m_2 = 54321$. The ciphertext R_2 of m_2 can be similarly computed by the algorithm ENCRYPT. The ciphertext R_2 after shuffling the chaff and genuine pairs is:

$$R_2 = \left\{ \begin{array}{l} (211,94), (251,97), (223,134), (241,96), \\ (227,68), (239,74), (229,44), (233,32) \end{array} \right\} \quad (5.5)$$

Let $\diamond$ be the operation $\times$. The algorithm COMP to find the encryption R_3 of $m_1 \diamond m_2$, given R_1 and R_2 , is illustrated in Figure 5.1. The dashed lines represent pair matching operation (Line 3) and the solid lines indicate the result of the operation $\times$ applied on each matched pair.

Figure 5.2: Illustrative example of multiplying two ciphertexts by algorithm COMP.

The resulted ciphertext R_3 is:

$$R_3 = \left\{ \begin{array}{l} (211,141), (251,18), (223,150), (241,123), \\ (227,14), (239,203), (229,89), (233,105) \end{array} \right\} \quad (5.6)$$

To decrypt R_3 , the secret set $S = \{211, 227, 233, 241\}$ is used to filter out the genuine modulus-residue pairs hidden inside R_3 . These pairs are:

$$\{(m_i, x_i)\}_{i=1}^4 = \{(211, 141), (227, 14), (233, 105), (241, 123)\} \quad (5.7)$$

By applying the CRT (Equation (5.1)) on the recovered pairs, with $M = 211 \times 227 \times 233 \times 241 = 2689560241$, $M_1 = M/211 = 12746731$, $M_2 = M/227 = 11848283$, $M_3 = M/233 = 11543177$ and $M_4 = M/241 = 11160001$, we have:

$$\begin{aligned} & \left(\sum_{i=1}^4 M_i (M_i^{-1} x_i \bmod m_i) \right) \bmod M \\ &= \left(\begin{array}{l} 12746731 \times (190 \times 141 \bmod 211) \\ + 11848283 \times (164 \times 14 \bmod 227) \\ + 11543177 \times (171 \times 105 \bmod 233) \\ + 11160001 \times (155 \times 123 \bmod 241) \end{array} \right) \bmod 2689560241 \quad (5.8) \\ &= 670592745 \end{aligned}$$

It can be easily verified that the obtained number is equal to $m_1 \diamond m_2$, which is 12345×54321 .

5.5 Security

In the proposed construction, the genuine modulus-residue pairs, which must be required to recover a message, are hidden among a number of chaff pairs inside a ciphertext. Thus, similar to the original fuzzy vault construction [88], the security of the proposed scheme depends on the number of chaff pairs $(r - n)$ in a ciphertext R . The more chaff pairs, the harder it becomes for an attacker to guess the message m . As many chaff pairs are inserted into R , a number of counterfeit messages that look like m , i.e., the messages that can be computed by any n modulus-residue pairs chosen from R , start to appear. A curious-but-honest adversary A^* defined in Section 5.3.2, without the information from the secret set S , is unable to distinguish the secret message m from the collection

of all possible counterfeit messages.

The number of chaff pairs in a fresh ciphertext generated by ENCRYPT is always $r-n$. However, due to the number theoretic property of modulo arithmetic, COMP can, with certain probability, turn several chaff pairs of the input ciphertexts into genuine pairs of the resulted ciphertext. For instance, in the example in Section 5.4.3, (223, 81) and (223, 134) are chaff pairs of ciphertexts R_1 and R_2 , respectively. If chaff pair (223, 23) is generated instead of (223, 134) by the random chaff pair generation process of ENCRYPT, then the operation on chaff pairs (223, 81) and (223, 23) becomes $(223, 81 \cdot 23 \bmod 223) = (223, 79)$. Since $670592745 \bmod 223 = 79$, (223, 79) is no longer a valid chaff pair but a genuine pair of ciphertext R_3 . This is called a corrupted chaff pair. Since corrupted chaff pairs can leak information about the hidden message, a ciphertext output by COMP is less secure than a ciphertext generated by ENCRYPT. In the following subsections, the security of ENCRYPT and COMP are analyzed to determine the parameters required to secure the proposed scheme.

5.5.1 Analysis of ENCRYPT

The following lemma and its proof are a modification of Lemma 4 in [88]. It proves that with high probability, there are many integers that can be recovered by n modulus-residue pairs uniformly chosen from R . Remember that in the algorithm ENCRYPT, a secret message m is encoded by n moduli specified by a secret set S and concealed by $(r-n)$ chaff modulus-residue pairs. The number of counterfeit messages depends on n , r , and the set of moduli U .

Lemma 1: Let α be a positive real number less than 1. Without loss of generality, assume that the set $U = \{u_i\}_{i=1}^r$ is arranged in increasing order. The set R generated by the algorithm ENCRYPT on message m and the secret set S fulfils

the following property: There exists at least $\alpha C_n^r \prod_{i=1}^n \frac{1}{u_{r-i+1}} \prod_{j=1}^{r-n} \left(1 - \frac{1}{u_j}\right) \prod_{i=1}^n u_i$

integers m' that agree with n modulus-residue pairs in R , i.e., R contains exactly

n elements of the form $(a, m' \bmod a)$, with a probability of at least $(1 - \alpha)$.

Proof: Let S' be a random set formed by n moduli chosen from the elements of U . The number of possible choices for S' is C_n^r . For a modulus-residue pair (a, b) and a random integer, the probability that b is and is not equal to the remainder of this integer divided by a is $1/a$ and $(1 - 1/a)$, respectively. Thus, the probability for a random integer to agree with only n elements of R implied by S' is:

$$C_n^r \prod_{x \in S'} 1/x \prod_{y \in U - S'} (1 - 1/y) \quad (5.9)$$

When $S' = \{u_{r-i+1}\}_{i=1}^n$, this probability achieves its minimum value. Therefore:

$$C_n^r \prod_{x \in S'} 1/x \prod_{y \in U - S'} (1 - 1/y) \geq C_n^r \prod_{i=1}^n \frac{1}{u_{r-i+1}} \prod_{j=1}^{r-n} \left(1 - \frac{1}{u_j}\right) \quad (5.10)$$

Recall that in the algorithm ENCRYPT, the secret message m is transformed into its residue representation with respect to n moduli of S . Thus, the dynamic range of an RNS defined by S to represent a message m should be at least $\prod_{i=1}^n u_i$. Consequently, the expected number of integers that agree only with n elements of R is at least:

$$D = C_n^r \prod_{i=1}^n \frac{1}{u_{r-i+1}} \prod_{j=1}^{r-n} \left(1 - \frac{1}{u_j}\right) \prod_{i=1}^n u_i \quad (5.11)$$

Let G be a bipartite graph constructed as follows. The left vertices relate to the integer numbers that can be represented by the moduli in the secret set S . The right vertices relate to all residue representations with respect to the moduli in the set U such that there is one vertex for one residue representation. A right vertex $\{y_i\}_{i=1}^r$ is a neighbor of a left vertex x if x corresponds to exactly n residue in $\{y_i\}_{i=1}^r$, i.e., there are exactly n residues y_i that satisfy $y_i = x \bmod u_i$. The previous analysis shows that the expected degree of a right vertex is at least D .

Remember that the algorithm ENCRYPT chooses a left vertex and outputs a random right vertex that is a neighbor to the chosen left vertex. It is necessary to prove that with high probability the output right vertex has sufficiently high degree. Since the graph G is symmetric about its left vertex, there exists an automorphism that maps a left vertex x to another left vertex x' . Thus, similar to Lemma 4 in [88], it is sufficient to argue that a random right vertex has its degree at least αD . The number of vertices with degree less than αD accounts for α portion of the vertices in G . The lemma is proved.

As an example, choose $r = 400$ and $n = 50$. Hence for every message, its encryption contains 400 modulus-residue pairs, in which 50 of them are genuine while the rest are chaffs. To satisfy the requirement that the elements of the set U are coprime, U is simply constructed as a set of 400 prime numbers. For a back-of-the-envelope calculation, assume that the set $U = \{u_i\}_{i=1}^r$ is arranged in increasing order. It is easy to see the following inequality:

$$D = C_n^r \prod_{i=1}^n \frac{1}{u_{r-i+1}} \prod_{j=1}^{r-n} \left(1 - \frac{1}{u_j}\right) \prod_{i=1}^n u_i > C_n^r \left(\frac{u_1}{u_r}\right)^n \left(1 - \frac{1}{u_1}\right)^{r-n} \quad (5.12)$$

Assume that the elements of U are 16 bits in length. Such assumption makes it easier to construct U since there are more than 3000 16-bit prime numbers [89]. Consider $u_1 = 32771$ (the smallest 16-bit prime number) and $u_{400} = 65521$ (the largest 16-bit prime number). With such parameters, it is expected that for any secret set S consisting of 50 elements chosen from U , given a ciphertext R , the number of integers that agree exactly with 50 modulus-residue pairs from R is greater than 2^{162} . Specifically, with probability of at least $(1 - 2^{-81})$, there are more than 2^{81} numbers expressing the same property. Thus, it can be said that the chosen parameters accomplish a security level of at least 81 bits. Now with the same $u_1 = 32771$, consider $u_{400} = 36899$ (the 399th prime number after 32771). With these new parameters, it is expected to have more than 2^{204} integers that agree exactly with 50 modulus-residue pairs from a ciphertext R for any secret set S consisting of 50 elements chosen from U . With the new parameters, at least a 102-bit security level is achieved. This phenomenon can be explained

by looking at Equation (5.6). Note that the lower bound of D depends on the difference between u_1 and u_r . A small difference will make the ratio u_1 / u_r close to one, increase the security level, and vice versa. Therefore, parameters such that u_1 / u_r is close to one are recommended.

5.5.2 Analysis of COMP

Let c be the number of corrupted chaff pairs of a ciphertext R output by COMP. The following lemma gives a lower bound on the number of counterfeit messages that can be recovered from the ciphertext R which contains c corrupted chaff pairs.

Lemma 2: Let α be a positive real number less than 1. Without loss of generality, assume that the set $U = \{u_i\}_{i=1}^r$ is arranged in increasing order. If a ciphertext R output by COMP contains c corrupted chaff pairs, then there exists at least $\alpha C_{n+c}^r \prod_{i=1}^{n+c} \frac{1}{u_{r-i+1}} \prod_{j=1}^{r-n-c} \left(1 - \frac{1}{u_j}\right) \prod_{i=1}^n u_i$ integers m' that match $n+c$ modulus-residue pairs in R , i.e., R contains exactly $n+c$ elements of the form $(a, m' \bmod a)$, with a probability of at least $(1 - \alpha)$.

Proof: Lemma 2 can be proved in a manner similar to Lemma 1. If a ciphertext R output by COMP contains c corrupted chaff pairs, the total number of genuine and corrupted chaff pairs will be $n+c$. Given such knowledge, an adversary will try to recover the secret message hidden inside R by picking $n+c$ pairs from R . Let S' be a random set formed by $n+c$ moduli chosen from the elements of U . The number of possible choices for S' is C_{n+c}^r , and the probability for a random integer to agree with only $n+c$ elements of R is given by:

$$C_{n+c}^r \prod_{x \in S'} 1/x \prod_{y \in U-S'} (1-1/y) \quad (5.13)$$

When $S' = \{u_{r-i+1}\}_{i=1}^{n+c}$, this probability has its minimum value. Therefore:

$$C_{n+c}^r \prod_{x \in S'} 1/x \prod_{y \in U-S'} (1-1/y) \geq C_{n+c}^r \prod_{i=1}^{n+c} \frac{1}{u_{r-i+1}} \prod_{j=1}^{r-n-c} \left(1 - \frac{1}{u_j}\right) \quad (5.14)$$

In ENCRYPT, the secret message m is transformed into its residue representation with respect to n moduli of the secret set. As the total number of integers that can be represented by the RNS defined by the secret set is at least $\prod_{i=1}^n u_i$, the expected number of integers that agree only with $n+c$ elements of R is at least:

$$D = C_{n+c}^r \prod_{i=1}^{n+c} \frac{1}{u_{r-i+1}} \prod_{j=1}^{r-n-c} \left(1 - \frac{1}{u_j}\right) \prod_{i=1}^n u_i \quad (5.15)$$

By constructing a bipartite graph G similar to the one for Lemma 1, Lemma 2 can be easily proved.

For a back-of-the-envelope calculation, the following inequality from Equation (5.15) is derived:

$$D > C_{n+c}^r \left(\frac{u_1^n}{u_r^{n+c}} \right) \left(1 - \frac{1}{u_1}\right)^{r-n-c} \quad (5.16)$$

As an example, consider the case of $r = 400$ and $n = 50$. With U being the set of the first four hundred 16-bit prime numbers, a ciphertext R generated by ENCRYPT offers at least 102-bit security level (see the example in Section 5.5.1). If the application of COMP on R causes 3 chaff pairs to become corrupted, R will contain 53 genuine and corrupted chaff pairs in total. From Equation (5.16), the number of integers that agree exactly with 53 modulus-residue pairs from the corrupted ciphertext R is about 2^{166} . Specifically, about 2^{83} numbers possess this property with a probability of $1 - 2^{-83}$. Thus, it can be deduced that 3 corrupted chaff pairs will reduce the security level from 102-bit to 83-bit.

5.5.3 Parameters Requirements

Though corrupted chaff pairs can reduce the security level of the proposed scheme, their occurrence in a ciphertext is probabilistic. The following lemma gives an upper bound on the probability that a certain number of chaff pairs can become corrupted.

Lemma 3: Without loss of generality, assume that the set $U = \{u_i\}_{i=1}^r$ is arranged in increasing order. Let R be a ciphertext output by COMP. The probability that R contains more than c corrupted chaff pairs, denoted $\Pr(x > c)$, satisfies

$$\Pr(x > c) \leq \left(\frac{e^\varsigma}{(1+\varsigma)^{1+\varsigma}} \right)^\mu, \text{ where } \mu = \sum_{i=1}^{r-n} 1/u_i \text{ and } \varsigma = \frac{c+1}{\mu} - 1$$

Proof: Let e_1 and e_2 be two numbers randomly chosen from the integer ring Z_q . For any element e_3 in Z_q , the probability that $e_3 = e_1 \diamond e_2 \bmod q$ is $1/q$, where $\diamond$ is either addition, subtraction, or multiplication modulo q . Thus, the probability of corrupting a chaff pair is $1/q$, where q is the modulus of that chaff pair.

Let $Q = \{q_i\}_{i=1}^{r-n}$ be the set of moduli corresponding to the chaff pairs in ciphertext R . Let x_i be a random variable such that $x_i = 1$ with probability $1/q_i$ and $x_i = 0$ with a probability of $1 - 1/q_i$. Let x be a random variable denoting the number of corrupted chaff pairs in the ciphertext R . Then:

$$x = x_1 + x_2 + x_3 + \dots = \sum_{i=1}^{r-n} x_i \quad (5.17)$$

The random variable x can be treated as the sum of $r-n$ Poisson trials $x_1, x_2, x_3, \dots, x_{r-n}$ with $\Pr(x_i = 1) = 1/q_i$. The expected value of x is given by:

$$\mu = E[x] = \frac{1}{q_1} + \frac{1}{q_2} + \frac{1}{q_3} + \dots = \sum_{i=1}^{r-n} \frac{1}{q_i} \quad (5.18)$$

For a given positive ς , the following bound holds [90]:

$$\Pr(x \geq (1 + \varsigma)\mu) \leq \left(\frac{e^\varsigma}{(1 + \varsigma)^{1+\varsigma}} \right)^\mu \quad (5.19)$$

Thus, the probability for more than c chaff pairs to become corrupted is:

$$\Pr(x > c) \leq \left(\frac{e^\varsigma}{(1 + \varsigma)^{1+\varsigma}} \right)^\mu \quad (5.20)$$

where $\varsigma = \frac{c+1}{\mu} - 1$

Since the probability for a chaff pair to become corrupted is inversely proportional to the magnitude of its corresponding modulus, the probability for having corrupted chaff pairs in a ciphertext R is maximized when:

$$Q = \{q_i\}_{i=1}^{r-n} = \{u_i\}_{i=1}^{r-n} \quad (5.21)$$

The lemma is proved by substituting Equation (5.21) into Equation (5.18).

To guarantee security, the parameter set must be chosen to satisfy the following requirements:

- a) A ciphertext generated by ENCRYPT should agree with a large number of counterfeit messages. This number is determined via Lemma 1.
- b) The chosen parameter set should tolerate up to c corrupted chaff pairs caused by COMP. The number of counterfeit messages is determined via Lemma 2.
- c) The probability of more than c corrupted chaff pairs should be sufficiently low, $\leq 2^{-128}$ for example. This probability is determined via Lemma 3.

5.6 Applications

5.6.1 Scenarios

The proposed scheme targets the following two scenarios. In the first scenario, party A would like to outsource a computation task on its data to another party

B , for example the cloud, but becomes concerned about the privacy of the data being kept in cloud storages, which is not under the control of party A . Party A may or may not wish to leak its computation model to party B . In the second scenario, party A would like to rely on a model provided by party B to extract useful information from party A 's data. While party A wishes to keep its data private, party B does not want to leak its intellectual property model to party A . In Figure 5.3, how to use the proposed scheme to outsource computation for the first scenario when the outsourced function is a public polynomial is illustrated.

Figure 5.3: Outsourcing a public computation by the proposed method.

Suppose that party A holds private data $x, y, z, \dots$ and would like to have party B evaluating the polynomial $func()$ on its private data. After both parties make agreement on the public set U , party A applies the algorithm ENCRYPT on its private data $x, y, z, \dots$ under the same secret set S . After receiving the ciphertexts and the description of polynomial $func()$ from party A , party B appropriately applies the algorithm COMP for a number of times on the ciphertexts to homomorphically evaluate the polynomial $func()$ and sends the encrypted result back to party A . Once receiving the encrypted result, party A applies the algorithm DECRYPT with the secret S to recover the evaluation of polynomial $func()$ on its private data $x, y, z, \dots$. In case party A wishes to keep the polynomial $func()$ private to itself, it can apply the algorithm ENCRYPT to

the coefficients of $func()$ and send the ciphertexts of coefficients to party B . For the second scenarios where $func()$ is an intellectual property of party B , to carry out a homomorphic operation between a coefficient and a ciphertext, party B only needs to apply the operation on the coefficient with each of the residue of the ciphertext, and perform modulo reduction with the corresponding modulus. It is not necessary for party B to obtain the encryption of every coefficient of polynomial $func()$.

5.6.2 Statistical Functions

The explosive growth of Internet of Things (IoT) devices has created a massive amount of digital data that have never been seen before. Since the analysis of the IoT data requires extensive computational power, it is reasonable for a client to offload this analysis to a powerful party such as the cloud. This section shows how to use the proposed method to compute the mean and variance, which are the most popular measurements in statistical analysis, while guarantee the privacy of the client's data.

Supposed that data collected by the client can be represented as a set of K numbers $\{c_i\}_{i=1}^K$. The mean and variance of the collected data, denoted by $\bar{c}$ and $\text{var}(c)$, can be computed as follows:

$$\bar{c} = \frac{c_1 + c_2 + c_3 + \dots + c_K}{K} = \frac{\sum_{i=1}^K c_i}{K} \quad (5.22)$$

$$\text{var}(c) = \frac{1}{K} \sum_{i=1}^K (c_i - \bar{c})^2 \quad (5.23)$$

To outsource the computation of the mean and variance to the cloud without revealing the collected data, the first algorithm ENCRYPT is applied on each term and they are sent to the cloud. However, there are two issues to obtain the encryption of the mean and variance from the encrypted data. Computing the encryption of $\sum_{i=1}^K c_i$ and $\sum_{i=1}^K (c_i - \bar{c})^2$ are straightforward due to the homomorphism property of RNS, but without decrypting these ciphertexts, dividing them by K to get the encryption of $\bar{c}$ and $\text{var}(c)$ is impossible unless it

is an exact division, which is not always the case. The second issue also originates from the nature of RNS: it can only represent integer numbers. However, in practice, data collected are not merely integers, they are often real numbers.

The second issue is easy to deal with. Before applying the algorithm ENCRYPT, each c_i is scaled up by a certain factor F and then rounded to the nearest integer. Let $sc_i \approx F \times c_i$ be the scaled and rounded value of c_i . The following relation holds:

$$\sum_{i=1}^K sc_i \approx \sum_{i=1}^K F \times c_i = F \times \sum_{i=1}^K c_i \quad (5.24)$$

$$\begin{aligned} \sum_{i=1}^K \left(sc_i - \frac{\sum_{i=1}^K sc_i}{K} \right)^2 &= \sum_{i=1}^K \left(F \times c_i - \frac{\sum_{i=1}^K F \times c_i}{K} \right)^2 \\ &= F^2 \times \sum_{i=1}^K \left(c_i - \frac{\sum_{i=1}^K c_i}{K} \right)^2 \\ &= F^2 \times \sum_{i=1}^K (c_i - \bar{c})^2 \end{aligned} \quad (5.25)$$

The client, after decrypting the ciphertext returned from the cloud, needs to scale down the result by a proper factor to obtain the correct answer. From Equation (5.24), to obtain $\sum_{i=1}^K c_i$, the decrypted result should be scaled down by F . Similarly, a scale factor of F^2 should be applied to get $\sum_{i=1}^K (c_i - \bar{c})^2$. For practical reason, the scale factor F can be chosen as a power of two 2^α so that each real number c_i can be approximated by an integer with the precision up to α binary digits after the binary point. With such values of F , scaling up and scaling down are merely cheap left and right shift operations, respectively.

The difficulty of dividing by K in RNS requires the client to play a role in determining $\text{var}(c)$: the client needs to decrypt the encryption of $\sum_{i=1}^K c_i$, divide the result by K , then encrypt $\bar{c}$ and send it to the cloud. To avoid this cumbersome process, Equation (5.23) is modified as follows:

$$\begin{aligned}
\text{var}(c) &= \frac{1}{K} \sum_{i=1}^K (c_i - \bar{c})^2 = \frac{1}{K} \sum_{i=1}^K \left(c_i - \frac{\sum_{i=1}^K c_i}{K} \right)^2 \\
&= \frac{1}{K} \sum_{i=1}^K \left(\frac{Kc_i}{K} - \frac{\sum_{i=1}^K c_i}{K} \right)^2 \\
&= \frac{1}{K^3} \sum_{i=1}^K (Kc_i - \sum_{i=1}^K c_i)^2
\end{aligned} \tag{5.26}$$

With Equation (5.26), the mentioned process can be replaced by a division by K^3 . The problem of division in RNS can be mitigated if K is a fixed parameter. The term $1/K$ can be approximated as follows:

$$\frac{1}{K} = \frac{2^\beta}{K} \times \frac{1}{2^\beta} \simeq \left[\frac{2^\beta}{K} \right] \times \frac{1}{2^\beta} \tag{5.27}$$

where $\left[\cdot \right]$ denotes rounding-to-the-nearest-integer function

Since K is a fixed parameter, the term $\left[2^\beta/K \right]$ can be precomputed. Division by K is replaced by multiplication by $\left[2^\beta/K \right]$, which is feasible on the ciphertext space. For correctness, the client needs to scale down the decrypted value by 2^β .

In summary, the procedure for the client to outsource the mean and variance computation to the cloud without revealing its collected data is as follows:

- Step 1: For each datum c_i , the client scales it up by 2^α and rounds it to the nearest integer sc_i .
- Step 2: The client applies the algorithm ENCRYPT to each sc_i and its encryption $Enc(sc_i)$ to the cloud.
- Step 3: The cloud applies the algorithm COMP on $Enc(sc_i)$ to perform the following calculation and send the results back to the client.

$$\left[\frac{2^\beta}{K} \right] \sum_{i=1}^K Enc(sc_i) \tag{5.28}$$

$$\left(\left[\frac{2^\beta}{K}\right]\right)^3 \sum_{i=1}^K \left(K \times Enc(sc_i) - \sum_{i=1}^K Enc(sc_i)\right)^2 \quad (5.29)$$

- Step 4: To get $\bar{c}$, the client decrypts the result from Equation (5.28) and scales it down by $2^{\alpha+\beta}$. To get $var(c)$, the client decrypts the result from Equation (5.29) and scales it down by $2^{2\alpha+3\beta}$.

5.6.3 Predictive Analysis for Medical Applications

Over the past few years, the healthcare industry has always been a lucrative target for cyber attackers. A notable case is the loss of 1.5 million patient records including those related to government officers last year [91]. Despite the increasing awareness of healthcare providers to protect themselves, criminals are more likely to come up with creative attacks and the healthcare industry will continue to be vulnerable. This section illustrates how a patient can use the proposed method to have a healthcare provider predicts if she is vulnerable to developing a specific illness, yet revealing no information about her health status. Specifically, the following health models are considered: the logistic regression model and the Cox proportional hazard model [92].

The logistic regression model and Cox proportional hazard model have been widely used to study the relation between a specific disease and a patient's health information such as age, body mass index, blood pressure, etc... In [93], Boyed et al. used logistic regression to develop the Trauma and Injury Severity Score method, which later becomes popular to predict mortality of injured victim. In [94], Cox model was employed to study the effectiveness of various treatments against breast cancer. Other examples include those presented in [95], [96], [97] and [98]. For the purpose of demonstration, the logistic regression model from [99] and the Cox proportional hazard model from [98] are adopted.

The logistic regression model from [99] predicts the possibility of a patient to have a heart attack based on the following health variables from the patient: age (a), systolic blood pressure (sys), diastolic blood pressure (dia), cholesterol level

(*chol*), height (*ht*), and weight (*wt*). The possibility output by the model is determined by the sigmoid function as follows:

$$P_L(x) = \frac{e^x}{e^x + 1} \quad (5.30)$$

where x is the weighted sum of the health variables:

$$x = 0.072a + 0.013sys - 0.029dia + 0.008chol - 0.053ht + 0.021wt \quad (5.31)$$

The Cox proportional hazard model from [98] forecasts the probability of a patient to experience general cardiovascular disease, given the following health variables from the patient: age (a), cholesterol level (*chol*), HDL cholesterol level (*hdl*), systolic blood pressure (*spb*), smoker (s) and diabetes (d). The model is described as follows:

$$P_C(y) = 1 - 0.95012^{\exp(y)} \quad (5.32)$$

where y is the weighted sum of the health variables:

$$y = 2.32888 \log(a) + 1.20904 \log(chol) - 0.70833 \log(hdl) + 2.76157 \log(spb) + 0.52873s + 0.69154d - 26.1391 \quad (5.33)$$

Two issues prevent the use of the proposed scheme for the evaluation of the above health models on encrypted data. First, the proposed scheme supports only integers as inputs. Second, it allows only polynomial computation. The first issue makes it hard to compute weighted sum of health variables while the second issue causes incompatibility due to the non-polynomial nature of $P_L(x)$ and $P_C(y)$. The first issue can be alleviated by applying the scaling and rounding method in the previous section. Equation (5.32) and Equation (5.33) can be normalized with proper precision as follows:

$$X = 72a + 13sys - 29dia + 8chol - 53ht + 21wt \quad (5.34)$$

$$Y = 232888 \log(a) + 120904 \log(chol) - 70833 \log(hdl) + 276157 \log(spb) + 52873s + 69154d - 2613910 \quad (5.35)$$

For the second issue, the recommendation from [85] and [86] are adopted. Specifically, a non-polynomial function can be approximated by a polynomial via Taylor expansion [85] or Remez minimax therapy [86]. Supposed the domain of

the function that needs to be approximated is limited in the interval (A,B) . Taylor method achieves excellent accuracy near the expansion point but approximation error becomes noticeable as the value of input moves toward to the interval ends A and B . On the other hand, Remez therapy reduces this error at the cost of flattening the error all over the interval domain.

$P_L(x)$ is approximated by Taylor method as:

$$P_L(x) \approx \frac{1}{2} + \frac{1}{4}x - \frac{1}{48}x^3 + \frac{1}{480}x^5 - \frac{17}{80640}x^7 + \frac{31}{1451520}x^9 \quad (5.36)$$

$P_C(y)$ is approximated by Remez method as:

$$P_C(y) \approx \frac{5.511}{10^2} + \frac{4.706}{10^2}y + \frac{1.048}{10^2}y^2 + \frac{8.302}{10^3}y^3 + \frac{6.069}{10^3}y^4 - \frac{2.761}{10^4}y^5 - \frac{6.022}{10^4}y^6 + \frac{2.624}{10^5}y^7 + \frac{1.860}{10^5}y^8 - \frac{1.754}{10^6}y^9 \quad (5.37)$$

Similar to Equation (5.32) and Equation (5.33), Equation (5.36) and Equation (5.37) need to be normalized to be compatible with the proposed method.

$$P_L(X) = 725760 + 362880X - 30240X^3 + 3024X^5 - 306X^7 + 31X^9 \quad (5.38)$$

$$P_C(Y) \approx 55110000 + 47060000Y + 10480000Y^2 + 8302000Y^3 + 6069000Y^4 - 276100Y^5 - 602200Y^6 + 26240Y^7 + 18600Y^8 - 1754Y^9 \quad (5.39)$$

Equation (5.34) and Equation (5.38) will be used to evaluate the logistic regression model while Equation (5.35) and Equation (5.39) are meant for Cox proportional hazard model. The procedure for the patient to outsource the computations of the two health models to the cloud without revealing her health data is similar to the previous section. The inputs to Equation (5.34) and Equation (5.35) should be scaled and rounded so that they can be represented as integers. To obtain the correct result, the patient should apply a proper scaling down constant after decrypting the result returned by the cloud.

5.7 Performance and Comparison

5.7.1 Benchmarking Settings

The proposed scheme is initiated with the following parameters: $r = 1600$, $n = 100$, and the public set U consisting of the first 1600 16-bit prime numbers. According to the security analysis (see Lemma 1 to Lemma 3), when the number of corrupted chaff pairs increases from 0 to 17, the security level of our system is reduced from 237-bit to 136-bit. The probability that there are more than 17 corrupted chaff pairs is less than 2^{-134} . Therefore, these parameters will guarantee more than 128-bit security. The proposed scheme and candidate applications are coded in C++ using NTL [100] and GMP [101] libraries. The platform specification of our work and others are summarized in Table 5.1. Notice that in [84], the author did not mention the version of the Intel Core 2 Duo processor that they use, hence the version is assumed to be T6500, the weakest Intel Core 2 Duo processor according to Geekbench [102].

Table 5.1: Implementation environment of the proposed work and others.

	Specification	Geekbench Score
This work	Intel Core-i7 7920HQ @3.1GHz	14287
[87]	Intel Core-i7 5930K @3.5GHz + NVIDIA GeForce GTX980	N/A
[86]	Intel Core i5-3337U @1.8GHz	4476
[85]	Intel Core i7-3520M @2.9GHz	6112
[84]	Intel Core 2 Duo T6500 @2.1 GHz	1955

In Table 5.1, various designs are implemented on different platforms, thus it is problematic to make a fair comparison among their performance. To mitigate this issue, the processor score report from Geekbench [102] is used. It is a popular test that has been adopted by processor design companies like MediaTek, ARM, Samsung and AMD to measure the performance of their products. The platform score of this work and others are shown in the rightmost column of Table 5.1. The higher the score, the better the performance, and double the score means double the performance. Note that the score for the

platform [87] is not included since the design is implemented on a CPU accelerated by a GPU. Nevertheless, its CPU score (22231) is the highest compared to this work and others works that are implemented on CPUs.

Table 5.2 summarizes the performance results of the proposed scheme and they are compared to [84], [85], and [87]. Due to the difference in implementation platform, the performance results of [84], [85] and [87] are normalized according to the Geekbench score reported in Table 5.1 with reference to the score of the platform of this work. For example, the reported encryption time of [85] is 577 ms and its platform score is 6112. Thus, the scaled encryption time of [85] is calculated to be $577 \times 6112 \div 14287 = 246.84$ ms. It can be seen from Table 5.2 that the CPU implementation of the proposed scheme outperforms other CPU implementations, [84] and [85], by at least 3.88 times. Compared to the GPU-accelerated CPU implementation in [87], despite the large difference in computation power, the CPU implementation of this work still achieves a speedup of at least 3.48 times. The most noticeable improvement offered by the proposed scheme is in homomorphic multiplication. Its CPU implementation is at least 42.18 times faster than other CPU implementations of [84] and [85], and 6.3 times faster than the CPU-GPU implementation of [87]. These results will be further discussed in Section 5.5.3.

Table 5.2: Comparison of the performance for different operations between this work and others. Results are reported in milliseconds.

Operation		Encryption	Decryption	H. Addition	H. Multiplication
This work		0.194	0.112	0.036	0.133
[85]	Original	577	549	1	5056
	Scaled	246.84	234.86	0.43	2162.96
Speedup over [85]		1272.37	2096.96	11.94	16262.85
[84]	Original	24	26	1	41
	Scaled	3.28	3.56	0.14	5.61
Speedup over [84]		16.90	31.78	3.88	42.18
[87]	Original	0.16	1	0.07	0.838
Speedup over [87]		0.82	8.92	1.94	6.3

5.7.2 Applications Performance

The performance results of the statistical functions described in Section 5.6.2 and the predictive analysis models (logistic regression model and Cox proportional hazard model) described in Section 5.6.3 are summarized in Table 5.3. It can be seen that the CPU implementation of this work outperforms other CPU implementations of [84], [85], [86] and [87] by at least 20.45 times, and the CPU-GPU implementation [87] by 5.34 times. These improvements are expected due to the shorter time of each operation of the proposed implementation compared to others, which can be seen in Table 5.2.

Table 5.3: Comparison of the performance of candidate applications between this work and others. Results are reported in milliseconds. M&V: mean and variance. LRM: logistic regression model. CPHM: Cox proportional hazard model.

Application		M&V	LRM	CPHM
This work		40.275	2.705	4.640
[84]	Original	6020	N/A	N/A
	Scaled	823.77	N/A	N/A
Speedup over [84]		20.45	N/A	N/A
[85]	Original	N/A	80196	N/A
	Scaled	N/A	34307.97	N/A
Speedup over [85]		N/A	12683.16	N/A
[86]	Original	N/A	989	719
	Scaled	N/A	309.84	225.27
Speedup over [86]		N/A	114.54	48.54
[87]	Original	N/A	14.46	N/A
Speedup over [87]		N/A	5.34	N/A

5.7.3 Discussion

This section discusses the reason behind the poorer performance of other implementations when compared to the proposed work. Remember that the scheme behind the works in [84], [85], [86] and [87] is homomorphic encryption with noise, as mentioned in Section 5.2.3. The analysis from [103] is employed to illustrate the effect of noise on the size of ciphertext. The homomorphic encryption with noise is defined by the following parameters: plaintext modulus t , ciphertext modulus q , polynomial degree n , and noise standard deviation σ .

To encrypt an integer, first it should be encoded as a polynomial with degree of at most n and maximum coefficient value at most t . Hence, n and t should be carefully chosen such that the maximum possible integer created by the targeted computation task still can be represented as a polynomial with degree at most n and maximum coefficient value at most t . A ciphertext has the form of a pair of polynomials with degree at most n and coefficient at most q . A fresh ciphertext, i.e., ciphertexts that have not undergone any homomorphic operation, contains the polynomial encoding an integer message masked with a noise polynomial whose coefficients are sample from a Gaussian distribution with standard deviation σ .

Supposed that the maximum absolute value of coefficients of a noise polynomial inside a fresh ciphertext is B . Summing s fresh ciphertexts together will result in a ciphertext with a noise polynomial whose coefficient value is at most sB . Multiplication of two ciphertexts, on the other hand, causes the coefficients of the noise polynomial to increase significantly due to cross-terms multiplication. To study the effect of noise growth, the authors of [103] use a circuit consisting of D rounds, where each round includes an evaluation of A additions followed by a multiplication. The magnitude of coefficient of the noise polynomial inside the ciphertext resulted from evaluating the mentioned circuit, according to the analysis from [103], is bounded by:

$$\left(14tn2^A\right)^D \frac{42\sigma tn}{q} \quad (5.40)$$

For decryption to work correctly, this term needs to be less than $1/2$, which gives:

$$q > \left(14tn2^A\right)^D 84\sigma tn \quad (5.41)$$

For $A = 0$ and $D = 0$, i.e., no support for homomorphic operation, the homomorphic encryption with noise scheme becomes a public-key encryption scheme and q can be chosen as a small number. This kind of scheme has been proved to have good performance in practice. It takes only a few milliseconds for encryption and decryption, though the implementation is realized on 8-bit micro controller platform [70], [71]. However, with fixed plaintext space and fixed

number of additions per round, the magnitude of ciphertext modulus q grows exponentially with the multiplication level D . Thus, q should be chosen as a very large number in homomorphic encryption with noise implementations, which explains its poor performance. In the proposed scheme, operation on ciphertexts does not cause noise growth, thus a better performance than homomorphic encryption with noise can be achieved. Table 5.4 summarizes the form and size of ciphertext in the implementation of the proposed scheme and others, [84], [85], [86] and [87]. It is easy to see that the ciphertext multiplication in the proposed scheme is preferred to high degree large coefficient polynomial multiplication in other works.

Table 5.4: Comparison of ciphertext size and form between the proposed scheme and others.

This work	1600 modulus-residue pairs, 16-bit modulus, 16-bit residue
[84]	2 polynomials with 2048 degree and 58-bit coefficient
[85]	1 polynomial with 16384 degree and 512-bit coefficient
[86]	2 polynomials with 13217(9487) degree and 229(328)-bit coefficient for LRM(CPHM)
[87]	31 polynomials with 1024 degree and 31-bit coefficient

5.8 Conclusion

This chapter proposes a method to outsource a polynomial computation without sacrificing privacy. The method takes advantage of RNS and fuzzy vault to isolate the effect of noise growth during computation to the size of the ciphertexts. Candidate applications, namely mean-variance, logistic regression model and Cox proportional hazard model, are evaluated using the proposed method. Smaller ciphertext size due to noise growth isolation boosts up the performance of the proposed method compared to homomorphic encryption with noise in [84], [85], [86] and [87]. For logistic regression model, the CPU implementation of the proposed method achieves a speedup of 5.34 times over the implementation in [87], which use a powerful computing CPU-GPU platform.

Chapter 6

Conclusions and Future Works

6.1 Conclusions

Alongside the convergence of multidisciplinary technologies of cloud, big data analytics, wireless connectivity, pervasive sensing and internet of things, the intensive data collection and processing in these systems also inflict rampant data breaches and growing privacy concerns on offloading computations to remote servers. Homomorphic encryption, which allows direct computation on encrypted data, is a promising solution to privacy-preserving outsourced computations. Despite tremendous recent progresses, the escalated ciphertext size with noise growth in homomorphic encryption and modulo reduction of large integer arithmetic on high degree polynomials are identified as the major performance bottleneck in the algorithm and circuit implementation of homomorphic encryption. This thesis aims to develop methods to boost the performance of homomorphic encryption towards its application viability. While Chapter 3 and Chapter 4 focus on developing hardware to accelerate the performance bottleneck of arithmetic operations in state-of-the-art schemes, Chapter 5 is devoted to a new approach to prevent the noise growth from exacerbating the ciphertext size via the homomorphism of residue number systems and the fuzziness offered by fuzzy vault construction.

In Chapter 3, a new algorithm for accelerating the residue-to-binary conversion of very high cardinality arbitrary moduli set has been presented. The proposed algorithm avoids very large operand arithmetic and very large modulo reduction of state-of-the-art CRT-based algorithms by transforming a large number of residues of arbitrary moduli into only three residues before the final conversion. The three residue channels are independent, which enables modulo arithmetic operation of smaller operand to be simplified by precomputation in the first step. The size of each of the three residues in the second step has been reduced to

only one third of the dynamic range of the original arbitrary set that defines the RNS. The modulo reduction operation is further accelerated by choosing the three moduli set as $\{2^n - 1, 2^n, 2^n + 1\}$ to take advantage of its special number theoretic properties for modulo arithmetic operations. As a result, the final step of the conversion can be completed by highly efficient adder-based residue-to-binary converter. Compared with other converter for the same moduli sets with different cardinality and modulus's size, significant improvements in terms of delay at the cost of negligible silicon area have been demonstrated by the logic synthesis experiments.

In Chapter 4, ASIC performance of Number Theoretic Transform towards accelerating very high degree polynomial multiplication is explored with a custom architecture. To minimize the latency of the butterfly circuit, the algorithmic properties of either Barrett or Montgomery modulo multiplication algorithm are exploited for the designs of two simplified modulo multipliers. Idle cycles are avoided by having two dual-port RAMs working in parallel and supporting four memory's accesses in the same clock cycle. Optimal number of pipeline stages is achieved by inserting pipelined flip-flops into the butterfly circuits according to the access time of the memories. The custom architecture avoids redundant area and unnecessary pipelined registers when compared to directly porting existing architecture on FPGA to ASIC. The logic synthesis experiments show that for large parameter sets, the proposed custom architecture has better performance at the cost of negligible hardware overhead, making it more suited to polynomial multiplication for homomorphic encryption.

Though the performance of homomorphic encryption can be improved by combining techniques presented in Chapter 3 and Chapter 4, the result is not significant enough to alleviate the hurdle caused by noise growth. Chapter 5 moves away from hardware optimizations by presenting a new approach to prevent noise from growing yet still support homomorphism. The proposed approach incorporates RNS and fuzzy vault. While fuzzy vault helps to assure the confidentiality of user's information, RNS enables homomorphism and separates noise term from the message, thus preventing its growth. The

proposed scheme is applied to evaluate privacy-sensitive applications, namely statistical functions and predictive analysis. Smaller ciphertext size due to noise growth isolation boosts up the performance of the proposed method by at least an order of magnitude compared to homomorphic encryption with noise, taking into consideration the difference in computing platform for benchmarking.

6.2 Future Works

Based on the research works presented in this thesis, the following ideas are recommended for future exploration.

- a) All homomorphic encryption schemes assumed that the party performing outsourced computation, the cloud for example, is semi-honest. It means that the cloud will carry out the outsourced computation as instructed but may become curious about the client's data. Besides, homomorphic encryption schemes are malleable by default. A curious cloud or an attacker can apply homomorphic operation on ciphertexts, obtain a valid ciphertext, and send it to the client. The client can decrypt the received ciphertext but has no method to discover that the ciphertext has been modified. The mentioned problem can be alleviated by extending RNS in Chapter 5 to redundant residue number system (RRNS). By adding redundant moduli into the existing moduli set of an RNS, an RRNS can be formed to detect and correct errors caused by computation. Thus, RRNS is a good candidate to alleviate problems caused by the malleability of homomorphic encryption. Computation inside the redundant moduli should be carried out by the client itself so that the redundant residues cannot be modified by untrusted parties. The more redundant moduli the client keeps, the lesser opportunities the attacker has to modify the ciphertext without being detected. The client can be freed from keeping the redundant moduli by adapting to a multi-cloud environment instead of a single-cloud environment, as shown in Figure 6.1. The client can assign each cloud the computation of a single modulus or multiple moduli by capitalizing on the distributed and independent arithmetic among residues of different modulus channels. As long as the number of compromised clouds is small enough, the compromised clouds can be

detected by exploiting the error detection capability of RRNS. The limitation of the scenarios in Figure 6.1 is that if all clouds are compromised, then RRSN loses its edge. Besides, increasing the capability to detect error, i.e., increasing the number of redundant moduli of RRNS, will place a heavier computation burden for detecting corrupted clouds on the client, which is often considered to have limited computation power. These limitations shall be addressed in the future works.

Figure 6.1: Deploying RRNS in multi clouds settings.

- b) Today, users often rely on different service providers to make inferences on their personal data. Recently, there is a trend that the service providers use machine learning to make accurate inferences. To protect user's personal data, further study is needed to explore the proposed scheme for evaluating neural network on encrypted personal data. We start by studying common function in neural networks. The convolution function is compatible with the proposed scheme since it is a weighted sum plus a bias. Nonlinear activation can be made compatible with the proposed scheme via polynomial approximation (see Chapter 5). Besides activation functions, there are also other functions that are not compatible with the proposed method, namely the average function and max function. For average function, if the divisor is

known in advance, the method presented in Chapter 5 – Section 5.6.2 for computing mean and variance can be applied to replace the division by precomputation and multiplication. Max function cannot be directly approximated by the polynomial approximation. However, its power can be approximated thanks to the relation $\max(x_1, x_2, x_3, \dots, x_n) = \lim_{d \rightarrow \infty} \left(\sum_i x_i^d \right)^{1/d}$.

The higher d is, the better we can approximate the max function, but it comes with higher computational cost. Besides, with approximation, there is a problem that whether approximation has a bad effect on the accuracy of the neural network. This problem will be studied in the future.

Author's Publications

[1] **Truong Phu Truan Ho** and Chip-Hong Chang, “Accelerating residue-to-binary conversion of very high cardinality moduli set for fully homomorphic encryption”, in *Proceedings of 2016 IEEE Asia Pacific Conference on Circuits and Systems*, Jeju, Korea, pp. 9-12, October 2016.

[2] **Truong Phu Truan Ho** and Chip-Hong Chang, “Towards Ideal Lattice-Based Cryptography on ASIC: A Custom Implementation of Number Theoretic Transform”, in *Proceedings of 2018 IEEE 23rd International Conference on Digital Signal Processing*, Shanghai, China, pp. 1-5, November 2018.

[3] Truong Phu Truan Ho and Chip-Hong Chang, “Privacy Preserving Outsourced Functional Evaluation via Residue Number System and Fuzzy Vault”, submitted to *IEEE Transactions on Dependable and Secure Computing*, (under review)

[4] **Truong Phu Truan Ho** and Chip-Hong Chang, “Accelerating Homomorphic Encryption in Hardware: Algorithms and Architectures”, to appear in Chapter 11, *Frontiers in Hardware Security and Trust*, The Institute of Electrical and Technology (IET) publisher, UK, October 2020.

Bibliography

[1] B. Grot, D. Hardy, P. Lotfi-Kamran, B. Falsafi, C. Nicopoulos, and Y. Sazeides, "Optimizing data center TCO with scale out processors", *IEEE Micro*, vol. 32, pp. 52-63, Sept. 2012.

[2] Synergy Research Group. (2016, January 7). *2015 Review Shows \$110 Billion Cloud Market Growing at 28% Annually* [Online]. Available: www.srgresearch.com.

[3] Cisco Systems, Inc. (2016). *Cisco Global Cloud Index: Forecast and Methodology 2015-2020* [Online]. Available: www.cisco.com.

[4] R. Padilha and F. Pedone, "Confidentiality in the cloud", *IEEE Security & Privacy*, vol. 13, no. 1, pp. 57-60, Feb. 2015.

[5] Ponemon Institute. (2015, January). *2014: A Year of Mega Breaches* [Online]. Available: www.ponemon.org.

[6] Verizon Enterprise. (2014). *2014 Data Breach Investigation Report* [Online]. Available: www.verizonenterprise.com.

[7] Verizon Enterprise. (2015). *2015 Data Breach Investigation Report* [Online]. Available: www.verizonenterprise.com.

[8] Verizon Enterprise. (2016). *2016 Data Breach Investigation Report* [Online]. Available: www.verizonenterprise.com.

[9] Verizon Enterprise. (2017). *2017 Data Breach Investigation Report* [Online]. Available: www.verizonenterprise.com.

[10] Verizon Enterprise. (2018). *2018 Data Breach Investigation Report* [Online]. Available: www.verizonenterprise.com.

- [11] Verizon Enterprise. (2019). *2019 Data Breach Investigation Report* [Online]. Available: www.verizonenterprise.com.
- [12] Intel Security-McAfee. (2014, June). *Net Losses: Estimating the Global Cost of Cybercrime* [Online]. Available: www.mcafee.com.
- [13] Kaspersky Lab. (2015). *Global IT Security Risks 2015* [Online]. Available: www.kaspersky.com.
- [14] Ponemon Institute. (2016, June). *2016 Cost of Data Breach Study: Global Analysis* [Online]. Available: www.ponemon.org.
- [15] Juniper Research. (2015, May). *Cybercrime and the Internet of Threats* [Online]. Available: www.juniperresearch.com.
- [16] R. Rivest, L. Adleman, and M. Dertouzos, "On data banks and privacy homomorphisms", *Foundations of Secure Computation*, pp. 169-180, 1978.
- [17] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol. 21, pp. 120-126, Feb. 1978.
- [18] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Transactions on Information Theory*, vol. 31, pp. 469-472, Jul. 1985.
- [19] S. Goldwasser and S. Micali, "Probabilistic encryption", *Journal of Computer and System Sciences*, vol. 28, pp. 270-299, Apr. 1984.
- [20] J. Benaloh, "Dense probabilistic encryption", in *Proceedings of Workshop on Selected Areas of Cryptography*, pp. 120-128, 1994.
- [21] P. Paillier, "Public-key cryptosystems based on composite degree

residuosity classes”, in *Proceedings of EUROCRYPT 1999*, Prague, Czech Republic, pp. 223-238, May 1999.

[22] O. Baudron, P-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard, “Practical multi-candidate election system”, in *Proceedings of the 20th annual ACM symposium on Principles of distributed computing*, Rhode Island, USA, pp. 274-283, Aug. 2001.

[23] C. Gentry, “A fully homomorphic encryption scheme”, Ph.D. dissertation, Stanford University, 2009.

[24] C. Gentry and S. Halevi, “Implementing Gentry’s fully homomorphic encryption scheme”, in *Proc. Advances in Cryptology - EUROCRYPT 2011*, Tallinn, Estonia, pp. 128-149, May 2011.

[25] W. Wang, Y. Hu, L. Chen, X. Huang, B. Sunar, "Exploring the feasibility of fully homomorphic encryption", *IEEE Transactions on Computers*, vol. 64, no. 3, pp. 698-706, 2015.

[26] V. Vaikuntanathan, “Computing blindfolded: New developments in fully homomorphic encryption”, in *Proceedings of FOCS 2011*, California, USA, pp. 5-16, Oct. 2011.

[27] C. Aguilar-Melchor, S. Fau, C. Fontaine, G. Gogniat, and R. Sirdey, “Recent advances in homomorphic encryption: A possible future for signal processing in the encrypted domain”, *IEEE Signal Processing Magazine*, vol. 30, no. 2, pp. 108-117, Feb. 2013.

[28] C. Moore, M. O’Neill, E. O’Sullivan, Y. Doroz, and B. Sunar, “Practical homomorphic encryption: A survey”, in *Proceedings of 2014 IEEE International Symposium on Circuits and Systems*, Melbourne, Australia, pp. 2792-2795, Jun. 2014.

[29] Z. Brakerski, C. Gentry, V. Vaikuntanathan, "Fully homomorphic encryption

without bootstrapping", *ACM Trans. Comput. Theory*, vol. 6, no. 3, 2014.

[30] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP", in *Proc. 32nd Annu. Cryptol. Conf. Adv. Cryptol.*, Santa Barbara, CA, USA, pp. 868-886, Aug. 2012.

[31] K. Nuida, K. Kurosawa, "(Batch) fully homomorphic encryption over integers for non-binary message spaces", in *Proc. 34th Annu. Int. Conf. Theory Appl. Cryptographic Tech.*, Sofia, Bulgaria, pp. 537-555, Apr. 2015.

[32] T. Lepoint and M. Naehrig, "A comparison of the homomorphic encryption scheme FV and YASHE", in *Proceedings of AFRICACRYPT 2014*, Marrakesh, Morocco, pp. 318-335, May 2014.

[33] W. Wang, Z. Chen, and X. Huang, "Accelerating leveled fully homomorphic encryption using GPU", in *Proceedings of 2014 IEEE International Symposium on Circuits and Systems*, Melbourne, Australia, pp. 2800-2803, Jun. 2014.

[34] S.S. Roy, K. Jarvinen, F. Vercauteren, V. Dimitroz, and I. Verbauwhede, "Modular hardware architecture for somewhat homomorphic function evaluation", in *Proceedings of Cryptographic Hardware and Embedded Systems-CHES 2015*, Saint-Malo, France, pp. 164-184, Sept. 2015.

[35] Y. Doroz, E. Ozturk, E. Savas, and B. Sunar, "Accelerating LTV based homomorphic encryption in reconfigurable hardware", in *Proceedings of Cryptographic Hardware and Embedded Systems-CHES 2015*, Saint-Malo, France, pp. 185-204, Sept. 2015.

[36] T. Popplemann, M. Naehrig, A. Putnam, and A. Macias, "Accelerating homomorphic evaluation on reconfigurable hardware", in *Proceedings of Cryptographic Hardware and Embedded Systems-CHES 2015*, Saint-Malo, France, pp. 143-163, Sept. 2015.

[37] W. Wang, X. Huang, N. Emmart, and C. Weems, "VLSI design of a large

number multiplier for fully homomorphic encryption”, in *IEEE Transactions on Very Large Scale (VLSI) Systems*, vol. 22, no. 9, pp. 1879-1887, 2014.

[38] Y. Doroz, E. Ozturk, and B. Sunar, “Accelerating fully homomorphic encryption in hardware”, in *IEEE Transactions on Computers*, vol. 64, no. 6, pp. 1509-1521, 2015.

[39] W. Dai, Y. Doroz, and B. Sunar, “Accelerating NTRU based homomorphic encryption using GPUs”, in *Proceedings of 2014 IEEE High Performance Extreme Computing Conference*, Waltham, Massachusetts, USA, pp. 1-6, 2014.

[40] X. Yi, R. Paulet, E. Bertino, Homomorphic Encryption and Applications, New York, NY, USA, Springer, 2014.

[41] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, “Fully homomorphic encryption over the integers”, in *Proceedings of EUROCRYPT 2010*, French Riviera, France, pp. 24-43, May 2010.

[42] O. Regev, H. N. Gabow, R. Fagin, "On lattices learning with errors random linear codes and cryptography", in *Proceedings of STOC'05*, Baltimore, MD, USA, pp. 84-93, May 2005.

[43] V. Lyubashevsky, C. Peikert, O. Regev, "On ideal lattices and learning with errors over rings", *Journal of the ACM*, vol. 60, no. 6, pp. 43, 2013.

[44] P. Martins, L. Sousa, A. Mariano, "A survey on fully homomorphic encryption: An engineering perspective", *ACM Comput. Surv.*, vol. 50, no. 6, pp. 33, Dec. 2017.

[45] A. Acar, H. Aksu, A. S. Uluagac, M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation", *ACM Comput. Surv.*, vol. 51, no. 4, pp. 79, Jul. 2018.

[46] J. W. Bos, “Post-quantum key exchange for the internet”, Sept. 2016.

[Online]. Available: <http://www.joppebos.com/presentations/ULB2016.pdf>.

[47] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, vol. 2012, pp. 144, 2012.

[48] N. P. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes", in *Proceedings of PKC 2010*, Paris, France, pp. 420-443, May 2010.

[49] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor", in *Proceedings of Advances in Cryptology*, California, USA, pp. 311-323, 1987.

[50] P. Montgomery, "Modular multiplication without trial division", *Mathematics of Computation*, vol. 44, no. 170, pp. 519-521, 1985.

[51] C. Moore, M. O'Neill, N. Hanley, and E. O'Sullivan, "Accelerating integer-based fully homomorphic encryption using Comba multiplication", in *Proceedings of 2014 IEEE Workshop on Signal Processing Systems*, Belfast, UK, pp. 1-6, Oct. 2014.

[52] X. Cao, C. Moore, M. O'Neill, E. O'Sullivan, N. Hanley, "Optimised multiplication architecture for accelerating fully homomorphic encryption", *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2794-2806, Sept. 2016.

[53] A. Karatsuba and Y. Ofman, "Multiplication of many-digit numbers by automatic computers", in *Proceedings of the USSR Academy Sciences*, no. 2, pp. 293-294, 1962.

[54] M. J. Kronenburg, "Toom-cook multiplication: Some theoretical and practical aspects," February 2016. [Online]. Available: <https://arxiv.org/abs/1602.02740>.

[55] N. Emmart and C. Weems, "High precision integer multiplication with a GPU

using Strassen's algorithm with multiple FFT sizes", *Parallel Processing Letter*, vol. 21, no. 3, pp. 359-375, 2011.

[56] J-S. Coron, D. Naccache, and M. Tibouchi, "Public key compression and modulus switching for fully homomorphic encryption over the integers", in *Proceedings of Advances in Cryptology-EUROCRYPT 2012*, Cambridge, UK, pp. 446-464, April 2012.

[57] P. G. Comba, "Exponentiation cryptosystems on the IBM PC," *IBM Systems Journal*, vol. 29, no. 4, pp. 526–538, 1990.

[58] E. Ozturk, Y. Doroz, E. Savas, and B. Sunar, "A custom accelerator for homomorphic encryption applications", *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 3-16, Jun. 2016.

[59] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, "Improved security for a ring-based fully homomorphic encryption scheme", in *Proceedings of IMACC 2013*, Oxford, UK, pp. 45-64, Dec 2013.

[60] L. Sousa, S. Antao, P. Martins, "Combining residue arithmetic to design efficient cryptographic circuits and systems", *IEEE Circuits Syst. Mag.*, vol. 16, no. 4, pp. 6-32, 4th Quart. 2016.

[61] P. V. A. Mohan, *Residue Number Systems: Algorithms and Architectures*, MA, Norwell:Kluwer, 2002.

[62] A.P. Shenoy and R. Kumaresan, "Fast base extension using redundant modulus in RNS," *IEEE Trans. Comput.*, vol. 38, no. 2, pp. 292-297, Feb. 1989.

[63] Y. Wang, X. Song, M. Aboulhamid, and H. Shen, "Adder based residue to binary number converter for $(2^n-1, 2^n, 2^{n+1})$," *IEEE Trans. Signal Process.*, vol. 50, no. 7, pp. 1772-1779, Jul. 2002.

[64] R. Zimmermann, "Efficient VLSI implementation of modulo $2^n \pm 1$ addition

and multiplication,” in *Proc. 14th IEEE Symp. Computer Arithmetic*, Adelaide, Australia, pp. 158-167, Apr. 1999.

[65] Y. Kong, “Optimizing the improved Barrett modular multipliers for public-key cryptography,” in *Proc. 2010 Int. Conf. on Computational Intelligence and Software Engineering*, Wuhan, China, pp. 1-4, Dec. 2010.

[66] C. Gentry, A. Sahai, B. Waters, “Homomorphic encryption from learning with errors: conceptually simple, asymptotically faster, attribute based,” in *Proc. CRYPTO 2013*, Santa Barbara, CA, USA, pp. 75-92, Aug. 2013.

[67] P. W. Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” in *Proc. 35th Annu. IEEE Symp. Foundations Comput. Sci.*, Santa Fe, NM, USA, pp.124-134, Nov. 1994.

[68] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila, “Post-quantum key exchange for the TLS protocol from the ring learning with errors problem,” in *Proc. 2015 IEEE Symp. Security and Privacy*, San Jose, CA, USA, pp. 553-570, May 2015.

[69] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, “Lattice signatures and bimodal gaussians,” in *Proc. CRYPTO 2013*, Santa Barbara, CA, USA, pp. 40-56, Aug. 2013.

[70] T. Poppelmann, T. Oder, and T. Guneyusu, “High performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers,” in *Proc. LATINCRYPT 2015*, Guadalajara, Mexico, pp. 346-365, Aug. 2015.

[71] Z. Liu, T. Poppelmann, T. Oder, H. Seo, S. S. Roy, T. Guneyusu, J. Grobschadl, H. Kim, and I. Verbauwhede, “High performance ideal lattice-based cryptography on 8-bit AVR microcontrollers,” *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 4, Article 117, pp. 1-24, Sept. 2017.

[72] T. Poppelmann and T. Guneyusu, “Towards efficient arithmetic for lattice-

based cryptography on reconfigurable hardware,” in *Proc. LATINCRYPT 2012*, Santiago, Chile, pp. 139-158, Oct. 2012.

[73] T. Poppelmann and T. Güneysu, “Towards practical lattice-based public key encryption on reconfigurable hardware,” in *Proc. SAC 2013*, Burnaby, BC, Canada, pp. 68-85, Aug. 2013.

[74] T. Oder, T. Güneysu, F. Valencia, A. Khalid, M. O’Neill, and F. Regazzoni, “Lattice-based cryptography: From reconfigurable hardware to ASIC,” in *Proc. ISIC 2016*, Singapore, pp. 1-4, Dec. 2016.

[75] "Homomorphic Encryption Applications and Technology", *Heat-project.eu*, 2019. [Online]. Available: <https://heat-project.eu>.

[76] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, “Compact ring-LWE cryptoprocessor,” in *Proc. CHES 2014*, Busan, South Korea, pp. 371-391, Sept. 2014.

[77] J. W. Cooley, J. W. Tukey, "An algorithm for the machine computation of the complex Fourier series", *Math. Comput.*, vol. 19, pp. 297-301, Apr. 1965.

[78] A. Aysu, C. Patterson, P. Schaumont, "Low-cost and area-efficient FPGA implementations of lattice-based cryptography", in *Proc. 2013 IEEE Int. Symp. Hardware-Oriented Security Trust (HOST)*, Austin, TX, USA, pp. 81-86, Jun. 2013.

[79] Z. Liu et al., "Efficient Implementation of Ring-LWE Encryption on 8-bit AVR Processors", in *Proc. CHES 2015*, Saint-Malo, France, pp. 663-82, Sept. 2015.

[80] J. Baek and K. Choi, “New address generation scheme for memory-based FFT processor using multiple radix-2 butterflies,” in *Proc. ISOCC 2008*, Busan, South Korea, pp. 273-276, Nov. 2008.

[81] C. Li, J. Hu, "A security-enhanced alignment-free fuzzy vault-based

fingerprint cryptosystem using pair-polar minutiae structures", *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 3, pp. 543-555, Mar. 2016.

[82] G. Revadigar, C. Javali, W. Xu, A. V. Vasilakos, W. Hu, S. Jha, "Accelerometer and fuzzy vault-based secure group key generation and sharing protocol for smart wearables", *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 10, pp. 2467-2482, Oct. 2017.

[83] T. K. Dang, M. T. Nguyen, Q. H. Truong, "Chaff point generation mechanism for improving fuzzy vault security", *IET Biometrics*, vol. 5, no. 2, pp. 147-153, Jun. 2016.

[84] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proc. 3rd ACM Cloud Computing Security Workshop*, Chicago, Illinois, USA, pp. 113-124, Oct. 2011.

[85] J. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," *J. Biomed. Informatics*, vol. 50, pp. 234-243, Aug. 2014.

[86] J. H. Cheon, J. Jeong, J. Lee, and K. Lee, "Privacy-preserving computations of predictive medical models with minimax approximation and non-adjacent form," in *Proc. FC 2017*, Sliema, Malta, pp. 53-74, Apr. 2017.

[87] A. Khedr et al., "SecureMed: Secure Medical Computation using GPU Accelerated Homomorphic Encryption Scheme," *IEEE J. Biomedical Health Informatics*, vol. 22, no. 2, pp. 597-606, Mar. 2018.

[88] A. Juels et al., "A fuzzy vault scheme", *Designs Codes and Cryptography*, vol. 38, no. 2, pp. 237-257, Feb. 2006.

[89] C. K. Caldwell, "The Prime Pages (prime number research, records and resources)," *The Prime Pages (prime number research, records and resources)*. [Online]. Available: <https://primes.utm.edu/>.

- [90] T. Elomaa. MAT-72306. Class Lecture, Topic: "Chernoff bounds". Department of Mathematics, Tampere University of Technology, Tampere, Finland, Feb. 2, 2017.
- [91] I. Tham, "Singapore's privacy watchdog fines IHiS \$750,000 and SingHealth \$250,000 for data breach," *The Straits Times*, 15-Jan-2019. [Online]. Available: <https://www.straitstimes.com/singapore/singapores-privacy-watchdog-fines-ihis-750000-singhealth-250000-for-data-breach>.
- [92] D. Cox, "Regression Models and Life-Tables", *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 34, no. 2, pp. 187-202, 1972.
- [93] C. Boyd, M. Tolson and W. Copes, "Evaluating Trauma Care", *The Journal of Trauma: Injury, Infection, and Critical Care*, vol. 27, no. 4, pp. 370-378, 1987.
- [94] A. Abadi, P. Yavari, M. Dehghani-Arani, et al, "Cox models survival analysis based on breast cancer treatments," *Iranian Journal of Cancer Prevention*, vol. 7, no. 3, pp. 124-129, 2014.
- [95] S. Biondo et al., "Prognostic factors for mortality in left colonic peritonitis: a new scoring system", *Journal of the American College of Surgeons*, vol. 191, no. 6, pp. 635-642, 2000.
- [96] M. Kologlu, D. Elker, H. Altun, and I. Sayek, "Validation of MPI and PIA II in two different groups of patients with secondary peritonitis", *Hepato-gastroenterology*, vol. 48, no. 37, pp. 147–151, 2000.
- [97] K. Tolosie and M. Sharma, "Application of Cox Proportional Hazards Model in Case of Tuberculosis Patients in Selected Addis Ababa Health Centres, Ethiopia", *Tuberculosis Research and Treatment*, vol. 2014, pp. 1-11, 2014.
- [98] R. D'Agostino et al., "General Cardiovascular Risk Profile for Use in Primary Care", *Circulation*, vol. 117, no. 6, pp. 743-753, 2008.

[99] B. Tabaei and W. Herman, "A Multivariate Logistic Regression Equation to Screen for Diabetes: Development and validation", *Diabetes Care*, vol. 25, no. 11, pp. 1999-2003, 2002.

[100] "NTL: A Library for doing Number Theory", *Shoup.net*. [Online]. Available: <https://www.shoup.net/ntl/>.

[101] "The GNU MP Bignum Library", *Gmpilib.org*. [Online]. Available: <https://gmplib.org>.

[102] "Processor Benchmarks - Geekbench Browser", *Browser.geekbench.com*. [Online]. Available: <https://browser.geekbench.com/processor-benchmarks>.

[103] H. Chen, K. Laine, R. Player, and Y. Xia, "High-precision arithmetic in homomorphic encryption", in *Proc. CT-RSA 2018*, San Francisco, USA, pp. 116-136, April 2018.