Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

TOWARDS PRACTICAL HOMOMORPHIC ENCRYPTION AND EFFICIENT IMPLEMENTATION

by
Gyana R. Sahu

Cloud computing has gained significant traction over the past few years and its application continues to soar as evident from its rapid adoption in various industries. One of the major challenges involved in cloud computing services is the security of sensitive information as cloud servers have been often found to be vulnerable to snooping by malicious adversaries. Such data privacy concerns can be addressed to a greater extent by enforcing cryptographic measures. Fully homomorphic encryption (FHE), a special form of public key encryption has emerged as a primary tool in deploying such cryptographic security assurances without sacrificing many of the privileges of working with data in cleartext. In brief, a FHE scheme allows for computation of arbitrary functions on encrypted data stored on cloud and retrieve results in encrypted form.

In this dissertation, construction of various Proxy Re-encryption (PRE) schemes based on FHE schemes are presented and their parameter selection leading to secure instantiation discussed. PRE is a valuable cryptographic primitive that enables users to exchange information in an untrusted environment via a proxy.

In the second line of work, bootstrapping algorithms for FHE schemes and their efficient implementation in **PALISADE** lattice cryptography software library is presented. Originally proposed by Gentry, bootstrapping plays a central role in extending a somewhat homomorphic encryption (SHE) scheme towards full homomorphism. Several novel techniques to extend bootstrapping algorithms for larger secret key bounds, plaintext modulus and other FHE parameters are discussed.

Despite several advances and nearly a decade of research, efficiency of FHE schemes still remains one of the primary concern in deploying them in real-life applications. These concerns are addressed in the last line of work by demonstrating practical implementation of PRE schemes and bootstrapping algorithms on various heterogeneous GPGPU computing platforms. Since FHE schemes fall into the category of "embarrassingly parallel" computing workloads, the massive computing power of GPUs consisting of multiple processors can be leveraged to result in multiple order of improvement in performance. To aid this effort, parallel NTT algorithms are designed and various other optimizations suitable for GPU architectures discussed.

TOWARDS PRACTICAL HOMOMORPHIC ENCRYPTION AND EFFICIENT IMPLEMENTATION

by
Gyana R. Sahu

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

August 2020

Copyright © 2020 by Gyana R. Sahu

ALL RIGHTS RESERVED

APPROVAL PAGE

TOWARDS PRACTICAL HOMOMORPHIC ENCRYPTION AND EFFICIENT IMPLEMENTATION

Gyana R. Sahu

Dr. Kurt Rohloff, Dissertation Advisor — Date
Associate Professor of Computer Science, NJIT

Dr. Yuriy Polyakov, Committee Member — Date
Associate Research Professor of Computer Science, NJIT

Dr. Qiang Tang, Committee Member — Date
Assistant Professor of Computer Science, NJIT

Dr. Abdallah Khreishah, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Giovanni Di Crescenzo, Committee Member — Date
Senior Research Scientist, Perspecta Labs, Basking Ridge, NJ

BIOGRAPHICAL SKETCH

Author: Gyana R. Sahu

Degree: Doctor of Philosophy

Date: August 2020

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2020
- Master of Science in Computer Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2015
- Bachelor of Technology in Electrical Engineering,
  National Institute of Technology, Rourkela, 2010

Major: Computer Science

Presentations and Publications:

Polyakov, Y., Rohloff, K., Sahu, G., and Vaikuntanathan, V. (2017)., "Fast Proxy Re-encryption for Publish/Subscribe Systems,"*ACM Transactions on Privacy and Security (TOPS)*, 20(4), 1-31.

Sahu, G., Rohloff, K., "Construction and Evaluation of Proxy Re-Encryption on GSW FHE Scheme and other Primitives," Submitted to *IEEE Transactions on Dependable and Secure Computing.*

Sahu, G., Rohloff, K., "Accelerating Lattice based Proxy Re-Encryption Schemes on GPUs," Submitted to *International Conference on Cryptology and Network Security, 2020.*

Sahu, G., Rohloff, K., "Efficient and Scalable Bootstrapping of BV-LWE FHE Scheme," Submitted to *International Conference on Availability, Reliability and Security, 2020.*

This dissertation is dedicated to my beloved parents, Anupama and Kartikeswar Sahu for their endless love, support and encouragement on this rollercoaster of a journey.

To my late grandparents for instilling the value of education and hard work in our family. Although they are no longer in this world, their memories continue to regulate my life.

ACKNOWLEDGMENT

Undertaking this PhD has been a truly life-changing experience for me and it would not have been possible to do without the support and guidance that I received from many people.

First of all, I want to thank my advisor, Professor Kurt Rohloff, for introducing me to the exciting topic of FHE and most importantly, believing in me throughout these years. Without his constant support and mentoring, this dissertation would never be finished.

I would also like to thank Professor Yuriy Polyakov who has been a second mentor to me and guided me in several aspects of my research. I am always surprised by his intelligence and energy level and only wish I had more of his qualities. I am really lucky to have the privilege to work with him.

I would like to thank late Professor Gerard Ryan who was also a fellow PhD candidate of our research group. He always made himself available in helping me out in software trouble shooting. On a serious note, your witty one liners and jokes will always be missed.

I would also like to thank my dissertation committee members, Professor Qiang Tang, Professor Abdallah Khreishah and Dr Giovanni Crescenzo for spending their valuable time to read and improve my dissertation. I am grateful for their remarks and suggestions.

I am also grateful to my friends and fellow research group members for maintaining a friendly and cheerful atmosphere. In particular, I would like to thank Hadi, Nishanth and Hammad for helping me out in preparing for qualifiers and other helpful discussions.

Many thanks to multiple funding agencies in sponsoring my research. My research was partially sponsored by the Defense Advanced Research Projects Agency

(DARPA) and the Army Research Laboratory (ARL) under Contract Numbers W911NF-15-C-0226, W911NF-15-C-0233, and W911NF-15-C-0236. Project also partially sponsored by the National Security Agency under Grant H98230- 15-1-0274.

Last, but not the least, I want to thank my family and friends. Thank you for your love, support and relentless efforts to understand my passion for research. Your prayers for me have sustained me this far and I only wish it will continue to do so in the future.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF TABLES
(Continued)

Table **Page**

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The field of cryptology revolves around the study of techniques for preserving the privacy of sensitive information in the presence of adversaries. The early days of cryptology primarily focused on confidentiality of information by converting a message (or plaintext) into an "unintelligible" form by the process of encryption and converting them back by decryption. While this was sufficient for establishing secure communication links, researchers soon realized the need for other advanced cryptographic primitives which could enable computation of functions on confidential inputs between mutually distrusting parties while ensuring correctness of the outputs. Among the many possible solutions to this problem, fully homomorphic encryption is probably the simplest (conceptually) which can be applied to a wide variety of applications.

The notion of fully homomorphic encryption (FHE) was introduced by Rivest *et al.* [RAD$^+$78] in 1978 under the name "privacy homomorphism". It was postulated that by using a special form of encryption one can outsource computations while keeping the data secure under the security constraints governed by the underlying encryption scheme. As an example, the authors considered a loan company interested in storing encrypted records with a commercial time-sharing service. When required to fetch some meaningful information the time-sharing system, using the properties of privacy homomorphism, computes on encrypted data stored on private data banks and sends back ciphertext which can only be decrypted by the loan company. This flexible paradigm of computation restricts the time-sharing service from accessing sensitive information while retaining the capability to extract desired information via homomorphic computations. To support their hypothesis the authors provided examples of privacy homomorphisms exhibited by basic RSA encryption scheme with the caveat that instantiations of such scheme would be cryptographically weak and can be easily broken by chosen plaintext attack. Since construction of such an encryption scheme was not known at the time, its existence was left as an open problem.

In the following years, multiple encryption schemes have been presented which were endowed with homomorphic properties to some extent or in other words were *partially* homomorphic in nature however, achieving full homomorphism seemed to be a much difficult problem. To understand the significance of homomorphisms in cryptosystems and the role they play in cryptographic applications, we need to look into the abstract definition of homomorphic encryption. More formally, homomorphic encryption is a class of encryption scheme that allows elementary operations of addition and multiplication on encrypted messages without the necessity of decryption. Given such an encryption scheme (Π_{HE}) with encryption represented as a function $E\left(\cdot\right)$ then, using the homomorphic properties we can produce addition and multiplication as $f_{add}\left(E\left(m_0\right), E\left(m_1\right)\right) \mapsto E\left(m_0 + m_1\right)$ and $f_{mult}\left(E\left(m_0\right), E\left(m_1\right)\right) \mapsto E\left(m_0 * m_1\right)$, respectively. Now, depending on availability and limitation on the number of application of the operator, the encryption scheme (Π_{HE}) can be classified as: *partially*, somewhat (SHE) or fully homomorphic encryption (FHE) scheme.

Since the early days of public key cryptography, most of the encryption schemes were *partially* homomorphic in nature as they only allowed homomorphic multiplication or only homomorphic addition and further in some cases, only a limited number of homomorphic operations could potentially be applied. For example, the well known (unpadded) RSA cryptosystem is partially homomorphic in nature as they only support homomorphic multiplication of ciphertexts. ElGamal [ElG85] encryption scheme is another cryptosystem which supports such homomorphic multiplications. The first semantically secure additive homomorphic encryption scheme was proposed by Goldwasser and Micali [GM82] which enables addition over $\mathbb{Z}_2$ or *XOR* operations.

Another example of additive homomorphic encryption scheme is the well known Paillier cryptosystem [Pai99] which supports both addition of ciphertexts and scalar multiplication on ciphertexts. It can be noted that even in the absence of multiplication operator many privacy preserving applications have been developed on Paillier cryptosystem such as biometrics with fingerprints and face recognition [EHKM11, BBC$^+$10, EFG$^+$09, SSW09], medical data processing [YBG$^+$15], and different types of user matching applications [ZZZ$^+$13, GR15]. The first scheme to support both homomorphic addition and multiplication is the Boneh-Goh-Nissim cryptosystem [BGN05] which allows an arbitrary number of additions along with a single multiply without growing the ciphertext size.

A major limitation of these *partially* homomorphic encryption scheme is their inability to take part in higher order polynomial functions. In some trivial cases (*e.g.,* computation of squared Euclidean distances using Paillier encryption scheme) *partially* homomorphic encryption scheme can still be used at the expense of pre-computing certain inputs however, such solutions require greater coordination between the data providers and further efficiency can easily be degraded because of bottlenecks in network communication. A better and more flexible choice in such scenario is a somewhat homomorphic encryption scheme. In such encryption schemes, a finite number of addition and multiplication operators can be applied on encrypted inputs. A class of schemes known as "Polly Cracker" [LdVMPT09, AFFP11] introduced in the early 1990s were the first to be considered as somewhat homomorphic encryption scheme. These schemes are based on the hardness of computing remainders modulo an ideal over multivariate polynomial rings and use Gröbner bases as trapdoor information. However, almost all of these schemes have been broken and establishing security of these encryption schemes is a long-standing open research problem. Furthermore, Polly Cracker encryption schemes suffered from the problem of ciphertext expansion since homomorphic operations led to exponential growth in ciphertext size.

In his breakthrough seminal work, Gentry [G$^+$09] presented the first plausible construction of a fully homomorphic encryption (FHE) scheme based on the ideal coset problem over ideal lattices that supports the evaluation of an arbitrary number of both additions and multiplications. Gentry's construction initially begins with a somewhat homomorphic encryption (SHE) scheme that works with a limited number of homomorphic operations or in other words the scheme is capable of evaluating low degree polynomials homomorphically. To achieve full homomorphism, Gentry's blueprint introduced an ingenious step of bootstrapping which entails running the decryption circuit on ciphertext homomorphically using encrypted secret keys. In the following section, we give an outline on the first Fully homomorphic encryption scheme (FHE) introduced by Gentry.

1.1 Gentry's FHE Scheme

Gentry's construction of FHE scheme starts off with the key realization that the scheme should have a decryption algorithm of low circuit complexity and further the decryption algorithm should have circuit complexity in NC (problems that can be efficiently solved in poly-logarithmic time on a parallel computer). The reason behind this realization is that for a scheme to be bootstrappable homomorphic capacity of the scheme should be higher than the depth of decryption circuit or in other words the decryption algorithm should have a shallow depth. This immediately rules out approaches taken by other traditional encryption schemes based on modular exponentiation or pairing. In particular, the underlying mathematical object chosen to create a SHE scheme are ideal lattices as they inherently possess additive and multiplicative homomorphism properties. An ideal I is simply a subset of ring R (algebraic objects closed under addition and multiplication). It is suffixed with the term "lattices" to emphasize the fact that security of the SHE scheme is based on hard problems over lattices. Concretely, in his construction Gentry used the polynomial quotient ring $R = \mathbb{Z}[x]/\left(f\left(x\right)\right)$ as ideal lattices where $f\left(x\right)$ is a monic polynomial of degree n.

Gentry's scheme uses "good" basis B_J^{sk} as secret keys which are relatively short and nearly orthogonal vectors. Public key consists of some "bad" basis B_J^{pk} of an ideal lattice J along with a fixed basis B_I of an ideal lattice I such that I and J are relatively prime. To encrypt a message m in plaintext space $\mathcal{P} = \{0, 1\}$, the encrypter first adds a random ideal $i \in I$ and sends $(m + i) \bmod B_J^{pk}$ as ciphertext. A correctly formed ciphertext of the scheme has the form $c = m + i + j$ for $i \in I$ and $j \in J$. Since ciphertext c is an ideal lattice represented as a residue polynomial, coefficients of c can alternatively be represented as a coefficient vector $(c_0, \cdots, c_{n-1})$. The polynomial term $m + i$ is considered as a "noise" parameter and generated from a narrow distribution $\mathcal{D}$ such that the ciphertext c belongs to the ring R $(I + J)$ with respect to B_J^{sk} and not to individual ideals I and J. Furthermore, after the modulo reduction with the public basis B_J^{pk} this polynomial term $m + i$ is indistinguishable from random uniform distribution as per the ideal coset problem (ICP). For decryption, one simply removes the ideal j with the help of secret key B_J^{sk}. In the concrete instantiation, the secret key is again a polynomial $v_J^{sk} \in J^{-1}$ and decryption is shown as $m' = c - \lfloor v_J^{sk} \times c \rceil \bmod B_I$.

Homomorphic addition and multiplication defined on ciphertexts work correctly as they do not change the essential form of ideals. To understand this further, we fix the ideal lattice as even $(I = 2)$ and now, ciphertext can be shown in a simple form $c = m + 2i$. Homomorphic addition of two ciphertexts, $c_0 \left(= m_0 + 2i_0\right)$ and $c_1 \left(= m_1 + 2i_1\right)$ in this scheme is quite intuitive as the resultant ciphertext is in the form $c_{add} = m_0 + m_1 + 2\left(i_0 + i_1\right)$. Similarly, homomorphic multiplication of the two ciphertexts results in a polynomial $c_{mult} = m_0 \cdot m_1 + 2\left(i_0 m_1 + i_1 m_0 + 2 i_0 i_1\right)$. We can observe that the error term increases by small amounts in the case of homomorphic addition however, for homomorphic multiplication the error increases quadratically. Decryption of ciphertext still recovers the plaintext successfully as long as this error is within the parallelepiped formed by the basis of secret key, *i.e.,* $\mathcal{P}\left(B_J^{sk}\right)$. In the absence of a bootstrapping procedure, this implies that a fixed depth function (dominated by number of multiplications) can only be evaluated with this scheme so that the decryption error always lies within some predetermined radius.

In a clever move, Gentry then transforms this scheme with limited homomorphism into a fully homomorphic scheme by introducing a bootstrapping procedure. However, running a bootstrapping procedure or homomorphic decryption of a ciphertext in the initial scheme is not possible because of the requirement of a large circuit depth. This problem specifically arises due to multiplication and rounding of two k bit integers which take atleast $\mathcal{O}(\log k)$ depth of computation. To bring down the decryption complexity under the homomorphic capacity of the initial scheme, Gentry introduced a squashing step of the decryption circuit. In this step, he places a hint τ in the public key which consists of a set of vectors that has a (secret) sparse subset of vectors which adds up to the secret key, v_J^{sk}. Secret key in the sparse vector is considered to be secure under the computational hardness assumption of sparse subset sum problem (SSSP). To re-encrypt a ciphertext, first it is expanded using the vectors from the hint τ. Essentially, the expansion step pre-processes a ciphertext so that it can be decrypted by a shallower circuit. This expanded ciphertext is finally decrypted homomorphically in the squashed circuit simply by adding up appropriate indices of expanded ciphertext. This refreshing procedure is considered successful if the noise in the final ciphertext is lower than that in the initial ciphertext.

Implementations: Variants of Gentry's theoretical FHE scheme were implemented by Smart and Vercauteren [SV10] and Gentry and Halevi [GH11]. In [SV10] Smart and Vercauteren present a simplified version of Gentry's SHE scheme with greatly reduced key sizes and ciphertext sizes. In their construction of SHE scheme, they replace the ideal lattices with principal prime ideals or algebraic number fields. These prime ideals can still be represented as n dimensional $\mathbb{Z}$ basis and hence, they base the semantic security of their scheme on computational hardness of Polynomial Coset Problem (PCP), a problem very similar to Ideal Coset Problem (ICP). Most of the times these prime ideals are kept as pair of integers which allows to represent the ciphertext and keys in a compact form. Secret key of the scheme consists of an inverse of a small generator of the principal prime ideal while the public key is an unique root of an integer polynomial corresponding to the generator ideal. Encryption of an plaintext $\mathcal{P} \in \{0, 1\}$ is simply the evaluation of sum polynomial of a randomized low norm polynomial and binary encoded polynomial at the value given by public key. Further, homomorphic addition and multiplication of ciphertexts are associated with isomorphism properties of residue field resulting from factorization of prime ideal irreducible polynomial. As a result, homomorphic operations simply map to addition and multiplication of integers in finite field. Next, following Gentry's blueprint the authors describe a bootstrapping procedure on the squashed decryption circuit. However, their estimate of depth of squashed decryption circuit for some reported lattice dimension reveals that bootstrapabilty cannot be achieved because of insufficient homomorphic capacity.

One of the drawback of this SHE scheme is that the key generation algorithm is associated with a very large computational overhead because of the requirement of primality testing on a large number of candidates. Further, the lattice dimensions selected in their work estimates prime ideals with very high bit width modulus. Because of these two reasons, they were unable to generate keys in dimensions greater than 2048. Moreover, the impractical runtimes associated with key generation prevents their scheme to support bootstrapping as their estimate suggests to use lattices of dimension at least $n = 2^{27}$.

In a followup work [GH11], Gentry and Halevi describe a more practical variant of Gentry's fully homomorphic encryption scheme. A number of optimization steps were proposed in their work to reduce the key-generation complexity thereby, making it possible for the scheme to implement bootstrapping functionality. Similar to the Smart and Vercauteren [SV10] approach, construction of this scheme retains the usage of principal ideal lattices in the ring of polynomials, however they remove the requirement of a prime determinant lattice. The key generation phase consists of the following steps:

KeyGen: Sample a random vector $\vec{v}$ of dimension n associated with a rotational basis V (rows of the basis are negacyclic rotations of $\vec{v}$). The vector $\vec{v}$ is implicitly associated with the polynomial $v(x) \bmod f_n(x)$ where $f_n(x)$ is a monic polynomial of the form $x^{2^k}+1$. To compute the secret key a scaled inverse of $v(x) \bmod f_n(x) = w(x)$ is computed such that $w(x) \times v(x) = d$ where d is the determinant of the lattice $\mathcal{L}(V)$. Although the polynomial $w(x)$ can be found by extended Euclidean-GCD algorithm, the authors describe a more efficient procedure for computing the inverse polynomial via FFTs with $\mathcal{O}(n \log n)$ computational complexity. The secret key is considered to be correctly formed if the Hermite normal form of V satisfies a special form, namely all except the leftmost column equal to the identity matrix. This further implies that the lattice $\mathcal{L}(V)$ contains a vector of the form $\langle -r, 1, 0, \cdots, 0\rangle$. Secret key is formed by the pair $(\vec{v}, \vec{w})$ however, it suffices to store only a single odd coefficient of $\vec{w}$. Similarly, the public is composed of the Hermite normal form V but only a pair of integers d, r are kept as its representative.

Encryption: To encrypt a single bit $b \in \{0, 1\}$, we choose a random binary noise vector $\vec{u}$ and set $\vec{a} = 2\vec{u} + b \cdot \vec{e}_1$. The ciphertext is then formed by the evaluation of polynomial $a(x)$ at the point r. Since evaluation of polynomial is expensive, the authors discuss a batching procedure where k plaintexts can be encrypted simultaneously in time $\mathcal{O}\left(\sqrt{kn}\right)$.

Decryption: To decrypt a ciphertext $c \in \mathbb{Z}_d$ we compute the plaintext bit, $b = [c \cdot w_i]_d \bmod 2$.

Recryption: Recryption procedure refreshes a “dirty” ciphertext by reducing the expanded noise incurred due to homomorphic computations. This procedure stems from Gentry’s proposed squashing technique *i.e.,* to modify the decryption circuit of the SHE scheme so that bootstrapping can be performed with a shallower depth. To facilitate bootstrapping, a hint is added to the public key which contains an instance of the sparse subset sum problem. This set consists of elements $\{x_i \in \mathbb{Z}_d : i = 1, 2, \cdots, S\}$ such that there exists a very sparse subset of x_i’s that sums up to the secret key w. The characteristic vector associated with the sparse subset is a bit vector $\vec{\sigma} = \langle \sigma_1, \cdots, \sigma_S \rangle$ such that $\sum_i \sigma_i x_i = w \bmod d$. To begin the recryption procedure, a ciphertext $c \in \mathbb{Z}_d$ is first combined with the x_i’s to generate y_i’s such that $y_i = \langle c x_i \rangle$. Next, the homomorphic decryption is executed as per the modified equation:

$$D_{c,d} = \left[\sum_{i=1}^{S} \sigma_i y_i \right]_d \bmod 2 = \left(\sum_{i=1}^{S} \sigma_i y_i \right) - d \cdot \left\lceil \sum_{i=1}^{S} \sigma_i \frac{y_i}{d} \right\rfloor$$

Implementation of the FHE scheme was carried out on a server grade platform with lattice dimensions ranging from $n = 512$ to $n = 32768$. Since the parameter selected for these dimensions estimated integers with bit size much larger than 64-bits, the implementation relied on GNU GMP library and Shoup’s NTL library. Public key size for the select parameters and lattice dimensions is quite large on account of large subset sum bootstrapping keys ranging from 70 MBs to 2.3 GBs. Further, performance of the bootstrapping or recryption procedure is found to be quite unsatisfactory with runtimes ranging from 30 second for small dimension to 30 minutes for large dimensions.

1.2 Early Attempts and Evolution of FHE Scheme

Gentry's plausible construction of FHE scheme and it's concrete implementations galvanized the researchers of cryptology community to search for other efficient constructions. Over the past decade a number of FHE schemes have been proposed and considerable work has been done to make them practical. Starting from Gentry's work these FHE schemes are sometimes classified into three generations of literature. The first generation comprises of FHE schemes that directly evolved from Gentry's theoretical construction of FHE scheme and ideal lattices. These schemes include the already discussed Smart-Vercauteren implementation, Gentry-Halevi implementation and the integer arithmetic based simpler FHE scheme of van Dijk *et al.* [DGHV10]. FHE schemes of first generation were found to be rather impractical for implementation owing to their larger bit lengths and rapidly growing noise. Nonetheless, these schemes gave an insightful understanding into concrete construction of FHE schemes and paved the way for creation of more efficient constructions and optimizations. The second generation of FHE schemes [LTV13, BLLN13, BV11b, BV11a, BGV14] were based on improved algebraic structures and stronger hardness assumptions which translated into better noise controlling techniques and higher efficiency. A common feature of these schemes is that they deviated from Gentry's squashing procedure and eliminated the sparse subset sum assumption. The third generation is mostly attributed to GSW [GSW13] FHE scheme and it's RLWE variant Ring-GSW FHE scheme [KGV16] which further improved the noise growth nature by reducing it to a asymmetric nature. We further augment the third generation with a number of subsequent LWE based FHE schemes [DM15, GINX16, CGGI16] which leveraged the asymmetric noise growth property of GSW FHE scheme to enable bootstrapping procedure in a pragmatic manner. In the next couple of subsections we present a brief overview of FHE schemes of these three generations.

1.2.1 The First Generation of FHE Schemes

In [DGHV10], van Dijk *et al.* presented a FHE scheme over integers using elementary modular arithmetic. The scheme was conceptually simpler, because it operated on integers instead of ideal lattices over polynomial rings. It was shown that security of the scheme can be reduced to hardness assumption of approximate-gcd problem, a problem introduced by Howgrave-Graham [HG01]. Informally, an instance of this problem consists of t random large integers $\{x_1, \cdots, x_t\}$ where $x_i = pq_i + r_i$ and recovering the common multiple p is considered to be a hard problem. The scheme was presented with a message space $\mathcal{P} = \{0, 1\}$ but can be easily extended to support a larger message space. For a security parameter λ, the encryption scheme is parameterized by integers $\eta(\lambda), \gamma(\lambda), \rho(\lambda), \tau(\lambda)$. The scheme is then composed of the following algorithms:

KeyGen: An odd integer, p of η-bit is generated and set as the secret key. Public key is generated by setting $x_i \leftarrow_{\$} pq_i + r_i$ where q_i and r_i are sampled as $q_i \leftarrow_{\$} \mathbb{Z} \cap [0, 2^{\gamma}/p)$ and $r_i \leftarrow_{\$} \mathbb{Z} \cap (-2^{\rho}, 2^{\rho})$. Public key, pk consists of the set of integers $\langle x_0, x_1, \cdots, x_{\tau} \rangle$.

Encrypt: To encrypt a bit $b \in \{0, 1\}$, we choose a random subset $S \subseteq \{1, 2, \cdots, \tau\}$ and a random integer r and set the encryption as $c \leftarrow \left[b + 2r + 2\sum_{i \in S} x_i\right]$.

Decrypt: Decryptpion results in bit $b' = (c \bmod p) \bmod 2 = (c \bmod 2) \oplus (\lfloor c/p \rceil \bmod 2)$.

Homomorphic addition and multiplication of ciphertexts simply map to corresponding integer addition and multiplication in modulo-x_0. It was remarked that multiplication of ciphertexts roughly doubles the bit length of noise and may result in decryption error even after a single evaluation. To remedy this problem, an alternative procedure was explained where noise grows in small amounts using a sequence of modular reductions. Finally, following Gentry's squashing strategy the public key is expanded with subset sum secret key to reduce the decryption circuit depth. It was shown theoretically that this step allows to “post-process” or decrypt ciphertext homomorphically thus achieving bootstrapability.

1.2.2 The Second Generation of FHE Schemes

FHE schemes of the second generation brought in some radical changes and introduced many optimization techniques. These schemes were based on hardness assumption of well established lattice problem, LWE [Reg09] and it's variant RLWE [LPR10]. In addition, the second generation also comprises of FHE schemes such as LTV [LTV13] and YASHE [BLLN13] which were based on the NTRU problem [HPS98] and arithmetic of cyclotomic polynomials. In the next few subsections, we present a brief overview of FHE schemes based on NTRU problem and LWE problem.

NTRU FHE Schemes NTRU public key cryptosystems came into existence with the work of Hoffstein, Pipher and Silverman [HPS98]. The main advantages of NTRU cryptosystem over other classical encryption schemes were it's moderate key sizes, excellent asymptotic performance owing to fast FFT based polynomial operations in $\mathbb{Z}_q[X]/\left(X^n - 1\right)$ and conjectured resistance to quantum computers. Stehlé and Steinfeld [SS11] later showed a modified NTRU encryption scheme that is considered to be IND-CPA secure. Specifically, they replaced the polynomial ring with $R = \mathbb{Z}_q[X]/\left(X^{2^k} + 1\right)$ and suggested the secret key to be sampled from a discrete Gaussian with very large standard deviation ($\approx q^{1/2}$) so that public key distribution is statistically close to uniform. However, such modification leaves their scheme with little or no homomorphic capabilities. López, Tromer and Vaikuntanathan [LTV13] showed a transformation of Stehlé and Steinfeld NTRU scheme to obtain a fully homomorphic encryption scheme by basing security on decisional small polynomial ratio (DSPR) assumption in addition to the RLWE security assumption. Another important property of this scheme is that the LTV FHE scheme can evaluate homomorphic functions on ciphertexts encrypted under a number of different and independently generated keys making the scheme mutikey fully homomorphic. Other notable optimizations shown in this work were the application of modulus switching and key-switching techniques borrowed from previous literatures [BGV14, BV11b]. In a subsequent work, [BLLN13] Bos *et al.* presented another NTRU modified scheme dubbed as YASHE where the authors removed the non-standard DSPR security assumption by adopting polynomial tensoring techniques. The polynomial tensoring technique, first introduced in the work of Brakerski [Bra12], was used to construct a scale invariant FHE scheme similar to BGV FHE scheme. In brief, scale variance is a sort of noise management technique which allows to keep a single modulus as opposed to a ladder of moduli found in the parameters of BGV [BGV14] FHE scheme. We briefly summarize LTV [LTV13] and YASHE [BLLN13] FHE schemes as follows:

LTV FHE scheme The scheme differs from the original NTRU encryption scheme [HPS98] in the usage of polynomial rings of the form $R = \mathbb{Z}[X]/\langle X^n + 1\rangle$, where n is a power of two. The scheme is parameterized by a security factor λ as follows:

- lattice dimension, $n = n(\lambda)$.
- plaintext modulus p.
- a prime ciphertext modulus $q = q(\lambda)$ such that p, q are relatively prime.
- a B-bounded distribution χ over R, $B \ll q$.

LTV encryption scheme comprises of the following operations:

KeyGen: We sample polynomials $f', g \leftarrow_{\$} \chi$ and set $f = pf' + 1$ so that $f \equiv 1 \bmod p$. Polynomial f' should be sampled in a way so that f^{-1} exists and if this is not the case then we re-sample f'. Finally we set secret key, sk and public key, pk as $pk = h = pgf^{-1} \in R_q$ and $sk = f \in R$.

Encrypt: Plaintext space of the scheme is $\mathcal{P} \in R_p$ which supports encryption of integers in $\mathbb{Z}_p$. To encrypt a message $m \in R_p$ we sample polynomials $s, e \leftarrow \chi$ and output the ciphertext $c = hs + pe + m \in R_q$.

Decrypt: To decrypt the ciphertext c with secret key f we compute m' as follows $m' = (fc \bmod q) \bmod p$.

Homomorphic evaluations of addition and multiplication directly map to respective operations in cyclotomic polynomial domain. Since multiplication of polynomials leads to quadratic growth in noise it is generally recommended to keep a ladder of moduli (of length equal to depth D) and perform modulus switching after each homomorphic evaluation or levels. Another problem associated with multiplication is that the resulting ciphertext can no longer be decrypted with the original secret key. This can be seen from the following equation.

$$\langle c_1, s\rangle \cdot \langle c_2, s\rangle = \langle c_1 \otimes c_2, s \otimes s\rangle = \langle c_{mult}, s \otimes s\rangle$$

From the above equation, it is the clear that the ciphertext resulting from homomorphic multiplication can only be decrypted with the tensored secret key. To normalize the ciphertext so that it can be decrypted with the original secret key an invocation of key switching procedure is required. Key switching procedure is aided by augmenting the public key with an evaluation key generated from original secret key and tensored secret key. Further, in a leveled LTV FHE scheme these key switching hints should be generated for each level separately.

YASHE FHE Scheme Construction of YASHE [BLLN13] FHE scheme is very similar to LTV scheme but mainly differs by the usage of a wide discrete Gaussian χ_{key} to avoid DSPR assumption and polynomial tensoring technique for homomorphic multiplication. In the following, we present a brief overview of the construction of YASHE scheme.

The scheme operates on a cyclotomic ring $R = \mathbb{Z}[X]/\Phi_m(X)$, ciphertext modulus q and plaintext modulus p with $1 < p < q$. The scheme use two different discrete Gaussians χ_{key} and χ_{err} defined on ring R to generate keys and errors respectively.

KeyGen: Key generation is very similar to the LTV *KeyGen* procedure and proceeds as follows: We sample f', g as $f', g \leftarrow_{\$} \chi_{key}$ and set $f = pf' + 1$. Polynomial f should be invertible or else we resample polynomial f'. We compute $f^{-1} \in R_q$ and set $h = pgf^{-1}$. Set public key $pk = h$ and secret key $sk = f$.

Encrypt: Encryption is a RLWE adaptation of Regev's encryption scheme as shown in [Bra12, FV12]. To encrypt a message $m \in R_p$ we sample polynomials $s, e \leftarrow_{\$} \chi_{err}$ and output the ciphertext c as follows:

$$c = [\lfloor q/p \rfloor (m + e + hs)]$$

Decrypt: To recover message m' we compute :

$$m' = \left[\left\lfloor \frac{p}{q} \cdot [fc]_q \right\rceil \right]_p$$

where $\lfloor x \rceil$ rounds x to the nearest integer.

Homomorphic addition of two ciphertexts c_1 and c_2 is given by $c_{add} = c_1 + c_2$. Homomorphic multiplication of ciphertexts is conducted with the help of evaluation key, evk which is generated as follows:

$$\text{Sample } e, s \leftarrow_{\$} \chi_{err}^{\ell_{w,q}^3}$$
$$\gamma = \left[f^{-1} P_{w,q} \left(D_{w,q}(f) \otimes D_{w,q}(f) \right) + e + h \cdot s \right]_q \in R^{\ell_{w,q}^3}, \text{ Set } evk = \gamma$$

Here, $P_{w,q}(\cdot)$ and $D_{w,q}(\cdot)$ are the generalized base-w PowersOfTwo and bit decomposition functions respectively. More in detail, for a polynomial $x \in R_q$ base-w bit decomposition outputs a vector $\langle x_0, x_1, \cdots, x_{\ell_{w,q}-1} \rangle$ whereas PowersOfTwo on the same polynomial returns a vector $\langle xw^0, xw^1, \cdots, x^{\ell_{w,q}-1} \rangle$ such that $\langle D_{w,q}(x), P_{w,q}(y) \rangle =$

$xy \bmod q$ and $\ell_{w,q} = \lfloor \log_w q \rfloor + 2$. Using these relinearization operators homomorphic multiplication of ciphertexts c_1 and c_2 can be shown as follows:

$$c_{mult} = \left[\left\lfloor \frac{t}{q} P_{w,q}(c_1) \otimes P_{w,q}(c_2) \right\rceil \right]_q \in R^{\ell_{w,q}}$$

The final step is to remove the tensored secret key by performing key-switching on this intermediate ciphertext as follows:

$$c_{mult} = \text{KeySwitch}(c_{mult}, evk)$$

Indeed, the polynomial tensoring technique removes the additional DSPR assumption by slowing down the noise growth however, this modified scheme ceases to be multi-key homomorphic. Furthermore, the size of evaluation key is increased by a polynomial factor when compared with key sizes of LTV FHE scheme.

RLWE FHE schemes Cryptographic schemes based on NTRU assumption were considered to be provably secure until Cheon *et al.* [CJL16] and Albrecht *et al.* [ABD16] showed that NTRUEncrypt can be attacked by subfield lattice attacks which are exploited because of improper parameter set generation. Subfield lattice attack work by repeatedly reducing the size of the ring and solve the Shortest Vector Problem (SVP) for $n = 512$ and lower. It is further conjectured that the same subexponential attack can break the security of other NTRU FHE schemes such as LTV [LTV13] and YASHE [BLLN13] FHE schemes for weak choices of parameters. On the other hand FHE schemes which are purely based on LWE/RLWE security assumption are still considered to be secure and to the best of our knowledge no attack that breaks it in subexponential time has been reported in literature.

The first public key encryption scheme based on the learning with errors (LWE) assumption was introduced by Regev [Reg09]. It was shown from the worst-case to average-case reductions of Regev [Reg09] and Peikert [Pei09] that solving LWE problem is at least as hard as finding short vectors in any lattice. Soon after Gentry's breakthrough work on FHE, Brakerski and Vaikuntanathan in a series of work showed the construction of simpler FHE schemes [BV11a, BV11b] based on the LWE assumption and it's more efficient alternative, RLWE assumption. In these works, many optimization techniques, such as dimension reduction, modulus switching and most importantly relinearization operations, were discussed which allows succint representation of ciphertext with smaller noise while also maintaining the cryptographic strength. In another work, [BGV14] Brakerski *et al.* showed an improvement of the BV [BV11b] FHE scheme by introducing a noise controlling technique, namely, a leveled HE approach which allows to keep the noise level steady by shedding down bit length of modulus. Typically, in a leveled SHE scheme with a-priori bounded depth D instead of keeping a single ciphertext modulus q we store a ladder of gradually decreasing moduli q_i such that $q = \prod_{i=1}^{D+1} q_i$. After multiplication of two mod-q ciphertexts we switch to a smaller modulus by an application of modulus switching procedure and the resultant ciphertext c_{mult} is produced w.r.t $q = \prod_{i=1}^{D} q_i$, i.e., we drop the largest modulus. When compared with the conventional BV [BV11b] FHE scheme where the noise grows quadratically for each level (B^{2^D}, B- bounded noise) the leveled approach keeps the noise magnitude roughly the same ($\approx B$) assuming that we scale down by B after each homomorphic multiplication. In a related work, Brakerski [Bra12] presented a scale invariant FHE scheme which further removes the requirement of storing a ladder of moduli and hence, eliminates the need of modulus switching. It turns out that in this scheme evaluation of a D-depth function leads to a conservative noise growth of $B \cdot poly\,(n)^D$. We briefly summarize the construction of these LWE based FHE schemes in the next few subsections.

BV-LWE FHE scheme: In [BV11a], Brakerski and Vaikuntanathan presented a FHE scheme solely based on LWE assumption. The encryption scheme is roughly similar to Regev's [Reg09] scheme but uses additional tricks to obtain homomorphism properties. In the symmetric-key scheme, a secret key is simply a n-dimensional short vector $\vec{s} \in \mathbb{Z}_q^n$. To encrypt a bit $m \in \{0,1\}$, we choose a random uniform vector $\vec{a} \in \mathbb{Z}_q^n$ and a short error value $e \in \mathbb{Z}$ and set the encryption as $c = (\vec{a}, b = \langle \vec{a}, \vec{s} \rangle + 2e + m) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. Homomorphic addition of two ciphertexts c_1 and c_2 is given by $c_{add} = (\vec{a_1} + \vec{a_2}, b_1 + b_2)_q$. However, homomorphic multiplication leads to quadratic expression on the secret key , $\vec{s}$. At this stage, we can relinearize the quadratic expression by publishing roughly $\mathcal{O}(n^2)$ encryptions of individual terms of $\vec{s}$ vector. In summary, we can see that the [BV11a] scheme is a conceptually simple scheme but suffers from inefficient quadratic runtimes and larger memory overheads for storing the evaluation keys.

BGV FHE scheme: Similar to LTV FHE scheme, BGV [BV11b, BGV14] FHE scheme uses a polynomial ring of the form $R = \mathbb{Z}_q[X]/\langle \Phi_m(X) \rangle$ to define the cryptosystem. For the most efficient implementations, a cyclotomic polynomial of the form $\Phi_m(X) = X^{m/2} + 1$ is used where m is a power of two. However, arbitrary cyclotomic polynomials (m is arbitrary) have also been used to define batching or packing multiple messages in a ring polynomial. The basic parameters of the scheme are as follows:

- ring dimension, n.
- plaintext modulus p.
- ciphertext modulus $q = \prod_{i=1}^{L+1} q_i$.
- discrete Gaussian error distribution χ_{err} with bound B_{err}.

Message space for the scheme consists of $\mathcal{M} \in R_p$. The scheme consists of the following operations:

KeyGen: Sample polynomials $a \leftarrow_{\$} \mathcal{U}_{R_q}$ and $s, e \leftarrow_{\$} \chi_{err}$. Compute $b = a \cdot s + pe \in R_q$. Set the public key pk and private key sk:

$$sk = s \in R, \quad pk = (a, b) \in R_q^2$$

Encrypt: To encrypt a message $m \in R_p$ we sample polynomials $v, e_0, e_1 \leftarrow_{\$} \chi_{err}$. Compute the ciphertext $c = (c_0, c_1) \in R_q^2$:

$$c_0 = b \cdot v + pe_0 + m \in R_q, \; c_1 = a \cdot v + pe_1 \in R_q$$

Decrypt: Compute the ciphertext error $t = c_0 - s \cdot c_1 \in R_q$. Output $m' = t \, (\bmod p)$.

The scheme is correct as long as there is no wrap-around modulo q. To guarantee correct decryption of the ciphertext we should ensure $q > 6\sqrt{n}pB_{err}^2$. Homomorphic addition is performed by addition of corresponding polynomials in ciphertexts. Homomorphic multiplication operation is however more cumbersome and increases the number of ring elements in a ciphertext. Specifically, result of multiplication of two ciphertexts (c_0, c_1) and (c_0', c_1') is given by the ciphertext $c_{mult} = (c_{mult,0}, c_{mult,1}, c_{mult,2})$ where $c_{mult,0} = c_0 c_0', c_{mult,1} = c_0 c_1' + c_0' c_1, c_{mult,2} = c_1 c_1'$. This ciphertext can be decrypted by computing $(c_{mult,0} + c_{mult,1}s + c_{mult,2}s^2) \bmod p$. Alternatively, using relinearization we can reduce the ciphertext back to two ring elements. To aid relinearization, we publish evaluation keys as follows:

$$h_i = \left(a_i, b_i = -(a_i s + pe_i) + 2^i s^2\right) \text{ for } i = 0, \cdots, \lceil \log q \rceil - 1$$

where $a_i \leftarrow_{\$} R_q$ and $e_i \leftarrow_{\$} \chi_{err}$ are chosen independently for every i. Finally, relinearization proceeds as follows to transform a ciphertext $c_{mult} = (c_0, c_1, c_2)$ to

$c_{mult} = \left(c_0^{\text{relin}}, c_1^{\text{relin}}\right)$:

$$c_1^{\text{relin}} = c_1 + \sum_{i=0}^{\lceil \log q \rceil - 1} c_{2,i} a_i \ , \quad c_0^{\text{relin}} = c_0 + \sum_{i=0}^{\lceil \log q \rceil - 1} c_{2,i} b_i$$

where $c_{*,i}$ represents the i-th bit decomposition of polynomial c_*

1.2.3 The Third Generation of FHE Schemes

FHE scheme of third generation mainly refers to the work by Gentry *et al.* [GSW13] which describes another homomorphic encryption scheme based on the learning with errors (LWE) problem. The greatest benefit of GSW FHE scheme is that the noise in this scheme grows asymmetrically which removes the need for modulus-switching and makes bootstrapping optional. To explain this further, in previously mentioned LWE FHE schemes multiplication of ciphertexts with initial noise B expands quadratically to B^2. Evaluation of a D-depth circuit by these schemes is kept to a minimum B^{2^D} where homomorphic multiplication is performed in a binary tree representation to get a logarithmic depth. In contrast to this, noise growth in GSW FHE scheme is asymmetric and homomorphic multiplication is performed sequentially. More specifically, final noise level in the resultant ciphertext mainly depends on the noise magnitude of only the left multiplicand ciphertext. The other important feature of this scheme is that relinearization is performed implicitly and hence the need to generate evaluation keys is eliminated. Next, we present a brief overview of the construction of GSW FHE scheme.

The scheme is parameterized by a modulus q, lattice dimension n, $m = \mathcal{O}(n \log q)$, error distribution χ. We set $\ell = \lceil \log q \rceil$ and $N = (n+1)\,\ell$.

KeyGen: Secret key is generated by sampling a short vector $\vec{t}$ as $\vec{t} \leftarrow_{\$} \chi$ and set $sk = \vec{s} \leftarrow (1, -t_1, \cdots, -t_n) \in \mathbb{Z}_q^{n+1}$. Further, we set a vector $\vec{v} = \text{PowerOf2}(\vec{s})$. Public key is set by sampling a uniform random matrix $B \leftarrow_{\$} \mathbb{Z}_q^{m \times n}$ and an error vector $\vec{e} \leftarrow_{\$} \chi^m$. Public key is formed by concatenating the $\vec{b}$ vector and B matrix resulting in a $(m \times n+1)$ matrix A. Finally, we set $pk = A$. It can be observed that the m rows of the public key matrix B can be interpreted as m independent instances of the LWE problem such that $A \cdot \vec{s} = \vec{e}$.

Encrypt: Messages are restricted to bits, $\mathcal{M} \in \{0,1\}$. To encrypt a bit μ we sample a uniform binary matrix $R \leftarrow_{\$} \mathcal{U}_{\{0,1\}}^{N \times m}$ and output the ciphertext C given as :

$$C = \text{Flatten}\left(\mu \cdot I_N + \text{BitDecomp}\left(R \cdot A\right)\right) \in \mathbb{Z}_q^{N \times N}$$

Here, the flattening operation is simply a compounded bit decomposition operation which keeps the ciphertext bounded with low Euclidean norm. Flattening procedure on a vector $\vec{a}$ returns the expanded vector such that $\text{Flatten}(\vec{a}) = \text{BitDecomp}\left(\text{BitDecomp}^{-1}(\vec{a})\right)$.

Decrypt: Decryption recovers the ciphertext by using the approximate eigenvector property of the scheme. As per this property, a ciphertext C encrypts message μ if the following holds true, $C \cdot \vec{v} = \mu \cdot \vec{v} + \vec{e}$. It can be interpreted that the secret key $\vec{v}$ is an approximate eigenvector of the ciphertext matrix C and the message μ is the eigenvalue. Since the first ℓ coefficients of $\vec{v}$ are simply powers of 2, i.e., $(1, 2, \cdots, 2^{\ell-1})$ we can use any index i of the vector $\mu \cdot \vec{v}$ to recover the message. Specifically, to differentiate from the error we use a index i such that $v_i = 2^i \in (q/4, q/2]$. Finally, we compute $x_i \leftarrow \langle C_i, \vec{v} \rangle$ and set $\mu' = \lfloor x_i / v_i \rceil$.

Homomorphic operation are quite straightforward in this scheme as they simply map to addition and multiplication of ciphertexts represented in LWE matrix form. Further, these operations do not require any evaluation key as they implicitly store the gadget matrix G.

Because of the steady noise growth property, GSW FHE scheme has been shown to be very useful in executing the bootstrapping procedure of other FHE schemes. In these work [BV14b, ASP14], secret key of a particular SHE scheme is encrypted with GSW FHE scheme, decryption circuit is evaluated using the homomorphic operations of GSW FHE scheme and finally a ciphertext of the SHE scheme is extracted from the resultant GSW ciphertext. A major drawback of such GSW FHE scheme based bootstrapping is that efficiency is severely affected due the sub-cubic $\mathcal{O}\left(n^{2.3727-3}\right)$ runtime of computing matrix products needed for homomorphic multiplication. For practically feasible, bootstrapping using GSW scheme can be replaced with it's RLWE counterpart, Ring-GSW FHE scheme [KGV16]. Starting from the work of Ducas and Micciancio [DM15] many LWE FHE schemes [BR15, CGGI16, BDF18] have been shown which make a greater use of Ring-GSW FHE scheme to achieve bootstrapability. Next, we discuss the security aspects of lattice based cryptography and some of it's applications.

1.3 Resistance to Quantum Computer Attacks

Most of the primitives and encryption schemes in classical cryptography are based on the intractability of the integer factorization problem and discrete logarithm problem over a finite field. Examples of some of the well known schemes which base their security on these hard problems are RSA encryption scheme, Elliptic curve digital signature and encryption scheme etc.

In [Sho99], Peter Shor showed in his seminal work that a hypothetical quantum computer can find solution to discrete logarithm and integer factorization in polynomial time. This means if quantum computers some how come into existence then most of the classical crypto systems will be broken. Furthermore, the existence of Bitcoin and other crypto-currencies would suddenly come to a halt, causing wide spread panic in financial markets.

With the recent advances in the field of quantum physics backed by many reputable organizations and academia like Google, IBM, MIT etc, quantum computers have finally transitioned from theory to reality. With these advances, there is an immediate need to build efficient cryptographic schemes that can resist quantum computer attacks.

One such solution is offered by lattice based cryptography which base their security on the intractability of closest vector problem (CVP) and shortest vector problems (SVP). Ajtai [Ajt98] showed that SVP is NP-hard for randomized reduction of lattice basis. In another work, Micciancio [Mic01] proved that the approximate variant of SVP known as γ-SVP is NP-hard for randomized reduction if $\gamma < \sqrt{2}$. Furthermore, some of the efficient lattice reduction algorithms like LLL and block-KZ are known to solve approximate version of SVP and CVP in $2^{\mathcal{O}}(n)$ time complexity where n is the lattice dimension. These result prove that lattice based cryptographic schemes (based on hardness of ICP, LWE and RLWE problem) are indeed secure against quantum computer attacks.

1.4 Applications

Fully homomorphic encryption is considered as a generic computing tool that has numerous applications in the realm of secure computing. As envisoned by Rivest *et al.* [RAD+78], FHE has the potential to play a huge role in outsourcing data and enabling cloud services or applications on them. In other use cases, FHE can query private databases, aid multi-party computations, support Zero-Knowledge proofs and even execute other cryptographic schemes (such as AES, PRINCE etc). In the next few subsections, we describe some of these applications.

1.4.1 Server Aided Computations

In recent years, cloud computing has been flourishing and many tech giants and startups are offering commercial services for customers to store data and host applications. While the pervasive use of cloud computing has eased the burden on individual users and businesses to invest in expensive hardware, on the other hand it has also raised the question on data privacy. The level of trust a user places on a service provider varies from client to client and with the sensitivity of information. Furthermore, servers are always vulnerable to security breach where adversaries are constantly looking out to exploit any security flaws or improper firewall setup.

Fully homomorphic encryption presents an elegant solution to alleviate such concerns on data privacy. In such a paradigm, the users of the cloud server outsources the data in encrypted form. Once the data is present on the server, the user can then specify the computation to be carried out. Finally, the computed data can be retrieved from the server and decrypted at user end for further consumption. A major advantage of using FHE in this paradigm is that the user can be offline in the computation phase thus allowing low-power or resource constrained devices to save power.

Lauter *et al.* [NLV11] discussed some of the applications of homomorphic encryptions pertaining to medical, financial and advertising sectors. They described the computation of simple statistical functions such as mean, standard deviation and logistical regression on outsourced encrypted data. In another work by Wu and Haven [WH12], an implementation of large scale statistical analysis on encrypted data was presented. The crux of the work was to show that computation on large scale encrypted data can be made practical in reasonable amount of time by the application of ciphertext packing techniques [GHS12c]. Lastly, FHE can be used to run predictive models such as medical or genomics predictions, spam detection, face recognition, and financial predictions on user encrypted data with pre-trained machine learning classifiers. Many efficient frameworks [JVC18, GBDL$^+$16, BPTG15] have been shown which support such machine learning modeling on encrypted data.

1.4.2 Private Information Retrieval

A (single-database) private information retrieval (PIR) is a cryptographic scheme that enables the client to retrieve records stored on the server without the server learning which record was retrieved. Trivially, this can be achieved when the server sends the entire database to the user and hence, doesn't reveal the users access pattern. However, such solutions are not practical for any realistic use case because of large communication overheads. Implementing PIR protocols with single fetch query leads to more efficient and secure design as they rely on computational security assumptions. Among the many possible solutions, FHE is considered to be most practical in terms of communication efficiency. In this approach, using the multiplicative homomorphism property database indices are first matched with encrypted user query bits. Finally, a sum of all query results are added and returned as the desired ciphertext. Some of the notable FHE based PIRs were designed in [AMBFK16, ACLS18] which introduced several compression techniques to amortize the computational cost further.

1.4.3 Multiparty Computations

Secure multiparty computation has been one of the significant research areas in the field of cryptography. Initially proposed by Yao [Yao82, Yao86] in the two-party setting with honest-but-curious adversary model, the protocol was extended to multiple parties interested in computing a joint function. Secure Multi-Party Computation (MPC) protocols allows a group of mutually distrusting users to compute a function jointly on their inputs without revealing any information beyond the output. Many secure multi-party computation protocols have been proposed such as privacy preserving data mining, privacy preserving database query targeting specific application, however, most of them suffer from the bottleneck arising from communication complexity during protocol execution.

Alternatively, MPC protocols can be executed by a FHE scheme which supports evaluation of functions on ciphertexts encrypted under different keys. Such FHE schemes are commonly termed as multi-key FHE. Using server aided computation strategy, participant of the MPC pool their data on an untrusted server and let the server compute an arbitrary function without any participation from the users. The server and the set of participants then interact in a decryption phase and retrieve their corresponding outputs. López *et al.* [LTV13] introduced a multi-key FHE scheme based on NTRU and DSPR assumption. The LTV multi-key FHE scheme is dynamic in nature as it allows to compute arbitrary functions with an arbitrary choice of participants chosen on the fly. Clear and McGoldrick [CM15], in another work showed the construction of a multi-key FHE scheme based on a variant of GSW FHE scheme. The [CM15] scheme was further simplified by Mukherjee and Wichs [MW16] and extended to allow one round of distributed decryption of resultant multi-key ciphertext.

1.5 Our Contributions

In this dissertation, we demonstrate our results from three major areas related to FHE and it's application in secure computing. In the first set of work, we present construction of Proxy Re-Encryption (PRE) schemes from various homomorphic encryption schemes. Our unidirectional PRE schemes enable users to share and read encrypted data without any prior exchange of decryption keys. PRE is particularly helpful when the publisher and subscriber of messages are working in an untrusted environment such as cloud servers. Our next contribution lies in the construction of efficient bootstrapping techniques for BV-LWE [BV11a] and it's variant BV-GSW FHE scheme. Further, we present an extension of BV-LWE bootstrapping technique to accommodate larger ciphertext modulus by running the decryption circuit on multi-dimensional grids. Lastly, we tackle the efficiency aspects of our PRE schemes and bootstrapping algorithms by implementing them on NVIDIA GPUs. We enumerate our key results as follows:

1.5.1 Proxy Re-encryption

- We develop two IND-CPA-secure multi-hop unidirectional Proxy Re-Encryption (PRE) schemes on NTRU-RLWE [SS11] and BV [BV11b] homomorphic encryption schemes.
- We develop PRE schemes for GSW [GSW13] FHE scheme and it's RLWE variant Ring-GSW [KGV16] FHE scheme.
- We present an open-source C++ implementation of these PRE schemes in PALISADE lattice crypto software library and discuss several algorithmic and software optimizations.

1.5.2 Bootstrapping

- We introduce a new bootstrapping technique for symmetric key BV-LWE scheme and it's GSW analogue BV-GSW encryption scheme resulting in a fully homomorphic symmetric key scheme.
- We extend these bootstrapping techniques to work with secret keys generated from wider discrete Gaussian distributions without affecting the runtimes. These requirement are sometimes deemed necessary for compliance with *Homomorphic Encryption standards* [ACC+18].
- We introduce a new bootstrapping procedure, Gridstrapping which works on a large finite field represented as a multi-dimensional grid.

1.5.3 Implementation on NVIDIA GPUs

- We present a GPU implementation of BV-PRE and Ring-GSW-PRE schemes and design several other low level kernels for parallel execution. Our results show upto 228x factors of improvement for BV-PRE scheme and upto 11x improvement in performance for Ring-GSW-PRE scheme when compared with CPU implementation.
- We accelerate the performance of BV-LWE bootstrapping algorithm by porting over the critical operations to NVIDIA GPUs. Our results indicate a speedup of 2-8 times for lower values of relinearization factor, $r = 1$ and ring dimension $n = 512, 1024$.

CHAPTER 2

BACKGROUND AND PRELIMINARIES

In this chapter, we introduce some of the mathematical notations, background on lattices and the associated hard problems, algebra of polynomial rings and other definitions. This chapter only aims at giving a general background on lattice based cryptography and therefore compiled from several other prominent literatures [Pei09, Pei16, Reg04, Reg09] in the area.

2.1 Notations

We denote the set of integers by $\mathbb{Z}$, the set of non-negative integers by $\mathbb{N}$, the set of reals by $\mathbb{R}$ and the set of integers modulo some q by $\mathbb{Z}_q$. We denote scalars in plain (e.g., x) and vectors in bold lowercase (e.g., $\mathbf{v}$), and matrices in bold uppercase (e.g., $\mathbf{A}$). The i-th element of vector $\mathbf{v}$ is denoted by $\mathbf{v}[i]$ or v_i. The ℓ_i norm of a vector is denoted by $\|\mathbf{v}\|_i$. Infinite norm, ℓ_∞ of a vector $\mathbf{v}$ is given as $\|\mathbf{v}\|_\infty = \max_i |v_i|$. The norm of a polynomial $p(x)$ is the norm of its coefficient vector. Inner product is denoted by $\langle \mathbf{u}, \mathbf{v} \rangle$ and can be interpreted as $\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{v}^T \cdot \mathbf{u}$. Unless explicitly mentioned, all logarithms are assumed to be base 2. For a positive integer k, we let $[k] = \{0, \cdots, k-1\}$. We use $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ to denote respectively rounding down and up to the nearest integer. We use $\lfloor \cdot \rceil$ to denote rounding to the nearest integer, rounding up in case of ambiguity. When these operations are applied to a polynomial, we apply the respective rounding operation to individual coefficients of the polynomial.

2.2 Lattices

Lattices are regular arrangements of points in Euclidean space. A n-dimensional lattice Λ of rank $k \leq n$ is a discrete additive subgroup of $\mathbb{R}^n$. The lattice Λ is concretely generated from all integer linear combinations of some basis $\mathbf{B} =$

$\{\mathbf{b}_1, \cdots \mathbf{b}_k\}$ where the columns $\mathbf{b}_i$'s are linearly independent vectors.

$$\Lambda = \mathcal{L}(\mathbf{B}) = \{\mathbf{Bc} = \sum_{i \in [k]} c_i \cdot \mathbf{b}_i : \mathbf{c} \in \mathbb{Z}^k\}$$

Here, we are only interested in full-rank lattices, i.e., those for which $k = n$. Every lattice (of dimension $n > 1$) has an infinite number of lattice bases. If $\mathbf{B}_1$ and $\mathbf{B}_2$ are two lattice bases of Λ, then there is some unimodular matrix $\mathbf{U}$ (that has integer entries and $\det(\mathbf{U}) = \pm 1$) satisfying $\mathbf{B}_1 = \mathbf{U} \times \mathbf{B}_2$. To basis $\mathbf{B}$ of lattice Λ we associate the half-open parallelepiped $\mathcal{P}(\mathbf{B}) \leftarrow \{\sum_{i=1}^{n} x_i \mathbf{b}_i : x_i \in [-1/2, 1/2)\}$. The determinant of a lattice $\det(\Lambda)$ defines the n-dimensional volume of the fundamental parallelepiped associated to $\mathbf{B}$.
The dual lattice of Λ, denoted Λ^* is defined as $\Lambda^* = \{\mathbf{x} \in \mathbb{R}^n : \forall\ \mathbf{v} \in \Lambda, \langle \mathbf{x}, \mathbf{v} \rangle \in \mathbb{Z}\}$. It holds that $\det(\Lambda) \cdot \det(\Lambda^*) = 1$. Further, if $\mathbf{B}$ is a basis for the full-rank lattice Λ, then the dual basis $\mathbf{B}^* = (\mathbf{B}^{-1})^{\mathrm{T}}$ is in fact a basis of Λ^*. By symmetry, we can write $(\Lambda^*)^* = \Lambda$.

Definition 2.2.1. *(Minimum distance) The minimum distance of a lattice Λ is the length of a shortest nonzero lattice vector:*

$$\lambda_1(\Lambda) = \min_{\mathbf{v} \in \Lambda \setminus \{0\}} \|\mathbf{v}\|$$

2.2.1 Computational Problems

The geometrical properties of lattices allow us to define some hard combinatorial problems. In [Ajt96], Ajtai showed the first construction of a cryptographic primitive which can be based on worst-case to average-case reduction of lattice problems. Specifically, Ajtai introduced the short integer solution (SIS) problem and proved that solving it is at least as hard as approximating various lattice problems in the worst case. In another work, Ajtai and Dwork [AD97] constructed a probabilistic public key cryptosystem and reduced the security of the scheme to intractability of unique shortest vector problem (u-SVP). These computationally hard problems on lattices along with others serve as fundamental basis for the design of some powerful cryptographic primitives such as fully homomorphic encryption schemes, digital signature schemes, identity-based encryption etc. In relation to FHE, the lattice problems often used to establish the semantic security of the schemes are mainly the approximate shortest vector problem and the shortest independent vector problem (SIVP). We briefly present an overview of these hard problems and their definitions.

Definition 2.2.2. *(Shortest Vector Problem (SVP)). Given an arbitrary basis* $\mathbf{B}$ *of some lattice* $\Lambda = \mathcal{L}(\mathbf{B})$*, find a shortest nonzero lattice vector, i.e., a vector* $\mathbf{v} \in \Lambda$ *such that* $\|\mathbf{v}\| = \lambda_1(\Lambda)$*.*

While several algorithms are known to solve SVP in exponential and even super exponential time complexity, cryptosystems mostly rely on approximate variant of the lattice problems to derive computational hardness. These approximations are parameterized by a factor $\gamma \geq 1$.

Definition 2.2.3. *(Approximate Shortest Vector Problem (SVP$_\gamma$)). Given an arbitrary basis* $\mathbf{B}$ *of an n-dimensional lattice* $\Lambda = \mathcal{L}(\mathbf{B})$*, find a nonzero vector* $\mathbf{v} \in \Lambda$ *such that* $\|\mathbf{v}\| \leq \gamma \cdot \lambda_1(\Lambda)$*.*

The approximate SVP problem is known to be solved by family of lattice basis reduction algorithms such as LLL [LLL82] algorithm which admits solution in polynomial time when the approximation factor is very large, $\gamma = 2^{\Omega(n)}$.

Definition 2.2.4. *(Decisional Approximate Shortest Vector Problem (SVP$_\gamma$)). Given an arbitrary basis* $\mathbf{B}$ *of an n-dimensional lattice* $\Lambda = \mathcal{L}(\mathbf{B})$*, where* $\lambda_1(\Lambda) \leq 1$ *or* $\lambda_1(\Lambda) > \gamma$*, determine which is the case.*

Definition 2.2.5. *(Approximate Shortest Independent Vector Problem (SIVP$_\gamma$)). Given an arbitrary basis* $\mathbf{B}$ *of an* n*-dimensional lattice* $\Lambda = \mathcal{L}(\mathbf{B})$*, output a set* $\mathcal{S} = \{\mathbf{v}_1, \cdots, \mathbf{v}_n\} \in \Lambda$ *of* n *linearly independent lattice vectors such that* $\|\mathbf{v}_i\| \leq \gamma \cdot \lambda_i(\Lambda)$ *for all* $i \in [n]$*.*

Definition 2.2.6. *(Approximate Bounded Distance Decoding Problem (BDD$_\gamma$)). Given an arbitrary basis* $\mathbf{B}$ *of an* n*-dimensional lattice* $\Lambda = \mathcal{L}(\mathbf{B})$ *and a target vector* $\mathbf{t} \in \mathbb{R}^n$ *such that* $\boldsymbol{Dist}(t, \Lambda) \leq \gamma^{-1} \cdot \lambda_1(\Lambda)$*, find a vector* $\mathbf{v} \in \Lambda$ *such that* $\|\mathbf{v} - \mathbf{t}\| = \boldsymbol{Dist}(t, \Lambda)$*.*

The BDD$_\gamma$ problem is a slight variation of the approximate closest vector problem (CVP) which searches for a vector $\mathbf{v}$ close to target point $\mathbf{t} \in \mathbb{R}^n$.

2.2.2 Gaussian Distributions

We review some Gaussian measures over lattices in this section. Gaussian properties are mainly used to substantiate the claim of computational hardness of lattice problems by generating noise vectors from Gaussian distributions with varying parameters. For any $s > 0$, we define the Gaussian function on $\mathbb{R}^n$ centered at $\mathbf{c}$ with parameter s:

$$\forall\, \mathbf{x} \in \mathbb{R}^n,\ \rho_{s,\mathbf{c}}(\mathbf{x}) = \exp\left(-\pi \|\mathbf{x} - \mathbf{c}\|^2 / s^2\right)$$

Deviation s and center c are taken to be 1 and 0 respectively when omitted.

For any $\mathbf{c} \in \mathbb{R}^n$, real $s > 0$, and n-dimensional lattice Λ, define the discrete Gaussian distribution over Λ as:

$$\forall\, \mathbf{x} \in \Lambda,\ \mathcal{D}_{\Lambda,s,c}(\mathbf{x}) = \frac{\rho_{s,\mathbf{c}}(\mathbf{x})}{\rho_{s,\mathbf{c}}(\boldsymbol{\Lambda})}$$

Definition 2.2.7. *(Addition of Gaussians). Let* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *be Gaussian distributions with parameters* s_1 *and* s_2*, respectively. Then distribution obtained by sampling*

$\mathcal{D}_1$ and $\mathcal{D}_2$ and summing them results in another Gaussian distribution $\mathcal{D}_+$ with parameter $\sqrt{s_1^2 + s_2^2}$.

Smoothing parameter: In [MR07] Micciancio and Regev introduced an important parameter termed as smoothing parameter which shows the uniformity of a vector when generated from certain noise vector with radius at least as large as the smoothing parameter. It is defined as follows:

Definition 2.2.8. *For a n-dimensional lattice Λ, and positive real $\epsilon > 0$, smoothing parameter, $\eta_\epsilon(\Lambda)$ is defined as the smallest s such that $\rho_{1/s}(\Lambda^* \setminus \{0\}) \leq \epsilon$.*

2.3 Learning with Errors (LWE)

Construction of lattice based public-key encryption schemes were greatly simplified by the introduction of learning with errors (LWE) problem by Regev in [Reg09]. The LWE problem is seen as an extension of the learning from parity with error problem with the modulus raised to higher values. In this work, Regev showed that if the LWE problem can be solved by a polynomial time algorithm then, this implies the existence of an efficient quantum algorithm which can solve the decision version of shortest vector problem (GapSVP) and the shortest independent vectors problem (SIVP) to within $\tilde{\mathcal{O}}(n/\alpha)$ factor in the worst case. Here, α is the error rate such that $\alpha \in (0, 1)$. In another work [Pei09], Peikert showed a classical reduction of the LWE problem from GapSVP problem by relying on a BDD oracle that solves lattice problem.

LWE is parameterized by positive integers n and q, and an error distribution χ over $\mathbb{Z}$. The error distribution is usually a discrete Gaussian distribution with error rate α with $\alpha < 1$.

Definition 2.3.1. *(LWE distribution). For a vector $\mathbf{s} \in \mathbb{Z}_q^n$ called the secret, the LWE distribution $A_{\mathbf{s},\chi}$ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is sampled by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \leftarrow_{\$} \chi$, and outputting $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e \bmod q)$.*

Definition 2.3.2. *(Search-LWE$_{n,q,\chi,m}$). Given m independent samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ drawn from LWE distribution LWE$_{n,q,\chi}$, find s.*

Definition 2.3.3. *(Decision-LWE$_{n,q,\chi,m}$). Given independent samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ the problem is to decide whether the pair is sampled from LWE distribution LWE$_{n,q,\chi}$ or uniform distribution in $\mathbb{Z}_q^n \times \mathbb{Z}_q$.*

Regev further showed the construction of a public key encryption scheme and proved the semantic security of the scheme on indistinguishably of LWE samples (decision-LWE). In this scheme, secret key consists of a uniform vector $\mathbf{s} \in \mathbb{Z}_q^n$ chosen at random. Public key consists of m instances of the LWE problem given by $(\mathbf{a}_i, b_i)_{i=1}^m$. To encrypt a bit μ, we choose a random binary set $\mathcal{S} \in \{0,1\}^m$ and set encryption as $\left(\sum_{i\in\mathcal{S}} \mathbf{a}_i, \mu \cdot \lfloor \frac{q}{2} \rfloor + \sum_{i\in\mathcal{S}} b_i\right)$. To decrypt ciphertext $c = (\mathbf{a}, b)$, we compute $t = b - \langle \mathbf{s}, \mathbf{a} \rangle$. We output 0 if t is closer to 0 otherwise 1 if t is closer to $\lfloor \frac{q}{2} \rfloor$. In subsequent construction of LWE based FHE schemes, it is proved that generation of secret keys from narrower distribution leads to higher homomorphic capacity while retaining the quantum hardness of intractability of lattice problems.

In recent years, LWE has served as the basis for many other cryptosystems such as CPA secure encryption schemes [PVW08, LP11], oblivious transfer [PVW08], identity-based encryption [GPV08, CHKP10, ABB10a, ABB10b], fully homomorphic encryption [BV14a, GSW13] and many more.

2.4 Ring LWE

Ciphertexts in the LWE form suffer from the problem of dimension expansion upon homomorphic evaluation. This not only results in greater computational overhead but adds significant noise growth. To accelerate cryptographic constructions based on the learning with errors problem (LWE) Lyubashevsky, Peikert and Regev [LPR10] introduced the ring learning with error (RLWE) problem. Lyubashevsky *et al.* first showed a quantum reduction from approximate SVP (in the worst case) on ideal lattices in R to the search version of ring-LWE, where the goal is to recover the secret $s \in R_q$ (with high probability, for any s) from arbitrary number of noisy products. They also gave a reduction from the search problem to the decision variant, which shows that RLWE distribution is pseudo random assuming worst-case problems on ideal lattices are hard for polynomial-time quantum algorithms.

Efficiency: The primary reason for adopting RLWE security assumption is because of it's efficiency which is attributed to the algebraic and embedding properties of ideal lattices over ring. Moreover, RLWE can be viewed as n instances of LWE in a compact representation and operate simultaneously. Using the Fast Fourier Transform (FFT) or its variants like Number Theoretic Transforms (NTT), operations on polynomial rings can be restricted to $\mathcal{O}(n \log n)$ scalar operations.

2.4.1 Definitions

RLWE is parameterized by a polynomial ring R such that $R = \mathbb{Z}[X]/\Phi_m(X)$, where $\Phi_m(X)$ is the m-th cyclotomic polynomial of degree n. Also, let $q \geq 2$ be an integer modulus, and let $R_q = R/qR$ be the quotient ring. Finally, let χ be an error distribution over R having an error rate $\alpha < 1$.

Definition 2.4.1. *(RLWE distribution). For a polynomial ring $s \in R_q$ called the secret, the RLWE distribution $A_{s,\chi}$ over $R_q \times R_q$ is sampled by choosing $a \in R_q$ uniformly at random, sampling $e \leftarrow_{\$} \chi$, and outputting $(a, b = s \cdot a + e \bmod q)$.*

Definition 2.4.2. *(Search-RLWE$_{n,q,\chi,m}$). Given m independent samples $(a_i, b_i) \in R_q \times R_q$ drawn from RLWE distribution RLWE$_{n,q,\chi}$, find secret polynomial s.*

Definition 2.4.3. *(Decision-RLWE$_{q,\chi,m}$). Given independent samples $(a_i, b_i) \in R_q \times R_q$ the problem is to decide whether the pair is sampled from RLWE distribution RLWE$_{n,q,\chi}$ or uniform distribution over $R_q \times R_q$.*

Lyubashevsky *et al.* [LPR10] further described a simple cryptosystem, a RLWE analogue of Regev's encryption scheme, which based the semantic security on the pseudo randomness of RLWE (decision-RLWE). In this scheme, secret key, s is generated from the error distribution χ. To generate the public key, pk we choose a uniformly random element $a \in R_q$ and sample a error, e from error distribution χ. Set the public key as RLWE pair, $pk = (a, b = a \cdot s + e) \in R_q^2$. The RLWE scheme is capable of encrypting a n-bit message $\mu \in \{0, 1\}^n$ as opposed to a single bit in the LWE scheme. The encryption algorithm then samples three random ring elements $r, e_1, e_2 \in R$ from the error distribution χ and sets the ciphertext, c as follows:

$$u = a \cdot r + e_1 \bmod qv = b \cdot r + e_2 + \mu \cdot \lfloor q/2 \rceil \bmod q$$

$$c = (u, v) \in R_q^2$$

The decryption algorithm computes the noise term t as follows:

$$t = v - u \cdot s = (r \cdot e - s \cdot e_1 + e_2) + \mu \cdot \lfloor q/2 \rceil \bmod q.$$

To recover the bit vector, we proceed to round each coefficient of t to either 0 or $\lfloor q/2 \rceil$, whichever is closest modulo q.

RLWE security assumption has been a huge success in construction of FHE schemes and numerous variations of RLWE based FHE schemes have been proposed in literature [BV11b, BGV14, LTV13, Bra12].

2.4.2 Cyclotomic Polynomials

Cyclotomic polynomials, $\Phi_m(X)$ in the definition of ring polynomials are mostly used to embed integer coefficients in field extensions $K \mapsto \mathbb{C}^n$.

Definition: For any positive integer m, the m-th cyclotomic polynomial $\Phi_m(X)$ is the minimal polynomial with integer coefficients such that $X^m - 1 = 0$ and is not a divisor of $\left(X^k - 1\right)$ for any $k < m$. It is defined by

$$\Phi_m(X) = \frac{X^m - 1}{\prod_{1\leq d<n, d|n} \Phi_d(x)} \tag{2.1}$$

When the field extension $K(\zeta_m)$ is a m-th order multiplicative group where ζ_m is a primitive m-th root of unity, we can derive the cyclotomic polynomial as follows:

$$\Phi_m(X) = \prod_{i\in\mathbb{Z}_m^*} \left(X - \zeta_m^i\right) \tag{2.2}$$

As per the above definition, degree of the cyclotomic polynomial is given by $n = \varphi(m)$, the totient function of m. For prime m, cyclotomic polynomial is reduced to $\Phi_m(X) = X^{m-1} + X^{m-2} + \cdots + X^2 + X + 1$. When m is a power of two the cyclotomic polynomial is maximally sparse and given by $\Phi_m(X) = X^n + 1$, where $n = \varphi(m) = m/2$. Most of the RLWE based cryptosystems use a power of two cyclotomics as polynomial multiplication can be performed efficiently by a simple tweak of the classical n-dimensional FFT algorithm. Even though power of two cyclotomic polynomials are pervasive in RLWE cryptosystems, there are special cases where arbitrary cyclotomic polynomial rings are deemed worthy. Most notably, non-power-of-two cyclotomic rings are useful in obtaining more nimble movement of plaintext in ciphertext slots as shown in [GHS12c]. However, it should be noted that FFT algorithms tailored for non-power-of-two cyclotomic rings are relatively inefficient because of large constants hidden factors and rather complex and hard to implement. A toolkit of modular algorithms for designing applications to work in arbitrary cyclotomic rings was described in [LPR13] and implemented in PALISADE lattice crypto software library. Particularly in our implementation of non-power-of-two cyclotomic rings we used the definition 2.1 to generate and store cyclotomic polynomials.

2.4.3 Plaintext Slots and Embedding

It was first shown by Smart and Vercauteren [SV14] that by an application of the Chinese Remainder Theorem to number fields the plaintext space in polynomial ring can be partitioned into a vector of "plaintext slots". Addition and multiplication of polynomial rings in R correspond to component-wise operations on these plaintext slots and hence it is sometimes referred as SIMD or batch operations. To understand how this finite field splits and the homomorphisms between them, we let the plaintext be $a(X) \in R_2$. We assume the cyclotomic polynomial $\Phi_m(X)$ of degree n to split into exactly r distinct irreducible factors of degree $d = n/r$.

$$\Phi_m(X) = F(X) = \prod_{i=1}^{r} F_i(X)$$

This implies that, algebra on the plaintext space $a(X)$ now has a natural isomorphisms w.r.t. a number of plaintext slots $a_i(X)$ obtained by a direct application of polynomial Chinese Remainder Theorem.

$$a(X) \mod \Phi_m(X) \cong \frac{a(X)}{F_1(X)} \otimes \frac{a(X)}{F_2(X)} \cdots \otimes \frac{a(X)}{F_r(X)}$$

In practice, we replace the plaintext modulus 2 with a higher modulus p such that $p = 1 \bmod m$ and plaintext space R_p splits into r isomorphic subfield. In the simplest case, to encode or batch r integers in $\mathbb{Z}_p$ in the plaintext space R_p we apply inverse nega-cyclic transform when working on power of two cyclotomics. To summarize, the SIMD plaintext batching allows to amortize the cost of homomorphic operations by a factor of r which in turn is a function of the security parameter n and therefore leads to higher efficiency.

2.4.4 Automorphisms Transforms

In general, automorphism transform define a set of permutations in the plaintext space R_p. In [LPR10], the authors used the permutation properties of automorphism group to prove the pseudo-randomness of RLWE. However, the automorphism transform is more useful for obtaining flexible movement of data across the plaintext slots.

We denote the automorphism transform as $\tau_\kappa : K \mapsto K$ for a positive integer $\kappa \in \mathbb{Z}_m^*$ acting on the m-th cyclotomic number field represented as field extension $K = \mathbb{Q}(\zeta)$ having ζ_m as the primitive root of unity. The number field K has n automorphisms where each transform acting on a polynomial ring, $a \in R$ is represented as $\tau_\kappa(a(X)) \mapsto a(X^\kappa)$ or simply $\tau_\kappa : a \mapsto a^\kappa$. Since, there exist a transform for every $i \in \mathbb{Z}_m^*$ and $j \in \mathbb{Z}_m^*$, $\ni i \cdot j = 1 \bmod m$, the set of transformations forms a multiplicative group under composition. When automorphism transforms, τ_κ are applied to a plaintext $a \in R$ they act transitively on the plaintext slots. Specifically, for a non-power-of-two cyclotomic ring these transforms shift the vector of slots cyclically. For example, if the ring a encodes integers $(a_0, a_1, \cdots, a_{n-1})$, then after a transformation we could get a ring $a' = (a_{n-1}, a_0, \cdots, a_{n-2})$. Most importantly, these set of automorphism transforms can be directly applied on a ciphertext without affecting the norm of error. However, applying automorphism operations on cipertexts have the downside of morphing the secret keys and therefore such operations are generally followed with additional procedures to switch the secret key back to its original form.

2.5 Syntax of Cryptographic Primitives

In this section, we describe the basic syntax of secret-key, public-key and homomorphic encryption schemes and the security notions associated with them.

Definition 2.5.1. *(Secret-key encryption scheme). A secret-key encryption scheme consists of the following algorithms:*

- ***KeyGen***$(1^\lambda) \mapsto (s, pp)$ *It takes input security parameter λ, outputs the key s and some public parameters pp;*
- ***Encrypt***$(s, m) \mapsto c$ *It takes secret key s and a message m, and outputs a ciphertext c;*
- ***Decrypt***$(s, c) \mapsto m$ *It takes secret key s and a ciphertext c, and outputs a message m;*

A public-key (also known as asymmetric-key) encryption scheme consists of separate keys for encryption and decryption, and defined as follows:

Definition 2.5.2. *(Public-key encryption scheme). A public-key encryption scheme consists of the following algorithms:*

- ***KeyGen***$(1^\lambda) \mapsto (sk, pk, pp)$ *It takes input security parameter λ, outputs the secret key sk, public key pk, and some public parameters pp;*
- ***Encrypt***$(pk, m) \mapsto c$ *It takes public key pk and a message m, and outputs a ciphertext c;*
- ***Decrypt***$(sk, c) \mapsto m$ *It takes secret key sk and a ciphertext c, and outputs a message m;*

A homomorphic encryption can be described as a public-key encryption scheme which can evaluate a certain class $\mathcal{C}$ of circuits. Formally, it is defined as follows:

Definition 2.5.3. *(Homomorphic Encryption scheme). A $\mathcal{C}$-Homomorphic encryption scheme consists of the following algorithms:*

- ***KeyGen***$(1^\lambda) \mapsto (sk, pk, pp, ek)$ *It takes input security parameter λ, outputs a secret key sk, a public key pk, a (public) evaluation key ek and some public parameters pp;*
- ***Encrypt***$(pk, m) \mapsto c$ *It takes public key pk and a message m, and outputs a ciphertext c;*
- ***Decrypt***$(sk, c) \mapsto m$ *It takes secret key sk and a ciphertext c, and outputs a message m;*
- ***Eval***$(ek, \mathcal{C}, c_1, \cdots, c_\ell) \mapsto c$ *Given a evaluation key ek, a (description of a) circuit $\mathcal{C}$ and ℓ ciphertexts $c_1, \cdots, c_\ell$, it outputs a ciphertext c. For the correctness of evaluation we require **Decrypt**$(sk, c) = \mathcal{C}(m_1, \cdots, m_\ell)$ where $c_i =$ **Encrypt**(pk, m_i).*

We now give a standard notion of security for an encryption scheme $\mathcal{E}$, namely indistinguishably under chosen-plaintext attacks or IND-CPA security [GM84]. The

```
Game IND-CPA$_{\mathcal{E}}$
1 :  $(\mathsf{pk}, \mathsf{sk}, pp) \leftarrow_{\$} \mathrm{KeyGen}\left(1^{\lambda}\right)$
2 :  $(m_0, m_1) \leftarrow_{\$} \mathcal{A}\left(1^{\lambda}, \mathsf{pk}, pp\right)$
3 :  $b \leftarrow_{\$} \{0, 1\}$
4 :  $c \leftarrow \mathcal{E}.\mathrm{Encrypt}(\mathsf{pk}, m_b)$
5 :  $b' \leftarrow \mathcal{A}\left(1^{\lambda}, \mathsf{pk}, pp, c\right)$
6 :  **return** $b = b'$
```

Figure 2.1 Game describing IND-CPA security.

IND-CPA notion guarantees that the encryption reveals nothing about encrypted messages to a passive (eavesdropping) adversary.

Definition 2.5.4. *(Indistinguishability under chosen-plaintext attacks). For a public-key encryption scheme $\mathcal{E}$, we define IND-CPA security via the game depicted in figure 2.1. Let the advantage of an adversary $\mathcal{A}$ when playing the game IND-CPA$_{\mathcal{E}}$ be as follows:*

$$\mathsf{Adv}_{\mathcal{A}}^{\text{ind-cpa}}(\lambda) = \left| \Pr[\text{IND-CPA}_{\mathcal{E},\lambda}] - \frac{1}{2} \right|$$

We say that $\mathcal{E}$ is IND-CPA *secure if, for any PPT adversary $\mathcal{A}$, it holds that $\mathsf{Adv}_{\mathcal{A}}^{\text{ind-cpa}}(\lambda) = \mathsf{negl}(\lambda)$.*

CHAPTER 3

PROXY RE-ENCRYPTION

Over the past decade, computing technologies such as mobile computing, cloud computing, data analytic, machine learning, etc. have matured to accommodate the needs and expectations of consumers. Today, a wide variety of services are hosted on the web or cloud computing systems. This paradigm of cloud computing has attracted small vendors and individual users to host their services on servers of large tech corporations. This has resulted in spawning collections of consumer data. Because of the sensitivity of consumer data, these cloud platforms are under threat of attack. Cloud platforms enforce a wide range of network security solutions to safeguard their computing infrastructure but in spite of counter measures and network security techniques, a flurry of data breaches have been reported attracting countless consumer lawsuits.

Traditional approaches to security are not sufficient to guarantee privacy and confidentiality. Security constraints which enforce stronger threat models are needed in practice. Under this new threat model we assume the cloud to be a malicious entity and confidentiality is enforced by cryptographic solutions. A user should interact with the cloud by publishing encrypted contents onto the cloud and only decrypt the contents at its end. In this scenario, even in the event of a successful attack the possibility of data breach is reduced. However, encrypting user data severely limits the possibility of sharing it across other users or applications.

Proxy re-encryption is a cryptographic primitive that presents an elegant solution to this problem. In a simple two user setting there are two parties, Alice and Bob. Alice stores encrypted data on an untrusted cloud and can read the data when decrypted with her secret key. When Bob needs to access the data, Alice generates a re-encryption key and gives it to the untrusted cloud. The cloud can now transform the data encrypted by Alice into a ciphertext that can be decrypted by Bob's secret key. Moreover, the cloud doesn't learn anything about the underlying data in this process.

In the context of cloud computing, PRE scheme can be interpreted as an access delegation mechanism which allows users to delegate rights to the cloud while enforcing a stronger security notion of an untrusted cloud. It can be easily seen that a PRE scheme is not just restricted to cloud computing environments but the same idea can be extended to many such privacy concerning applications. PRE schemes have been known to be used in digital rights management (DRM) system [TCG06], secure file storage system [AFGH06], email list services [KHP06] and many other applications.

Proxy Re-Encryption, as we have described it, is also called a *unidirectional* PRE scheme, and was defined in [ID03, AFGH06]. Henceforth, when we refer to proxy re-encryption, we mean a *unidirectional* scheme. In this chapter we describe four new IND-CPA-secure Proxy Re-Encryption (PRE) schemes and their implementations where the PRE functionality is provided using the LWE/RLWE key switching procedure. The first scheme (NTRU-ABD-PRE) is based on the NTRU encryption scheme with RLWE modifications [SS11] where the NTRU immunity constraint against subfield lattice attacks is applied to set the distribution parameter for NTRU key generation [ABD16]. The second scheme (BV-PRE) is based on the BV homomorphic encryption scheme [BV11b] and relies only the RLWE assumption. Our third scheme GSW-PRE is based on GSW [GSW13] homomorphic encryption scheme. The security of GSW FHE scheme is based on the LWE [Reg09] assumption. Our last PRE scheme, Ring-GSW-PRE is based on a Ring variant of GSW scheme, Ring-GSW FHE scheme [KGV16] where security is based on RLWE [LPR10] assumption which was introduced to speed up LWE based cryptosystems. Utilizing techniques from our

RLWE based PRE scheme we introduce a key-switching procedure for Ring-GSW FHE scheme. Further, we show that our key-switching procedure can be extended to enable Automorphism operations over cyclotomic polynomials. Owing to asymmetric noise growth property of Ring-GSW FHE scheme, this directly leads us to SIMD FHE computations with reduced noise.

A *unidirectional* PRE scheme can be constructed from any fully homomorphic encryption (FHE) scheme using the procedure based on double encryption of plaintext and evaluation of decryption circuit [G$^+$09]. However, this approach relies on heavyweight tools generally used for bootstrapping. The goal of this work is to avoid the use of these computationally expensive tools and study elementary and efficient constructions of lattice-based proxy re-encryption schemes based on LWE or RLWE key switching.

As opposed to other known approaches to PRE, lattice encryption approaches, such as ours, are generally considered post-quantum [MR09, Reg04], that is, potentially secure against attacks even from adversaries with practical quantum computing devices in addition to adversaries with classical computing devices. Our goal is to provide a software PRE capability that can support high-throughput pub-sub information sharing without direct interactions between publishers and subscribers. We provide experimental evaluation of software implementations of our PRE schemes to evaluate its security, scalability and performance. We further show that our PRE schemes based on GSW FHE scheme have the added advantage of asymmetric noise growth which permits evaluation of deeper circuits. We provide a direct comparison of Ring-GSW-PRE scheme with BV-PRE [PRSV17] scheme and highlight the difference in capabilities in terms of circuit depth and number of hops for the same set of parameters.

Motivation for extending PRE scheme for GSW and Ring-GSW FHE schemes: Before the advent of fully homomorphic encryption (FHE), PRE scheme's only role was to transform the ciphertext while preserving the message as it arrived from the producer. In other words, a PRE scheme dealt with *read-only* data. Designing a PRE scheme that has homomorphic capabilities opens up the door for many interesting applications. For example, consider the case of health care sector where different organizations may be involved in running diagnostic inference on patient health information collected from hospitals. However, due to sensitivity of information, US laws and regulations (HIPAA) do not allow such direct outsourcing of information. It is therefore more appropriate to encrypt patient information with a FHE scheme and then send it to various diagnosing bodies who can run blind evaluations on the data. Once the results are processed, it can be re-encrypted by a proxy which in turn can be deciphered into meaningful information by the patient. In other scenarios, we can form an information processing pipeline where each node evaluates and forwards the information to next node after re-encryption in a *multi-hop* manner.

Modern lattice based schemes require evaluating a re-encryption operation or function on ciphertexts which always leads to increase in noise in them. Hence, our goal is to design a robust and computationally efficient PRE scheme that is amenable to larger depth of computation over multiple hops without bloating the parameters too much. While BV-PRE and NTRU-ABD-PRE are PRE schemes which have homomorphic capabilities, the nature of noise growth in these schemes is quadratic. In order to achieve asymmetric nature of noise growth, it is highly pertinent to develop PRE schemes based on GSW and Ring-GSW FHE schemes.

BV-PRE outperforms NTRU-ABD-PRE and other PRE schemes in both single-hop and multi-hop settings, and is provably secure under the RLWE assumption, in contrast to the NTRU variant which is proven secure under a less well-studied variant of the so-called NTRU assumption. BV-PRE scales well with the number of hops due to a relatively small additive noise growth provided by the BV scheme and

RLWE key switching procedure. BV-PRE has small ciphertext modulus and ring dimension requirements: successful decryption after re-encryption can be achieved using a 17-bit ciphertext modulus and ring dimension of 512 (for at least 100 bits of security). This translates to submillisecond encryption/decryption runtime and re-encryption runtime of under 5 ms. When compared to the LWE-based PRE lattice schemes recently proposed in [ABPW13, Kir14, FL16a, PWA$^+$16], our BV-PRE scheme provides key sizes and ciphertext expansion factors as good as or better than the key sizes of any other lattice-based PRE schemes, and lower time complexity than any other LWE-based scheme.

A GPU implementation of GSW PRE shows 5x acceleration in key generation time and more than 100x in encryption, re-encryption and re-encryption key generation run times over CPU implementation. For Ring-GSW PRE scheme we complete critical operations under 10 milliseconds.

Chapter Organization. In Section 3.1 we review related results on encrypted computing and PRE. In Section 3.2 we discuss the the high-level concept and performance and security tradeoffs of PRE. In Sections 3.3, 3.4, 3.5 and 3.6 we describe the proposed lattice-based PRE cryptosystems. In Section 3.7 we discuss parameter selection for these schemes to provide practical secure computing on commodity computing hardware. In Section 3.8 we describe the overall software architecture of the library to support the end-to-end encrypted application. In Section 3.9 we discuss experimentation and evaluation of our design and implementation. In Section 3.10 we present practical use cases for the proposed PRE cryptosystem. We conclude the paper with a discussion of our insights and future work in Section 3.11.

3.1 Related Work

3.1.1 Proxy Re-Encryption

The notion of Proxy Re-Encryption (PRE) was introduced in the work of Blaze, Bleumer and Strauss [BBS98], where they also presented a construction based on the El-Gamal encryption scheme. Their construction was a *bidirectional* proxy re-encryption scheme in that a re-encryption key can be used to translate encrypted data from the publisher encryption to subscriber encryption but also in reverse, from the subscriber encryption to publisher encryption. In contrast, in this work, we focus on *unidirectional* proxy re-encryption that provides tighter control on which ciphertexts can be re-encrypted and to whom.

Unidirectional PRE schemes were first proposed in [ID03, AFGH06]. The scheme in [AFGH06] is based on the decisional bilinear Diffie-Hellman (DBDH) assumption. The schemes in [ID03, AFGH06] are single-hop proxy re-encryption schemes, meaning that a re-encrypted ciphertext cannot be re-encrypted further, to a third party.

Also, related is the work in [CH07] which provides a multi-hop bidirectional scheme based on bilinear maps. *Multi-hop* re-encryption schemes are important in applications where re-encryption needs to be applied multiple times, for example where information needs to be brokered in multiple steps from publisher to subscriber. We refer the reader to Sections 3.10.1 and 3.10.2 for a discussion of applications.

Our approach to PRE is based on two common ring variants of lattice-based homomorphic encryption schemes, with the PRE functionality provided using the LWE/RLWE key switching procedure of Brakerski and Vaikuntanathan [BV11b]. The first scheme (NTRU-ABD-PRE) is built on top of the NTRU encryption scheme [HPS98] with RLWE modifications [SS11] where the NTRU immunity constraint against subfield lattice attacks is applied to set the distribution parameter for NTRU key generation (c.f. [ABD16]). The second scheme (BV-PRE) is based on the BV homomorphic encryption scheme [BV11b] and relies only on the RLWE assumption. The third scheme (GSW-PRE) and fourth scheme (Ring-GSW-PRE) are based on GSW [GSW13] homomorphic encryption scheme and rely on LWE and RLWE assumptions respectively. FHE schemes are encryption schemes that allow anyone to run computations over encrypted data without decrypting the data.

It is at times difficult to establish direct comparisons between encryption schemes, even with similar computational hardness underpinnings. Following [CN11, LP11, MR09, vdP12], we use the standard "root Hermite factor" δ as the primary measure of the concrete security of RLWE-based encryption schemes, for a given set of parameters, where a smaller δ provides more security. Experimental evidence [CN11] suggests that $\delta = 1.007$ would require roughly 2^{40} core-years on recent Intel Xeon processors to break. We set the configuration parameters to attain δ of just less than 1.006 for each of the schemes for our parameter and key size comparisons, and for our later experimental analyses. The root Hermite factor parameter setting we use of $\delta < 1.006$ arguably provides at least 100 bits of security [CN11, LP11, MR09, vdP12].

Whereas our BV-PRE scheme provides sub-millisecond runtimes for optimal parameter settings for encryption and decryption operations and millisecond-order runtimes for re-encryption, the experimental results of [AFGH06] are in the ranges of 3 to 9 ms (for 256 bits of security) and 8 to 27 (for 512 bits of security) for these same operations. However, the experimental results of the non-lattice-based work in [AFGH06] are shown for 256 and 512 bits of security rather than approximately 100 bits in our case. Our estimates using equation (7) in [GHS12c] show that $\delta \approx 1.0034$ and $\delta \approx 1.002$ correspond to 256 and 512 bits of securiy, respectively. These values of δ increase the minimum ring dimension for 256 bits of security to 1024 and for 512 bits of security to 2048, while keeping the bit width approximately the same. This implies that the runtime is roughly doubled when one goes from 100 bits of security to 256 and then doubles again when going from 256 to 512 bits of security, which suggests that our runtimes are comparable to those reported in [AFGH06].

An independent work of [NAL15] proposes and implements a IND-CPA-secure proxy re-encryption scheme based on the NTRU encryption scheme with RLWE modifications [SS11], which they label as PS-NTRUReEncrypt. This PRE scheme relies on a variant of NTRU assumption. The PS-NTRUReEncrypt construction is a bidirectional PRE scheme, whereas ours is unidirectional (see Section 3.10 for why unidirectional schemes are critical to our applications). The runtimes reported in [NAL15] are of the order of one second. The authors [NAL15] also propose and implement another bidirectional (more efficient but not IND-CPA-secure) scheme called NTRUReEncrypt with runtimes of the order of one millisecond. However, NTRUReEncrypt is not directly comparable to our schemes in security as its security relies on an ad-hoc new assumption. It is therefore unclear how to set key-sizes for this scheme, and hence, we will not discuss this scheme further in this paper. We note, however, that our RLWE-based BV-PRE scheme achieves a comparable, even better, performance with the added provable security guarantee based on the relatively well-studied RLWE problem.

Several LWE-based PRE lattice schemes have recently been proposed in [ABPW13, Kir14, FL16b, PWA$^+$16]. The schemes presented in [Kir14, FL16b] are based on a Regev-style encryption, which can be regarded as an extension of the CCA-secure public key encryption scheme developed in [MP12]. The schemes developed in [ABPW13, PWA$^+$16, FL16b] are based on a public key encryption scheme formulated in [LP11]. [FL16b] shows an implementation of a IND-CPA-secure multi-hop scheme. The most efficient implemented variant [PWA$^+$16], which we label as IND-CPA-LWE, is similar to BV-PRE but relies on the LWE assumption (instead of ideal lattices and RLWE assumption). This scheme is also unidirectional and supports multiple hops of re-encryptions.

Table 3.1 Parameter Configuration and Key Size Comparison of LWE-based IND-CPA-Secure PRE Schemes for Normalized Conditions

Scheme	Parameters for Secure Configuration			Key Sizes for Secure Operation, KB			Asymptotic Key Sizes		
	δ	n	k	**sk**	**pk**	**rk**	**sk**	**pk**	**rk**
BV-PRE	1.0051	512	17	1.06	2.13	36.1	nk	$2nk$	$2nk^2$
NTRU-ABD-PRE	1.0054	1024	35	4.38	4.38	153	nk	nk	nk^2
GSW-PRE	1.0033	1023	22	2.74	1.23×10^5	6.32×10^7	nk	$2n(n+1)k^2$	$(n+1)^3k^2$
Ring-GSW-PRE	1.0033	1024	22	2.75	5.5	484	nk	$2nk$	$8nk^2$
PS-NTRUReEncrypt [NAL15]	1.0037	2048	47	11.8	11.8	11.8	nk	nk	nk
IND-CPA-LWE [PWA+16]	1.0042	450	14	346	692	9,690	n^2k	$2n^2k$	$2n^2k$

Plaintext Modulus $p = 2$, Key Switching Window $r = 1$, Message Length $l = n$ (in the LWE Scheme) with Comparable Security (Root Hermite Factor δ is Under 1.006; Bound Corresponds to Approximately 100 Bits of Security).

Table 3.1 shows the comparison of parameter selections, resulting concrete secure key sizes and asymptotic key sizes for the following LWE-based IND-CPA-secure PRE schemes: NTRU-ABD-PRE, BV-PRE, GSW-PRE, Ring-GSW-PRE, PS-NTRUReEncrypt [NAL15], and the IND-CPA-secure LWE scheme [PWA+16]. We base these comparisons on roughly equivalent functionality and security configurations. For notational simplicity we define $k = \lfloor \log_2 q + 1 \rfloor$, the number of bits required to represent the ciphertext modulus q. For the concrete parameters in the table, we set the ring dimension n (referred to as the lattice security parameter n in the case of the LWE scheme) and ciphertext modulus q for each of the schemes for a single-hop use case for plaintext modulus $p = 2$ to ensure that the security parameter $\delta < 1.006$. Note that for the PS-NTRUReEncrypt and IND-CPA-LWE schemes we use a tighter bound on the root Hermite factor δ due to the parameter selection decisions made in the papers we cited for the schemes.

In comparison with prior lattice-based PRE schemes:

- The key sizes and ciphertext expansion factors of BV-PRE and NTRU-ABD-PRE are as good as or better than the key sizes of the other lattice-based PRE schemes.
- The ciphertext expansion factor of NTRU-ABD-PRE and PS-NTRUReEncrypt is k and $2k$ for BV-PRE and CP-LWE. However, NTRU-ABD-PRE and PS-NTRUReEncrypt require higher parameter values, which automatically increases space requirements for the secret and public keys and encryption/decryption execution time.

- Noise grows multiplicatively with the number of re-encryption hops in the case of NTRU-ABD-PRE (at most two hops are supported). BV-PRE, Ring-GSW-PRE, IND-CPA-LWE, and PS-NTRUReEncrypt can support up to 100 hops without significantly increasing the ring dimension (lattice security parameter) and ciphertext bit length due to additive noise growth.
- Although the re-encryption space and time complexity for PS-NTRUReEncrypt is lower, this scheme is bidirectional and does not support the same security use cases as BV-PRE and IND-CPA-LWE.
- IND-CPA-LWE has much higher space requirements (an additional factor of n in the size of all keys) as compared to BV-PRE, which limits its applicability to embedded systems.

The above analysis implies that BV-PRE is more efficient for Pub/Sub systems than all existing lattice-based PRE schemes.

We also provide a high-level theoretical evaluation of the performance of our schemes in comparison with other identified recent lattice-based IND-CPA-secure PRE schemes. Rather than base this initial comparison on experimental runtime performance, we compare performance in terms of the computational operations which are generally the lower-level computational building blocks provided by math libraries and hardware accelerators which implementations are built from. In particular, we couch our evaluation of theoretical performance in terms of the number of slightly higher-level polynomial operations, inclusive of Number Theoretic Transforms (NTT's), Vector Products (VP's), Matrix Vector Products (MVP's) and Bit-decomposed Matrix Vector Products (BMVP). This allows us to present complexity comparisons independent of the specific differences in implementation libraries that might be used to experimentally compare these schemes. A table with comparisons for encryption, re-encryption and decryption operations of our schemes and PS-NTRUReEncrypt [NAL15] and IND-CPA-LWE [PWA$^+$16] schemes can be seen in Table 3.2. In this table, the short-hand notation of $(k+1)$ NTT + $2k$ VP for the cell corresponding to the re-encryption complexity for BV-PRE is used to indicate that the re-encryption operations requires $(k+1)$ calls to an NTT operation and $2k$ calls to a VP operation. As a matrix-vector product of $n \times n$ by n generally has a

Table 3.2 Theoretical Complexity Comparison of LWE-based IND-CPA-Secure PRE Schemes for Normalized Conditions

Scheme	Runtime/Latency		
	Enc	ReEnc	Dec
BV-PRE	1 NTT + 2 VP	(k+1) NTT + 2k VP	1 NTT + 1 VP
NTRU-ABD-PRE	1 NTT + 1 VP	(k+1) NTT + k VP	1 NTT + 1 VP
PS-NTRUReEncrypt [NAL15]	1 NTT + 1 VP	1 VP	1 NTT + 1 VP
IND-CPA-LWE [PWA$^+$16]	2 MVP	2 BMVP	1 MVP

Plaintext Modulus $p = 2$, Key Switching Window $r = 1$, Message Length $l = n$ (in the LWE Scheme).

higher complexity than Number Theoretic Transform (NTT), which is $O(n \log n)$, the runtime of BV-PRE is expected to be smaller for comparable values of ring dimension (lattice security parameter) and ciphertext modulus bit-width than IND-CPA-LWE.

In summary, the relation of our work to the prior work is as follows.

- Constructions of PRE based on bilinear maps are either single-hop unidirectional [AFGH06] or multi-hop bidirectional [CH07], whereas our scheme is multi-hop unidirectional. As noted in [CH07], constructing a multi-hop unidirectional PRE scheme using bilinear maps is an open problem. The practical execution times of our BV-PRE scheme (order of one millisecond), which supports dozens of hops without significant performance degradation, are comparable to those of bilinear map-based constructions.
- The BV-PRE scheme has a lower time and space complexity than existing IND-CPA-secure lattice-based PRE schemes.

3.1.2 Key-Switching and Automorphism

The notion of key-switching was introduced first by Brakerski and Vaikuntanathan [BV11b] as an optimization in the context of FHE. Specifically, a key-switching operation is invoked to transform a ciphertext encrypted under squared-secret key to a ciphertext encrypted under linear secret key.

Key-switching operations have also been shown to be very useful for performing algebraic operations on encrypted ciphertexts where data is embedded in plaintext slots [GHS12b, GHS12c]. Smart and Vercauteren [SV14] showed that in RLWE based

FHE schemes the plaintext space can be split into a vector of “plaintext slots” by an application of polynomial Chinese Remainder Theorem. Applying homomorphic operations to the ciphertext has a natural isomorphism with the embedded plaintext data. These operations are sometimes referred to as *SIMD* operations. One of the limitations of these SIMD operation is that we cannot move data across the slots of plaintext polynomial. This problem was resolved by Gentry, Halevi and Smart [GHS12b, GHS12c] with the construction of an Automorphism operation on ciphertext. By this tool of Automorphism transformation, the authors were able to carry out an implementation of AES on FHE encrypted data.

One of the problems associated with this transformation is that after an Automorphism operation, the ciphertext can only be decrypted by the transformed secret key. At this stage, one needs to apply key-switching techniques to bring the ciphertext into a form that can be decrypted by the original secret key. We present a key-switching techniques native to Ring-GSW FHE scheme and utilize it further to devise an Automorphism operation.

3.2 Proxy Re-Encryption

3.2.1 Workflow

The basic usage of Proxy Re-Encryption is shown in Figure 3.1. We assume a slightly more general model for PRE operations where a Policy Authority operates as a proxy for Alice to generate Alice’s public key and generate re-encryption keys to control who can decrypt information encrypted by Alice. It is also possible for Alice to be her own Policy Authority. The high-level operational flow of this key management infrastructure is as follows:

1. The policy authority generates public and secret key pairs for publishers such as Alice. These keys are designated as pk_A and sk_A, respectively. This key generation operation nominally occurs prior to deployment, or just as publishers need to send information to a PRE server.
2. Prior to deployment, the policy authority sends the publisher Alice public key pk_A. The publisher (and possibly multiple publishers) uses this public key to encrypt ciphertexts $c_A = \mathsf{Enc}(m, \mathsf{pk}_A)$ they send to the PRE server. The

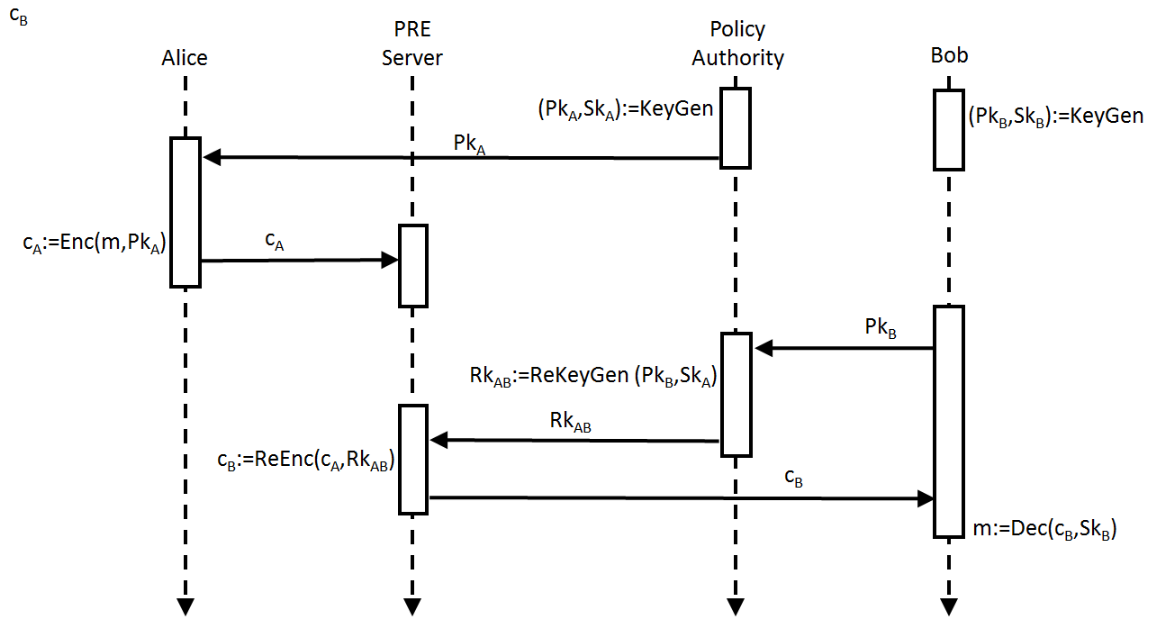

Figure 3.1 Proxy Re-Encryption functional key management and interaction workflow.

policy authority retains the secret key sk_A in case it needs to access information encrypted by the publisher.

3. When a subscriber needs to receive information from the PRE server, the subscriber Bob sends his public key (pk_B) to the policy authority.
4. The policy authority uses the publisher secret key (sk_A) and the subscriber public key (pk_B) to generate a re-encryption key (rk_{AB}). This re-key generation could occur prior to deployment or just as a subscriber needs to receive information.
5. The policy authority sends the re-encryption key to the PRE server.
6. The PRE server re-encrypts ciphertext so it can be decrypted by Bob.
7. Bob receives ciphertext and decrypts it using its secret key sk_B.

An important aspect of this key management infrastructure is that PRE pushes trust from the publisher to the policy authority and computational effort and bandwidth requirements to the PRE server. The policy authority determines who can share information and the PRE server uses information access policies to determine what subset of information from the publisher should be sent to the subscriber. The publisher and subscriber, who generally have the lowest computational capability in mobile applications, require the lowest computational effort and only need to maintain single keys, thus simplifying mobile deployments.

3.2.2 Syntax of Non-Interactive PRE

The workflow depicted in Figure 3.1 can only be supported by non-interactive PRE schemes, which require that re-encryption keys are generated using Bob's public key and Alice's private key. In this case, direct interaction between Bob and Alice can be avoided.

A non-interactive scheme includes algorithms ($\mathsf{ParamsGen}$, KeyGen, $\mathsf{ReKeyGen}$, Enc, ReEnc, Dec), described as follows:

- $\mathsf{ParamsGen}(\lambda)$: returns public parameters pp corresponding to security parameter λ;
- $\mathsf{KeyGen}(pp, \lambda)$: returns public-secret key pairs $(\mathsf{pk}, \mathsf{sk})$;
- $\mathsf{ReKeyGen}(pp, \mathsf{sk}_i, \mathsf{pk}_j)$: returns the re-encryption key $rk_{i \to j}$;
- $\mathsf{Enc}(pp, \mathsf{pk}, m)$: encrypts message m using pk and returns the ciphertext;
- $\mathsf{ReEnc}(pp, rk_{i \to j}, c_i)$: transforms a ciphertext c_i of party i into a ciphertext c_j that party j can decrypt;
- $\mathsf{Dec}(pp, \mathsf{sk}, c)$: recovers message m.

3.2.3 IND-CPA Security of PRE Schemes

Our security definition is a variant of the one postulated by Ateniese, Fu, Green and Hohenberger [AFGH06]. While Ateniese et al. [AFGH06] considered the notion of single-hop PRE schemes, both us and [PWA$^+$16] consider multi-hop PRE schemes. We remark that *the distinction between single-hop and multi-hop PRE is one of correctness, not security.* We state the definition below.

Definition 3.2.1 (IND-CPA Security)**.** *We consider the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ which proceeds in two phases.*
Phase 1:

- *$\mathcal{C}$ generates public parameters $pp \leftarrow$ $\mathsf{ParamsGen}(\lambda)$ and gives them to $\mathcal{A}$.*

- *Uncorrupted key generation: $\mathcal{C}$ generates $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(pp, \lambda)$ and gives pk to $\mathcal{A}$ upon request. $\mathcal{A}$ can request polynomially many such pk. Let Γ_H be the set of honest public keys (also called honest entities).*
- *Corrupted key generation: $\mathcal{C}$ generates $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(pp, \lambda)$ and gives $(\mathsf{pk}, \mathsf{sk})$ to $\mathcal{A}$ upon request. $\mathcal{A}$ can request polynomially many such $(\mathsf{pk}, \mathsf{sk})$. Let Γ_C be the set of corrupted public keys (also called corrupted entities).*

The adversary can issue a polynomial number of these queries, in arbitrary order.

Phase 2:

- *Re-encryption key generation: The adversary submits a directed acyclic* re-encryption graph *$G = (V, E)$ where the vertex set V is the set of all uncorrupted keys the adversary requested in Phase 1. $\mathcal{A}$ is given all the re-encryption keys $rk_{i \to j} \leftarrow \mathsf{ReKeyGen}(pp, \mathsf{sk}_i, \mathsf{pk}_j)$ where $(i, j) \in E$.*
  We remark that the adversary can generate by herself all re-encryption keys $rk_{i \to j}$ where $i \in \Gamma_C$, since she knows the secret keys sk_i. On the other hand, she is not allowed to obtain $rk_{i \to j}$ where $i \in \Gamma_H$ and $j \in \Gamma_C$ as that could allow her to decrypt the challenge ciphertext simply by performing a sequence of re-encryptions.
  We also note that this mechanism already allows the adversary to obtain re-encryptions of any ciphertext that she wishes. To obtain the re-encryption of an adversarially chosen string c_i from the public key pk_i to pk_j, she simply uses the re-encryption key $rk_{i \to j}$ that she obtained and runs the honest re-encryption procedure. Thus, there is no need to handle a separate re-encryption query.
- *Challenge: $\mathcal{A}$ submits (i^*, m_0, m_1). $\mathcal{C}$ chooses a random bit $b \in \{0, 1\}$, and then returns $c_{i^*} \leftarrow \mathsf{Enc}(pp, \mathsf{pk}_{i^*}, m_b)$. This is done only once, and it is required that $i^* \in \Gamma_H$.*

$\mathcal{A}$ finally outputs $b' \in \{0, 1\}$ as a guess of b. Define $\mathcal{A}$'s advantage as

$$\boldsymbol{Adv}_{\mathcal{A}}^{cpa}(\lambda) = \left| Pr[b' = b] - \frac{1}{2} \right|.$$

The PRE scheme is IND-CPA-secure if this advantage is negligible for all poly-time adversaries $\mathcal{A}$.

A few remarks about this definition are in order.

First, assume that the proxy obtains a (unidirectional) re-encryption key from user Alice to user Bob. The security definition above implies that even if the proxy (who has the re-encryption key) and Alice collude, they cannot break the security of Bob's encryption.

Second, note that if the proxy and Bob collude, they can decrypt Alice's ciphertexts, *by definition*. This is simply because the proxy can use the re-encryption key to turn Alice's ciphertext into Bob's ciphertext (for the same message) and then proceed to use Bob's secret key to decrypt. In essence, this means that the proxy and Bob together have a decryption circuit for Alice. (We do not attempt to define the notion of allowing this collusion to obtain only a "weak secret key" as in [AFGH06]).

Third, we note that stronger definitions are possible in that they can allow for chosen ciphertext decryption queries as considered in the work of [CH07]. One way to capture such attacks in the framework of our definition is to allow the adversary to request for re-encryption keys from uncorrupted public keys to corrupted ones, except that he cannot ask for a path of re-encryption keys from the challenge public key to a corrupted public key. We do not pursue this line of definitions in this work.

Fourth, we note that our IND-CPA definition does not explicitly handle re-encryption queries by the adversary, namely where the adversary queries with a tuple (i, j, c) and obtains the result of the re-encryption algorithm applied to $rk_{i \to j}$ and c. The reason we do not do this explicitly is that the adversary can simulate by herself the execution of such a query by first asking our re-encryption key generation oracle for $rk_{i \to j}$ and using it to re-encrypt c by herself. Since the pairs of keys for which the adversary is permitted to make re-encryption queries is the same as the ones between which she can obtain a re-encryption key, this omission is without loss of generality.

Finally, we note that the single-hop scheme of [AFGH06] is secure in a stronger IND-CPA sense than the above, since they can handle re-encryption graphs that contain directed cycles. On the other hand, the security proof of [PWA$^+$16] appears to only handle our acyclic IND-CPA definition.

3.3 PRE Cryptosystem with NTRU Key Generation and RLWE Key Switching (NTRU-ABD-PRE)

The first PRE scheme proposed in this chapter is based on the NTRU encryption scheme [HPS98] with RLWE modifications [SS11]. The NTRU immunity constraint against subfield lattice attacks is used to set the distribution parameter for NTRU key generation [ABD16]. The subfield lattice attacks allow the adversary to reduce the ring dimension of the affected cryptosystems for certain parameter regimes and solve the Shortest Vector Problem for n = 512 or lower [ABD16, CJL16].

3.3.1 NTRU-RLWE Encryption Scheme

The scheme is parameterized using the following quantities:

- security parameter (root Hermite factor) δ [CN11],
- ciphertext modulus q,
- ring dimension n,
- B_k-bounded discrete Gaussian key distribution χ_k over the polynomial ring $R = \mathbb{Z}[n]/\langle x^n + 1\rangle$ with distribution parameter σ_k (subscript k refers to *key* distribution),
- B_e-bounded discrete Gaussian error distribution χ_e with distribution parameter σ_e (subscript e refers to *error* distribution),
- empirically selected assurance measure α to minimize the number of bits needed to represent q (introduced for better performance).

In this work, we focus on the case of power-of-2 cyclotomic polynomials where n is a power of 2 for multiple reasons. For one, the case of power-of-2 cyclotomics leads to much simpler and more efficient implementations of number theoretic transforms used to support ring operations. Further, the computational hardness underlying security assumptions for these cases is better understood as compared to the case of arbitrary cyclotomics [SS11].

We support a plaintext space of $\mathcal{M} = \{0, 1, \ldots, p-1\}^n$, where $p \geq 2$ is the plaintext modulus. All operations on ciphertexts are performed in the ring

$R_q \equiv R/qR$. Each coefficient in the polynomials is represented in the range $\{-\lfloor \frac{q}{2} \rfloor, ..., \lfloor \frac{q}{2} \rfloor\}$.

The scheme includes the following operations:

- $\mathsf{ParamsGen}(\lambda)$: Choose positive integers q and n. Return $pp = (q, n)$.
- $\mathsf{KeyGen}(pp, \lambda)$: Sample polynomials $f', g \leftarrow \chi_k$ and set $f := pf' + 1$ to satisfy $f \equiv 1 \,(\text{mod} p)$. If f has no modular multiplicative inverse in R_q, then re-sample f'. Set the public key pk and private key sk:
$$\mathsf{sk} := f \in R,\ \mathsf{pk} := h = pg\, f^{-1} \in R_q.$$
- $\mathsf{Enc}\,(pp, \mathsf{pk} = h, m \in \mathcal{M})$: Sample polynomials $s, e \leftarrow \chi_e$. Compute the ciphertext:
$$c := hs + pe + m \in R_q.$$
- $\mathsf{Dec}\,(pp, \mathsf{sk} = f,\ c)$: Compute the ciphertext error $b := f \cdot c \in R_q$. Output $m' := b \,(\text{mod} p)$.

The scheme is correct as long as there is no wrap-around modulo q. Indeed,

$$b = f \cdot c = f\,(h\,s + pe + m) = pgs + pfe + fm$$

and if the value of b does not wrap around modulo q, then

$$m' = pgs + pfe + fm = fm = m \ (\text{mod} p).$$

To derive the correctness constraint for decryption, we note that the coefficients in g, s cannot exceed B_k as they are generated by a B_k-bounded discrete Gaussian distribution χ_k. Analogously, the coefficients in f cannot exceed $pB_k + 1$ and coefficients in e cannot exceed B_e, yielding

$$\|b\|_\infty = \|pgs + pfe + fm\|_\infty < 2np^2 B_k B_e.$$

Here, we assume that $B_k, B_e \gg 1$, which is true for all practical scenarios of this scheme. To guarantee correct decryption of the ciphertext, coefficients in b should not exceed $q/2$ leading to the following correctness constraint:

$$q > 4np^2 B_k B_e. \tag{3.1}$$

When $\sigma > \omega(\log n)$, the bound B_i can be expressed as $\sigma_i\sqrt{n}$, where $i \in \{k, e\}$ and 2^{-n+1} is the probability that a coefficient generated using discrete Gaussian distribution exceeds the bound B_i [MR07, LTV13]. To obtain less conservative estimated bounds for noise, we introduce an assurance measure $\alpha < n$ corresponding to the probability of $2^{-\alpha+1}$ that a coefficient of a discrete Gaussian polynomial exceeds the bound B_i (the choice of a specific practical value of α is validated using an empirical analysis of decryption correctness for a large sample of plaintexts). In this case, the bounds B_k and B_e can be expressed as $\sigma_k\sqrt{\alpha}$ and $\sigma_e\sqrt{\alpha}$, respectively.

The constraint (3.1) was derived for the worst-case scenario where both B_i-bounded polynomials may simultaneously take the upper (or lower) bound values for all coefficients in the polynomials of dimension n. As the coefficients of polynomials generated by the discrete Gaussian distribution are taken from a relatively large sample of size n (where n is at least 512), we can apply the Central Limit Theorem (CLT) to derive a lower (average-case) bound for q.

If we examine a product of two B_k-bounded polynomials g and s in the ring R_q, we observe that each coefficient in g is multiplied by the mean of coefficients in s (followed by modulo reduction). This implies that each coefficient in $g \cdot s$ is bounded by $n\sigma_k\sigma_{kn}\alpha$, where σ_{kn} is the standard deviation of the mean expressed as $\sigma_{kn} = \sigma_k n^{-1/2}$. After simplification, the bound for $g \cdot s$ can be expressed as $\sqrt{n}\sigma_k^2\alpha$ instead of the original worst-case bound $n\sigma_k^2\alpha$. Therefore, this technique allows one to replace each occurrence of n (corresponding to a polynomial multiplication) with $\sqrt{n}$. Applying this logic to the worst-case constraint for the encryption scheme, we obtain the following average-case correctness constraint:

$$q > 4\sqrt{n}p^2 B_k B_e. \tag{3.2}$$

It should be noted that the effective probability associated with assurance measure α, i.e., $2^{-\alpha+1}$, gets significantly reduced for a product of two discrete Gaussians. This further justifies the use of an assurance measure much smaller than n.

3.3.2 Security of NTRU-RLWE Encryption Scheme

This general NTRU-RLWE encryption scheme can be instantiated for three different ranges of distribution parameter σ_k, giving us security from different computational assumptions [SS11, LTV13, ABD16]. The scheme can be proven secure based on the NTRU and RLWE assumptions for these different parameter regimes. Here, the NTRU problem is to distinguish between the following two distributions: a polynomial f/g with f and g sampled from distribution χ_k (assuming g is invertible over R_q) and a polynomial h sampled uniformly at random over R_q.

The first variant [SS11] is based on the RLWE assumption. The public key (polynomial f/g) distribution was shown to be *statistically indistinguishable* from uniform random distribution for $\Phi_m(x) = x^n + 1$ when $\sigma_k = \omega\left(q^{1/2}\right)$ [SS11]. This allowed the authors to rely solely on the RLWE assumption to prove semantic security of the encryption scheme. This logic was applied to show that the Stehlé-Steinfeld scheme defined by operations $\mathsf{KeyGen}, \mathsf{Enc}$, and Dec and constraint $\sigma_k = \omega\left(q^{1/2}\right)$ is IND-CPA secure [SS11].

Though based solely on the RLWE assumption, the original Stehlé-Steinfeld scheme is impractical for proxy re-encryption or any homomorphic encryption scheme requiring at least one multiplication of two polynomials generated from the distribution χ_k. In this case, the correctness inequality for q would never hold as we would have $B_k^2 \propto \sigma_k^2 = \omega(q)$ on the right hand side of the expression, i.e., $q > \omega(q)$.

For practical reasons, the constraint $\sigma_k = \omega\left(q^{1/2}\right)$ was suggested to be replaced with a smaller value corresponding to the error distribution χ_e by arguing that the resulting Decisional Small Polynomial Ratio (DSPR) problem is secure against all known practical attacks [LTV13]. This assumption was recently invalidated for some parameter ranges by two subfield lattice attacks [ABD16, CJL16], which are able to reduce the ring dimension of the affected cryptosystems and solve the Shortest Vector Problem for n = 512 or lower.

Albrecht et al. [ABD16] proposed a new practical constraint for σ_k based on the immunity of NTRU to subfield lattice attacks, conjecturing that the Stehlé-Steinfeld proof may be extended to this case:

$$\sigma_k > \left(\frac{2q}{n\pi e}\right)^{1/4}. \tag{3.3}$$

Our proxy re-encryption scheme, referred to as NTRU-ABD-PRE, uses this constraint. In contrast to the original Stehlé-Steinfeld scheme, this scheme supports ReKeyGen, ReEnc, and homomorphic indexing and multiplication operations.

To meet the RLWE security requirements of the encryption scheme, we use the inequality derived in [GHS12c], namely,

$$n \geq \frac{\log_2(q/\sigma_e)}{4\log_2(\delta)}. \tag{3.4}$$

3.3.3 Single-Hop Re-Encryption Scheme

The PRE scheme introduces a new configurable parameter, key switching window r, and two new operations (in addition to ParamsGen, KeyGen, Enc, and Dec):

- $\mathsf{ReKeyGen}\,(pp, \mathsf{sk} = f, \mathsf{pk} = h^*)$: For every $i = 0, 1, .., \lfloor \log_2 (q)/r \rfloor$, sample polynomials s_i and e_i, and compute γ_i

$$\gamma_i = h^* s_i + pe_i + f \cdot (2^r)^i \in R_q, \; rk := \gamma = \left(\gamma_0, \gamma_1, ..., \gamma_{\lfloor \log_2(q)/r \rfloor}\right).$$

- $\mathsf{ReEnc}\,(pp, rk = \gamma, c)$: Compute the ciphertext

$$c' = \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} (c_i \cdot \gamma_i),$$

where $c_i := \{h \cdot s + pe + m\}_i$, $c = \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} c_i \cdot (2^r)^i$.

Here, $rk = \gamma$ is the re-encryption key. The key switching window r is used to decompose the ciphertext coefficients into base-2^r components c_i, thus substantially reducing the noise growth. Each c_i is represented as a polynomial in R_q with coefficients in the range between 0 and $2^r - 1$.

The PRE scheme is devised using a generalized version of the RLWE key switching (bit decomposition) technique originally introduced for reducing the ciphertext error in homomorphic encryption [BV11a, LTV13]. Consider a new set of keys: private key f^* and public key $h^* = pg^* (f^*)^{-1}$. The goal is to re-encrypt the ciphertext c using the public key h^* without decrypting the data.

To this end, we introduce a set of elements γ_i as

$$\gamma_i = h^* s_i + pe_i + f \cdot (2^r)^i \in R_q,$$

where $i = 0, 1, .., \lfloor \log_2 (q)/r \rfloor$. This set of elements, referred to as the re-encryption key, represents an encryption of all powers-of-2^r multiples of the secret key f under the public key h^*. The key switching window was set to unity in [BV11a, LTV13]. In this study, we consider a range of key switching window values (powers of 2) to achieve a faster implementation of re-encryption. The vector $\gamma = \left(\gamma_0, \gamma_1, ..., \gamma_{\lfloor \log_2(q)/r \rfloor}\right)$ is public.

The ciphertext c is computed using the public key h:

$$c := hs + pe + m \in R_q.$$

For each window i of length r (in bits), we introduce $c_i := \{h \cdot s + pe + m\}_i$, and the ciphertext c can then be represented as

$$c = \sum_i c_i \cdot (2^r)^i.$$

The polynomial c' computed as

$$c' = \sum_i c_i \cdot \gamma_i$$

can be shown to represent an encryption of m under the new public key h^*.

Indeed,

$$f^* \cdot c' = f^* \cdot (\sum_i c_i \cdot \gamma_i) = \sum_i c_i \cdot (f^* \cdot \gamma_i) = p \sum_i c_i \cdot E_i + \sum_i c_i \, f^* f \cdot (2^r)^i = p \sum_i c_i \cdot E_i + f^* f \, c,$$

where $E_i = g^* s_i + f^* e_i$.

It can be seen that

$$f^* \cdot c' = f^* f \, c = m \, (\mathrm{mod} p)\,,$$

i.e., the decryption is correct, if the ciphertext error $f^* \cdot c'$ is not too large to wrap around q.

Considering that $\|c_i\|_\infty \le 2^r - 1$, $\|E_i\|_\infty \le nB_e\,(B_k + pB_k + 1)$, and

$$\|f^* f \, c\|_\infty \le n^2\,(pB_k + 1)\,\{pB_e\,(B_k + pB_k + 1) + (pB_k + 1)\,(p - 1)\}\,,$$

we have

$$\begin{aligned}\left\|f^{*}c'\right\|_{\infty} &\leq pn\left(\left\lfloor \log_2\left(q\right)/r\right\rfloor+1\right)\left(2^{r}-1\right)\left\{nB_e\left(B_k+pB_k+1\right)\right\}\\ &+n^2\left(pB_k+1\right)\left(pB_e\left(B_k+pB_k+1\right)+\left(pB_k+1\right)\left(p-1\right)\right\}\\ &< 2p^2n^2B_eB_k\left\{\left(2^{r}-1\right)\left(\left\lfloor \log_2\left(q\right)/r\right\rfloor+1\right)+pB_k\right\}.\end{aligned}$$

To guarantee correct decryption of the ciphertext, $f^{*}\cdot c'$ should not exceed $q/2$ leading to the following worst-case correctness constraint:

$$q > 4p^2n^2B_kB_e\left\{pB_k+\left(2^{r}-1\right)\left(\left\lfloor \log_2\left(q\right)/r\right\rfloor+1\right)\right\}.$$

Similar to the case of the encryption scheme, we can apply CLT to obtain the following average-case correctness constraint for the PRE scheme:

$$q > 4p^2nB_kB_e\left\{pB_k+\left(2^{r}-1\right)\left(\left\lfloor \log_2\left(q\right)/r\right\rfloor+1\right)\right\}. \tag{3.5}$$

3.3.4 Extension to Multiple Re-Encryption Hops

The presented re-encryption scheme can be generalized to support multiple re-encryption hops. Consider a new set of keys: private key f^{**} and public key $h^{**}=pg^{**}\left(f^{**}\right)^{-1}$. The goal is to re-encrypt the re-encrypted ciphertext c' devised in Section 3.3.3 using the public key h^{**} without decrypting the data.

Analogously to the case of single re-encryption, we introduce a set of elements

$$\gamma'_i = h^{**}s'_i+pe'_i+\ f^{*}\cdot\left(2^{r}\right)^{i}\in R_q,$$

where $i=0,1,..,\left\lfloor \log_2\left(q\right)/r\right\rfloor$. The vector $\gamma'=\left(\gamma'_0,\gamma'_1,...,\gamma'_{\lfloor \log_2(q)/r\rfloor}\right)$ is the re-encryption key to transform from $\{f^{*},h^{*}\}$ to $\{f^{**},h^{**}\}$.

The polynomial c'' computed as

$$c'' = \sum_i c'_i \cdot \gamma'_i$$

can be shown to represent an encryption of m under the new public key h^{**} as long as there is no wrap-around modulo q.

Indeed,

$$f^{**} \cdot c'' = \sum_i {c'}_i \cdot (f^{**} \cdot \gamma'_i) = p \sum_i {c'}_i \cdot E'_i + \sum_i {c'}_i \, f^{**} f^* \cdot \, (2^r)^i = p \sum_i c'_i \cdot E'_i + f^{**} f^* \, c',$$

where $E'_i = g^{**} s'_i + f^{**} e'_i$.

It can be easily shown that

$$f^{**} \cdot c'' = f^{**} f^* \cdot c' = f^{**} f^* f \cdot c = m \, (\mathrm{mod} p) \, ,$$

i.e., the decryption is correct, if the ciphertext error $f^{**} \cdot c''$ is not too large to wrap around q.

Applying the same procedure as for the first re-encryption, the correctness inequality after two re-encryptions can be expressed as

$$q > 4p^2 n B_k B_e \left\{ (2^r - 1) \, \left(\lfloor \log_2 (q) \, / r \rfloor + 1 \right) \right\}$$

$$+ \, 4p^2 n B_k B_e n^{0.5} \left(pB_k + 1 \right) \left\{ pB_k + (2^r - 1) \, \left(\lfloor \log_2 (q) \, / r \rfloor + 1 \right) \right\} .$$

Considering that the first summand is at least by a factor of $n^{0.5} \left(pB_k + 1 \right)$ (this value is larger than 2^{10} for all practical parameter ranges) smaller than the second summand, the correctness constraint can be rewritten as

$$q > n^{0.5} \left(pB_k + 1\right) \cdot 4p^2 nB_k B_e \left\{pB_k + \left(2^r - 1\right) \left(\left\lfloor \log_2\left(q\right)/r \right\rfloor + 1\right)\right\}, \tag{3.6}$$

which implies that the second re-encryption increases the lower bound for q by a factor of $n^{0.5} \left(pB_k + 1\right)$.

After two re-encryption hops, the expression on the right hand side of (3.6) is $\Omega(B_k^3)$ and $B_k^3 \propto \sigma_k^3 \propto q^{3/4+\epsilon}$, where $\epsilon > 0$. This implies that NTRU-ABD-PRE does not support more than two re-encryption hops because in the case of three hops the right-hand side expression of (3.6) will reach $q^{1+\epsilon}$.

The effective value of assurance measure α corresponding to a given probability can be decreased for each extra step of re-encryption as long as the empirical evaluation of decryption correctness is performed.

3.3.5 IND-CPA Security

We will show that the NTRU-ABD-PRE scheme is IND-CPA secure in the sense of Definition 3.2.1.

As noted in Section 3.3.2, the NTRU-ABD-PRE scheme is based on the NTRU and RLWE assumptions. Specifically, we use a variant of the NTRU assumption formulated in [ABD16] to achieve the immunity of NTRU to subfield lattice attacks. We refer to this variant as NTRU-ABD with the formal definition as follows:

Definition 3.3.1. *The NTRU-ABD$_{n,q,\chi_k}$ problem is to distinguish between the following two distributions over ring $R_q = \mathbb{Z}_q[x]/\left\langle x^n + 1\right\rangle$: a polynomial g/f with g and f sampled from distribution χ_k with $\sigma_k > \Theta(q^{1/4})$ (assuming g is invertible over R_q) and a polynomial h sampled uniformly at random over R_q.*

For the RLWE assumption, we use the Hermite normal form [LTV13], which is defined as follows:

Definition 3.3.2. *For all $\lambda \in \mathbb{N}$, let $\phi(x) = \phi_\lambda(x) \in \mathbb{Z}[x]$ be a cyclotomic polynomial of degree $n = n(\lambda)$, let $q = q(\lambda) \in \mathbb{Z}$ be a prime number, let the ring $R \doteq \mathbb{Z}[x]/\langle \phi(x) \rangle$ and $R_q \doteq R/qR$, and let χ_e denote an error distribution over the ring R.*

The decision ring-LWE assumption $RLWE_{\phi,q,\chi_e}$ states that for any $\ell = poly(\lambda)$,

$$\{(a_i, a_i \cdot s + e_i)\}_{i\in[\ell]} \overset{c}{\approx} \{(a_i, u_i)\}_{i\in[\ell]},$$

where s and "error polynomials" e_i are sampled from the noise distribution χ_e, and a_i and u_i are uniformly random in R_q.

Albrecht *et al.* [ABD16] conjectured that the Stehlé-Steinfeld IND-CPA proof [SS11], which was provided for $\sigma_k = \omega(q^{1/2})$, may be extended to this case assuming only RLWE. However, as it stands, the security of this scheme is based on RLWE as well as the NTRU-ABD assumption described above. We will assume that the NTRU-ABD assumption is stronger than RLWE and set the key-sizes accordingly.

We showed that the NTRU-ABD PRE scheme maintains *correctness* for only two hops, in the sense that once a ciphertext goes through more than two hops, it cannot be decrypted to the correct message any more. It is important to note that the "two-hopness" is a limitation on the correctness of the scheme, not its security. In particular, we will show below that the scheme is IND-CPA-secure in the sense of Definition 3.2.1 which is a notion of security for general, multi-hop PRE schemes.

Our proof for NTRU-ABD-PRE is similar to that of the IND-CPA-secure LWE scheme proposed in [PWA+16].

Theorem 1 (IND-CPA security of NTRU-ABD-PRE)**.** *Under the NTRU-ABD$_{n,q,\chi_k}$ and RLWE$_{\phi,q,\chi_e}$ assumptions, NTRU-ABD-PRE is IND-CPA-secure. Specifically, for a poly-time adversary $\mathcal{A}$, there exists a poly-time distinguisher $\mathcal{D}$ such that*

$$\boldsymbol{Adv}_{\mathcal{A}}^{cpa}(\lambda) \leq (\rho \cdot (Q_{rk} + Q_{re}) + N + 1) \cdot \mathsf{max}(\boldsymbol{Adv}_{\mathcal{D}}^{NTRU\text{-}ABD_{n,q,\chi_k}}(\lambda), \boldsymbol{Adv}_{\mathcal{D}}^{RLWE_{\phi,q,\chi_e}}(\lambda))$$

where Q_{rk} and Q_{re} are the numbers of re-encryption key queries and re-encryption queries, respectively; N is the number of honest entities; λ is the security parameter; $\rho := 1 + \lfloor \log_2 q/r \rfloor$.

Proof. We show that the NTRU-ABD-PRE scheme is IND-CPA-secure through a sequence of games.

Let Game$_0$ be an initial game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ with their interactions governed by Definition 3.2.1. For notational convenience, let us consider the case when $\Gamma_H = \{1, \ldots, N\}$ and $\Gamma_C = \{N+1, \ldots, M\}$ for some polynomial M. Furthermore, without loss of generality, let $1, 2, \ldots, N$ be the topological order dictated by the re-encryption graph, starting from the sinks to the sources, namely there are no edges from i to k if $i < k$. In more detail:

- The i-th key pair is defined as $\mathsf{sk}_i := f_i \in R$, and $\mathsf{pk}_i := h_i = pg_i f_i^{-1} \in R_q$, where $f_i = pf'_i + 1$ and $f'_i, g_i \leftarrow \chi_k$.
- The re-encryption key from party i to party k is written as

$$rk_{i \to k} := (h_k \cdot s_{iku} + pe_{iku} + f_k \cdot (2^r)^u)_{u \in \{0,1,\ldots \lfloor \log_2(q)/r \rfloor\}},$$

  where s_{iku}, e_{iku} are generated by party i.
- The challenge ciphertext of message m_b related to party i^* is:

$$c^* := h^* \cdot s^* + pe^* + m_b \in R_q,$$

  where $b \in \{0,1\}$ is the challenge bit, $s^*, e^* \leftarrow \chi_e$, and h^* is the challenge public key.

Let Game$_k$ $(1 \le k \le N)$ be defined by considering the honest party $k \in \Gamma_H$. Game$_k$ is identical to Game$_{k-1}$ except for the following changes:

- When generating the k-th key pair, $h_k = p \cdot r_k^*$, where r_k^* is a randomly generated ring element rather than an NTRU-ABD sample.
- When answering the re-encryption key query (i, k): First, note that $i > k$ because of the topological ordering. The re-encryption key is expressed as

$$rk_{i \to k} := (p \cdot \gamma_{iku})_{u \in \{0,1,\ldots \lfloor \log_2(q)/r \rfloor\}},$$

  where γ_{iku} is freshly random.

Each Game$_k$ is computationally indistinguishable from Game$_{k-1}$ because of the NTRU-ABD and RLWE assumptions. First, $k \in \Gamma_H$ and therefore, there is no re-encryption "edge" from user k to any user in Γ_C. Additionally, and crucially, all the re-encryption keys (k, i) have already been replaced with uniformly random ring elements in the prior games. Consequently, the secret key f_k is used only in the form of fresh NTRU-ABD samples in its public key and in the form of fresh RLWE samples in the re-encryption keys (the latter assumes that the public key is indistinguishable from a random sample based on the the NTRU-ABD assumption). Thus, all these can be replaced by uniformly random ring elements by invoking the NTRU-ABD and RLWE assumptions. The security loss is proportional to the number of re-encryption key and re-encryption queries that user k was part of (an additional multiplicative factor $1 + \lfloor \log_2 q/r \rfloor$ is incurred in the security loss as each re-encryption key contains that many RLWE samples).

Game$_{\text{final}}$ is the same as Game$_N$ except for the challenge ciphertext that is expressed as

$$c^* := p \cdot r^* + m_b \in R_q,$$

where r^* is a freshly random ring element in R_q. This is computationally indistinguishable from Game$_N$ by the RLWE assumption (assuming that the public key is indistinguishable from a random sample based on the the NTRU-ABD assumption).

The last change guarantees that the challenge bit b is information-theoretically hidden from $\mathcal{A}$, and therefore, the advantage of the adversary in Game$_1$ is 0.

Putting all this together, we see that

$$\mathbf{Adv}_{\mathcal{A}}^{cpa}(\lambda) \leq (\rho \cdot (Q_{rk} + Q_{re}) + N + 1) \cdot \mathsf{max}(\mathbf{Adv}_{\mathcal{D}}^{NTRU\text{-}ABD_{n,q,\chi_k}}(\lambda), \mathbf{Adv}_{\mathcal{D}}^{RLWE_{\phi,q,\chi_e}}(\lambda))$$

where $\rho := 1 + \lfloor \log_2 q/r \rfloor$. This finishes our proof. □

3.4 PRE Cryptosystem with RLWE Key Generation and Key Switching (BV-PRE)

The second PRE scheme proposed in this chapter, BV-PRE, is based on the Brakerski-Vaikuntanathan (BV) homomorphic encryption scheme [BV11b]. The BV-PRE scheme relies only on the RLWE security assumption.

3.4.1 The Encryption Scheme

The basic encryption scheme comes from the work of Lyubashevsky, Peikert and Regev [LPR10, LPR13] and Micciancio [Mic10]. The scheme is parameterized using the following quantities:

- security parameter (root Hermite factor) δ [CN11],
- ciphertext modulus q,
- ring dimension n,
- B_e-bounded discrete Gaussian error distribution χ_e with distribution parameter σ_e,
- empirically selected assurance measure α to minimize the number of bits needed to represent q (introduced for better performance).

As in the case of NTRU-ABD-PRE, we work with the polynomial ring $R_q = \mathbb{Z}_q[n]/\langle x^n+1\rangle$ and use a plaintext space of $\mathcal{M} = \{0,1,\ldots,p-1\}^n$, where $p \geq 2$ is the plaintext modulus. Each coefficient in the polynomials is represented in the range $\{-\lfloor \frac{q}{2} \rfloor, ..., \lfloor \frac{q}{2} \rfloor\}$. We also introduce $\mathcal{U}_q$ as a discrete uniform random distribution over R_q.

The scheme includes the following operations:

- $\mathsf{ParamsGen}(\lambda)$: Choose positive integers q and n. Return $pp = (q, n)$.
- $\mathsf{KeyGen}(pp, \lambda)$: Sample polynomials $a \leftarrow \mathcal{U}_q$ and $s, e \leftarrow \chi_e$. Compute $b := a \cdot s + pe \in R_q$. Set the public key pk and private key sk:

$$\mathsf{sk} := s \in R,\ \mathsf{pk} := (a, b) \in R_q^2.$$

- $\mathsf{Enc}\left(pp, \mathsf{pk} = (a, b), m \in \mathcal{M}\right)$: Sample polynomials $v, e_0, e_1 \leftarrow \chi_e$. Compute the ciphertext $c = (c_0, c_1) \in R_q^2$:

$$c_0 := b \cdot v + pe_0 + m \in R_q,\ c_1 := a \cdot v + pe_1 \in R_q.$$

- $\mathsf{Dec}\left(pp, \mathsf{sk} = s,\ c\right)$: Compute the ciphertext error $t := c_0 - s \cdot c_1 \in R_q$. Output $m' := t\,(\mathrm{mod} p)$.

The scheme is correct as long as there is no wrap-around modulo q. Indeed,

$$t = b \cdot v + pe_0 + m - s \cdot (a \cdot v + pe_1) = (a \cdot s + pe) \cdot v + pe_0 + m - s \cdot (a \cdot v + pe_1)$$
$$= m + p\left(e \cdot v + e_0 - s \cdot e_1\right).$$

where all computations are done mod q. If the value of t does not wrap around modulo q, then

$$m' = m + p\left(e \cdot v + e_0 - s \cdot e_1\right) = m\ (\mathrm{mod} p).$$

To derive the correctness constraint for decryption, we note that the coefficients in s, v, e, e_0, e_1 cannot exceed B_e as they are generated by a B_e-bounded discrete Gaussian distribution χ_e. Applying the same procedure as for the NTRU-RLWE scheme followed by CLT, we obtain

$$\|t\|_\infty < 3\sqrt{n} p B_e^2.$$

Here, we assume that $B_e > 1$. To guarantee correct decryption of the ciphertext, coefficients in t should not exceed $q/2$, leading to the following correctness constraint:

$$q > 6\sqrt{n} p B_e^2. \tag{3.7}$$

3.4.2 Proxy Re-Encryption Scheme

The PRE scheme introduces three new operations (in addition to ParamsGen, KeyGen, Enc, and Dec) in contrast to two needed for the PRE functionality in NTRU-ABD-PRE. In BV-PRE, the evaluation key generation is performed in two separate stages: Preprocess and ReKeyGen. First, the owner of key s^* generates a set of "public" keys $(\beta_i, \beta_i \cdot s^* + pe_i)$ and then sends these keys to the policy authority, as displayed in Figure 3.1. After that, the proxy authority computes γ_i to generate the complete re-encryption key. This allows one to apply the same non-interactive PRE workflow as discussed in Section 3.2.

- Preprocess $(pp, \lambda, \mathsf{sk}^* = s^*)$: For every $i = 0, 1, .., \lfloor \log_2(q)/r \rfloor$, where r is the key switching window, sample polynomials $\beta_i \leftarrow \mathcal{U}_q$ and $e_i \leftarrow \chi_e$ and compute

$$\theta_i^* = \beta_i \cdot s^* + pe_i \in R_q,$$

$$\mathsf{pk} := (\beta_i, \theta_i^*)_{i \in \{0,1,...\lfloor \log_2(q)/r \rfloor\}}\,.$$

- ReKeyGen $\left(pp, \mathsf{sk} = s, \mathsf{pk} = (\beta_i, \theta_i^*)_{i \in \{0,1,...\lfloor \log_2(q)/r \rfloor\}}\right)$:
  For every $i = 0, 1, .., \lfloor \log_2(q)/r \rfloor$, compute γ_i and store them in re-encryption key rk

$$\gamma_i = \theta_i^* - s \cdot (2^r)^i \in R_q, rk := (\beta_i, \gamma_i)_{i \in \{0,1,...\lfloor \log_2(q)/r \rfloor\}}\,.$$

- ReEnc $\left(pp, rk = (\beta_i, \gamma_i)_{i \in \{0,1,...\lfloor \log_2(q)/r \rfloor\}}, c\right)$: Compute the ciphertext $c' = (c'_0, c'_1)$ using the 2^r-base decomposition of ciphertext element c_1 of original ciphertext $c = (c_0, c_1)$

$$c'_0 = c_0 + \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} (c_1^{(i)} \cdot \gamma_i),\ c'_1 = \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} (c_1^{(i)} \cdot \beta_i),$$

  where $c_1^{(i)} := \{a \cdot v + pe\}_i$ is the i_{th} "digit" of the base-2^r representation of c_1 and

$$c_1 = \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} c_1^{(i)} \cdot (2^r)^i.$$

The ciphertext $c' = (c'_0, c'_1)$ can be shown to represent an encryption of m under the new key s^*. Indeed,

$$\begin{aligned} c'_0 - s^* c'_1 &= c_0 + \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} (c_1^{(i)} \cdot \gamma_i) - s^* \cdot \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} (c_1^{(i)} \cdot \beta_i) \\ &= c_0 + \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} \left(c_1^{(i)} \cdot \left\{ \beta_i \cdot s^* + p e_i - s \cdot (2^r)^i \right\} \right) - s^* \cdot \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} (c_1^{(i)} \cdot \beta_i) \\ &= c_0 - s \cdot c_1 + p E_i, \end{aligned}$$

where $E_i = \sum_{i=0}^{\lfloor \log_2(q)/r \rfloor} \left(c_1^{(i)} \cdot e_i \right)$.

It can be easily seen that

$$c_0 - s \cdot c_1 + p E_i \,(\mathrm{mod} p) = c_0 - s \cdot c_1 \,(\mathrm{mod} p) = m \,(\mathrm{mod} p) .$$

The above analysis implies that the ciphertext noise term grows only by a small additive factor $p \left\| E_i \right\|_\infty$ after each re-encryption. $\left\| E_i \right\|_\infty$ can be expressed as

$$\left\| E_i \right\|_\infty < \sqrt{n} B_e \left(2^r - 1 \right) \left(\lfloor \log_2 (q) / r \rfloor + 1 \right) .$$

Therefore, the correctness constraint for d re-encryption hops can be written as

$$q > 2\sqrt{n} p B_e \left\{ 3B_e + d \cdot \left(2^r - 1 \right) \left(\lfloor \log_2 (q) / r \rfloor + 1 \right) \right\} . \tag{3.8}$$

3.4.3 IND-CPA Security

We will show that the BV-PRE scheme is IND-CPA secure in the sense of Definition 3.2.1.

Theorem 2 (IND-CPA security of BV-PRE)**.** *Under the $RLWE_{\phi,q,\chi_e}$ assumption, BV-PRE is IND-CPA-secure. Specifically, for a poly-time adversary $\mathcal{A}$, there exists* *a poly-time distinguisher $\mathcal{D}$ such that*

$$\boldsymbol{Adv}_{\mathcal{A}}^{cpa}(\lambda) \leq (\rho \cdot (Q_{rk} + Q_{re}) + N + 1) \cdot \boldsymbol{Adv}_{\mathcal{D}}^{RLWE_{\phi,q,\chi_e}}(\lambda)$$

where Q_{rk} and Q_{re} are the numbers of re-encryption key queries and re-encryption queries, respectively; N is the number of honest entities; λ is the security parameter; ϕ is the cyclotomic polynomial defining the ring $R_q = \mathbb{Z}_q[x]/\langle\phi\rangle$ and $\rho := 1 + \lfloor \log_2 q/r \rfloor$.

Proof. We show that the BV-PRE scheme is IND-CPA-secure through a sequence of games.

Let Game$_0$ be an initial game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ with their interactions governed by Definition 3.2.1. For notational convenience, let us consider the case when $\Gamma_H = \{1, \ldots, N\}$ and $\Gamma_C = \{N+1, \ldots, M\}$ for some polynomial M. Furthermore, without loss of generality, let $1, 2, \ldots, N$ be the topological order dictated by the re-encryption graph, starting from the sinks to the sources, namely there are no edges from i to k if $i < k$. In more detail:

- The i-th key pair is defined as $\mathsf{sk} := s_i \in R$, and $\mathsf{pk} := (a_i, a_i \cdot s_i + pe_i) \in R_q^2$, where $s_i, e_i \leftarrow \chi_e$.
- The re-encryption key from party i to party k is written as

$$rk_{i \to k} := (\beta_{iku}, \beta_{iku} \cdot s_k + pe_{iku} - s_i \cdot (2^r)^u)_{u \in \{0,1,\ldots \lfloor \log_2(q)/r \rfloor\}},$$

  where β_{iku}, e_{iku} are generated by party k.
- The challenge ciphertext related to party i^* is $c^* = (c_0^*, c_1^*) \in R_q^2$:

$$c_0^* := b^* \cdot v^* + pe_0^* + m_b \in R_q,\ c_1^* := a^* \cdot v^* + pe_1^* \in R_q,$$

  where $b \in \{0,1\}$ is the challenge bit, $v^*, e_0^*, e_1^* \leftarrow \chi_e$, and (a^*, b^*) is the challenge public key.

Let Game$_k$, $i \in \{1 \leq k \leq N\}$, be defined by considering the honest party $k \in \Gamma_H$. Game$_k$ is identical to Game$_{k-1}$ except for the following changes:

- When generating the k-th key pair, b_k is a randomly generated ring element rather than a RLWE sample.

- When answering the re-encryption key query (i, k): First, note that $i > k$ because of the topological ordering. The re-encryption key is expressed as

$$rk_{i \to k} := (\beta_{iku}, \gamma_{iku})_{u \in \{0,1,\ldots \lfloor \log_2(q)/r \rfloor\}},$$

where γ_{iku} is freshly random.

Each Game$_k$ is computationally indistinguishable from Game$_{k-1}$ because of the RLWE assumption. First, $k \in \Gamma_H$ and therefore, there is no re-encryption "edge" from user k to any user in Γ_C. Additionally, as before, all the re-encryption keys (k, i) have already been replaced with uniformly random ring elements in the prior games. Consequently, the secret key s_k is used only in the form of fresh RLWE samples in its public key and in the re-encryption keys. Thus, all these can be replaced by uniformly random ring elements by invoking the RLWE assumption. The security loss is proportional to the number of re-encryption key and re-encryption queries that user k was part of (an additional multiplicative factor $1 + \lfloor \log_2 q/r \rfloor$ is incurred in the security loss as each re-encryption key contains that many RLWE samples).

Game$_{\text{final}}$ is same as Game$_N$ except for the challenge ciphertext that is expressed as

$$c_0^* := r_1^* + m_b \in R_q, \ c_1^* := r_2^* \in R_q,$$

where r_1^*, r_2^* are freshly random ring elements in R_q. This is computationally indistinguishable from Game$_N$ by the RLWE assumption as well.

The last change guarantees that the challenge bit b is information-theoretically hidden from $\mathcal{A}$, and therefore, the advantage of the adversary in Game$_1$ is 0.

Putting together, we see that

$$\mathbf{Adv}_{\mathcal{A}}^{cpa}(\lambda) \leq (\rho \cdot (Q_{rk} + Q_{re}) + N + 1) \cdot \mathbf{Adv}_{\mathcal{D}}^{RLWE_{\phi,q,\chi e}}(\lambda)$$

where $\rho := 1 + \lfloor \log_2 q/r \rfloor$. This finishes our proof. □

3.5 PRE Cryptosystem with LWE Key Switching (GSW-PRE)

We describe our third PRE scheme which is based on GSW [GSW13] identity based FHE scheme. Our construction relies only on LWE security assumption.

3.5.1 Encryption Scheme

GSW encryption scheme was introduced by Gentry, Shahai and Waters [GSW13]. Message space for the scheme is restricted to $\mathcal{M} \in \{0, 1\}$. The scheme is broadly parameterized by :

- Security parameter λ.
- Working modulus q of $\kappa = \kappa(\lambda, d)$ bits.
- d is the maximum depth of the circuit.
- Lattice dimension n and $m = \mathcal{O}(n \log q)$. It is sufficient to take $m \geq 2n \log q$.
- Bound B for generating error e from distribution $\chi = \chi_B(\lambda)$.

Encryption scheme is described by the following algorithms:

- ParamsGen(λ): Choose appropriate q, n and $m = 2n \log q$. Set $N = (n+1) \cdot \lceil \log q \rceil$ and $\ell = \lceil \log q \rceil$. The public parameter pp consists of (q, n, m, ℓ, N).
- SecretKeyGen(λ, pp): Sample $\vec{\mathbf{t}} \in \mathbb{Z}_q^n$ from distribution χ, $\vec{\mathbf{t}} \leftarrow_{\$} \chi^n$. We set $sk = \vec{\mathbf{s}} \leftarrow \left(1, -\vec{\mathbf{t}}\right) \in \mathbb{Z}_q^{n+1}$. Also, set $\vec{\mathbf{v}} = \text{PowersOf2}(\vec{\mathbf{s}})$.
- PublicKeyGen(λ, pp, sk): Public key consists of $\vec{\mathbf{b}}$ and matrix $\mathbf{B}$. Matrix $\mathbf{B}$ is obtained by sampling uniformly from $\mathcal{U}_q^{m \times n}$, $\mathbf{B} \leftarrow_{\$} \mathcal{U}_q^{m \times n}$. We generate error vector as $\vec{\mathbf{e}} \leftarrow_{\$} \chi^m$ and set $\vec{\mathbf{b}} = \mathbf{B} \cdot \vec{\mathbf{t}} + \vec{\mathbf{e}}$. Set public key as $\mathbf{A} = \left[\vec{\mathbf{b}} \,||\, \mathbf{B}\right] \in \mathbb{Z}_q^{m \times (n+1)}$.
- Encrypt(pp, pk, μ): For encrypting a message $\mu \in \{0, 1\}$ we sample a uniform matrix $\mathbf{R} \in \mathcal{U}_{\{0,1\}}^{N \times m}$ and output the ciphertext $\mathbf{C} \in \mathbb{Z}_q^{N \times N}$ as follows:

$$\mathbf{C} = \text{Flatten}\left(\mu \cdot \mathbf{I}_N + \text{BitDecomp}\left(\mathbf{R} \cdot \mathbf{A}\right)\right)$$

- Decrypt($pp, sk, \mathbf{C}$): We recover μ' by performing the following two operations.
  1. Compute $\mathbf{w} = \mathbf{C} \cdot \vec{\mathbf{v}} \in \mathbb{Z}_q^N$.
  2. Recover $\mu' = \lfloor w_i / 2^i \rceil$ where $2^i \in (q/4, q/2]$.

3.5.2 Proxy Re-Encryption Scheme

GSW-PRE augments the above mentioned PKE scheme with two additional operations, namely, ReKeyGen and ReEncrypt. The operations for Proxy Re-Encryptions are described as follows:

- ReKeyGen$(pp, \lambda, sk_A, pk_B)$: ReKeyGen generates the evaluation public key that is required for a proxy to transform the ciphertext. In this scheme the re-encryption key is a sequence of $(n+1)$ matrices where each matrix is represented as $\mathbf{EK}[i] \in \mathbb{Z}_q^{N\times(n+1)}$. Each of the matrix is generated as follows:

$$\begin{aligned}\text{Sample uniformly random matrix } \mathbf{R_i} &\leftarrow \mathcal{U}_{\{0,1\}}^{N\times m}\\ \mathbf{EK}[i] &= \mathbf{R}_i \cdot \mathbf{A}_B + (\vec{\mathbf{v}}_A \gg i)\\ rk_{A\to B} &= (\mathbf{EK}[0], \cdots, \mathbf{EK}[n])\end{aligned} \tag{3.9}$$

  Where $v \gg i$ represents the addition of v on the i-th column.

- ReEncrypt$(pp, \mathbf{C}_A, rk_{A\to B})$: This operations transforms a ciphertext $\mathbf{C}_A$ into $\mathbf{C}_{A\to B}$. For this purpose we work only on the top ℓ row of ciphertext $\mathbf{C}_A$, represented as $\mathbf{C}_A^{\ell\times N}$ and build the resulting ciphertext $\mathbf{C}_{A\to B}$ in blocks of $\mathbb{Z}_q^{\ell\times N}$. Finally, we assemble the blocks by stacking them vertically which gives us a ciphertext $\mathbf{C} \in \mathbb{Z}_q^{N\times N}$.

$$\begin{aligned}\mathbf{C}^i &= \text{BitDecomp}\left(\mathbf{C}_A^{\ell\times N} \cdot \mathbf{EK}[i]\right),\ i \in [0, n]\\ \mathbf{C}_{A\to B} &= \left[\mathbf{C}^0 \ ||^\intercal \cdots ||^\intercal \mathbf{C}^n\right]\end{aligned} \tag{3.10}$$

  Ciphertext $\mathbf{C}_{A\to B}$ can be shown to represent an encryption of μ under the new key sk_B as described below. For simplicity, let $\mathbf{C}_A^{\text{top}} \in \mathbb{Z}_q^{\ell\times N}$ represent the top ℓ row block of ciphertext $\mathbf{C}_A$. Then, $\mathbf{C}_A^{\text{top}}$ can be written as:

$$\mathbf{C}_A^{\text{top}} = \text{Flatten}\left(\mu \cdot \mathbf{g} + \text{BitDecomp}\left(\mathbf{R}^{\text{top}} \cdot \mathbf{A}_A\right)\right)$$

  Now, for $i = 0$ we can see that,

$$\begin{aligned}\mathbf{C}_A^{\text{top}} \cdot \mathbf{EK}[0] &= \mathbf{C}_A^{\text{top}} \cdot [\mathbf{R}_0 \cdot \mathbf{A}_B + \vec{\mathbf{v}}_A]\\ &= \mathbf{C}_A^{\text{top}} \cdot \vec{\mathbf{v}}_A + \mathbf{C}_A^{\text{top}} \cdot \mathbf{R}_0 \cdot \mathbf{A}_B = \mu \cdot \mathbf{g} + \mathbf{R}_0'^{\text{top}} \cdot \mathbf{A}_B\end{aligned}$$

  In general, we can see that for any $i \le n$ we have,

$$\begin{aligned}\mathbf{C}_A^{\text{top}} \cdot \mathbf{EK}[i] &= \mathbf{C}_A^{\text{top}} \cdot [\mathbf{R}_i \cdot \mathbf{A}_B + (\vec{\mathbf{v}}_A \gg i)]\\ &= \mathbf{C}_A^{\text{top}} \cdot (\vec{\mathbf{v}}_A \gg i) + \mathbf{C}_A^{\text{top}} \cdot \mathbf{R}_i \cdot \mathbf{A}_B\\ &= (\mu \cdot \mathbf{g} \gg i) + \mathbf{R}_i'^{\text{top}} \cdot \mathbf{A}_B\end{aligned}$$

Finally, we can see that vertical stacking of these partial results leads us to the final ciphertext $\mathbf{C}_{A\to B}$.

$$\begin{aligned}\mathbf{C}_{A\to B} &= \left[\mathbf{C}_A^{\text{top}} \cdot \mathbf{EK}[0] \,\|^\intercal \cdots \|^\intercal\, \mathbf{C}_A^{\text{top}} \cdot \mathbf{EK}[n]\right] \\ &= \mu \cdot \mathbf{G} + \mathbf{R}' \cdot \mathbf{A}_B\end{aligned}$$

3.5.3 Correctness Constraint and Run-time Analysis

For an analysis on the correctness constraint we determine the noise $\mathbf{w}$ as:

$$\mathbf{w} = \mathbf{C}_{A\to B} \cdot \vec{\mathbf{v}}_B = \mu \cdot \vec{\mathbf{v}}_B + \mathbf{R}'\vec{\mathbf{e}}.$$

We can see that error in ciphertext mainly depends on norm of $\mathbf{R}'$ or $\|\mathbf{C} \cdot \mathbf{R}\|_1$. Assuming worst-case conditions we correctly decrypt the ciphertext for the following condition:

$$\|\mathbf{R}'\vec{e}\|_1 \le q/8 \text{ or, } q \ge 8mNB \text{ [worst-case bounds]}$$

Invoking central limit theorem we arrive at a more conservative bounds as:

$$q \ge 8\sqrt{mN}B \text{ [average-case bounds]} \tag{3.11}$$

Run-time for performing the re-encryption procedure depends upon the size of the matrices $\mathbf{EK}[i]$ and ciphertext $\mathbf{C}$. Hence, run-time complexity to re-encrypt each block is $\mathcal{O}\left(n^2 \cdot \log^2 q\right)$. Repeating this for the entire $n+1$ blocks we arrive at the overall time complexity of the GSW-PRE scheme given by:

$$\textbf{GSW-PRE runtime: } \mathcal{O}\left(n^3 \log^2 q\right).$$

3.5.4 Multi-hop GSW PRE

In case of a multi-hop scenario, a proxy interacts with the next proxy, thus building a chain of re-encryptions until the ciphertext reaches the final entity. Given h such hops we expect the noise to grow by number exponential in h w.r.t to security parameter λ. Applying correctness constraints from single hop GSW-PRE to multi-hops we arrive at the following bounds for q:

$$q \geq 8\sqrt{N^h m}B$$

3.6 PRE Cryptosystem with RLWE Key Generation and Key Switching (Ring-GSW PRE)

In this section, we describe our last PRE scheme which is based on ring variant of GSW FHE scheme. Similar to BV-PRE, the construction relies solely on RLWE security assumption and underlying cyclotomic polynomial arithmetic.

3.6.1 Ring-GSW Encryption Scheme

Messages are restricted to plaintext space of $\mathcal{M} \in R_p$ where $p \geq 2$. Unlike our previous scheme, we consider the coefficients of the polynomial to be in the range $[-q/2, q/2]$. The following represents the parameters of the encryption scheme:

- security parameter λ and ciphertext modulus q.
- plaintext modulus $p \ll q$ and ring dimension n.
- B- bounded discrete Gaussian distribution χ_B.
- Ternary distribution $\mathcal{T}$ which generates a ring polynomial with coefficients uniformly sampled from $\{-1, 0, 1\}$.
- depth d of the circuit for homomorphic evaluation.

The scheme encapsulates the following operations:

- ParamsGen$\left(1^{\lambda}\right)$: Choose positive integers $q = q\left(\lambda, d\right)$ and $n = n\left(\lambda, d\right)$. Return public parameter set $pp = (\ell, N, p, q, n)$ where $\ell = \lceil \log q \rceil$ and $N = 2\ell$.
- SecretKeyGen$\left(1^{\lambda}, pp\right)$: Sample polynomial $s \leftarrow_{\$} \chi_{B,R_q}$ and set $sk = (1; -s) \in R_q^{2\times 1}$.
- PublicKeyGen$\left(1^{\lambda}, pp, sk\right)$: To generate public key pk sample polynomials a and e from uniform and discrete Gaussian distributions respectively and proceed as follows:
$$\begin{gathered} a \leftarrow_{\$} \mathcal{U}_{R_q},\ e \leftarrow_{\$} \chi_{B,R_q},\ b = as + pe \\ \text{Set the public key, } pk = A^{1\times 2} = [b\ a] \end{gathered} \tag{3.12}$$
- Encrypt(pp, pk, μ): To encrypt a message polynomial $\mu \in R_p$ we sample a random vector $r \in R_q^N$ from uniform ternary distribution and an error matrix $\mathbf{E} \in R_q^{N\times 2}$ and set the encryption as follows:
$$\mathbf{C} = \mu \cdot \mathbf{G} + r \cdot \mathbf{A} + p\mathbf{E},\ r \leftarrow_{\$} \mathcal{T}_{R_q}^{N},\ \mathbf{E} \leftarrow_{\$} \chi_{B,R_q}^{N\times 2}$$
- Decrypt$(pp, sk, \mathbf{C})$: Given a ciphertext $\mathbf{C}$, plaintext μ is recovered by multiplying the first row of the ciphertext with sk. This is represented as follows:
$$\mu' = ((C_0 \times sk \bmod q) \bmod p) \equiv \mu \bmod p$$

3.6.2 Ring-GSW PRE Scheme

As in the case of GSW PRE scheme we introduce two more operations ReKeyGen and ReEncrypt. For a proxy to gain access to the re-encryption key, party A retrieves party B's public key and proceeds with the ReKeyGen operation. The operations for Ring-GSW PRE scheme are described as follows:

ReKeyGen(pp, sk_A, pk_B): ReKeyGen evaluation key consists of two ring polynomial matrices. To generate the matrices we first sample two uniformly random matrices $r_0, r_1 \in R_q^N$. Next we sample two error matrices from discrete Gaussian distribution $\chi_{R_q,B}$ and set the evaluation matrices $\mathbf{EK}\left(i\right)$ as follows:

$$r_i \leftarrow_{\$} \mathcal{T}_{R_q}^{N}, \quad \mathbf{E_i} \leftarrow_{\$} \chi_{R_q,B}^{N\times 2}, \quad i \in \{0, 1\}$$

$$\mathbf{EK}[i] = r_i \cdot \mathbf{A}_B + p\mathbf{E}_i + \left(\text{PowerOf2}\left(sk_A\right) \gg i\right)$$

$$rk_{A\to B} = \{\ \mathbf{EK}[0],\ \mathbf{EK}[1]\ \}$$

ReEncrypt$(pp, \mathbf{C}_A, rk_{A\to B})$: This operations results in a ciphertext $\mathbf{C}_{A\to B}$ which can be decrypted under B's secret key sk_B. We use the top ℓ rows of the ciphertext $\mathbf{C}_A$ to perform re-encryption and denote this as $\mathbf{C}_A^{\text{top}}$. Next, we multiply each of the matrices $\mathbf{EK}[i]$ with $\mathbf{C}_A^{\text{top}}$ and reassemble the results into a Ring-GSW ciphertext $\mathbf{C}_{A\to B}$. This is shown as follows:

$$\mathbf{C}^i_{A\to B} = \text{BitDecomp}\left(\mathbf{C}_A^{\text{top}}\right) \cdot \mathbf{EK}[i] \in R_q^{\ell\times 2}$$
$$\mathbf{C}_{A\to B} = \left[\mathbf{C}^0_{A\to B} \mid\mid^{\intercal} \mathbf{C}^1_{A\to B}\right]$$

3.6.3 Correctness Constraint Analysis

To formulate the correctness constraint of the scheme we have to ensure that there is no wrap around mod-q and coefficients of the noise term t are indeed in the range $[-q/2, q/2]$. Noise term t is given by:

$$t = \mathbf{C}_{A\to B,0} \times sk_B = \mathbf{C}_{A\to B,0,0} - s_B\mathbf{C}_{A\to B,0,1}$$

Let, $\alpha_i = \text{BitDecomp}\left(\mathbf{C}^{\text{top}}_{A,j,0}\right)$ and $\beta_i = \text{BitDecomp}\left(\mathbf{C}^{\text{top}}_{A,j,1}\right)$ for $(i,j) \in [0,\ell)$. Then, $\mathbf{C}_{A\to B,0}$ can be shown as:

$$\mathbf{C}_{A\to B,0} = [\alpha_i\ \beta_i]_{i=0}^{\ell-1} \cdot [r_j\mathbf{A}_B + p\mathbf{E}_j + \text{PowerOf2}\left(sk_A\right)]$$
$$\text{where } i \in [0,\ell) \text{ and } j \in [0, 2\ell)\,.$$

$$\begin{aligned}\mathbf{C}_{A\to B,0,0} &= b_B\sum_{i=0}^{\ell-1}\left(\alpha_i r_i + \beta_i r_{\ell+i}\right) + p\sum_{i=0}^{\ell-1}\left(\alpha_i\mathbf{E}_{i,0} + \beta_i\mathbf{E}_{\ell+i,0}\right)\\ &+ \sum_{i=0}^{\ell-1}\alpha_i\text{PowerOf2}\left(1\right) + \sum_{i=0}^{\ell-1}\beta_i\text{PowerOf2}\left(-s_A\right)\\ &= b_B r'_0 + p\mathbf{E}'_{0,0} + \alpha - s_A\beta\end{aligned}$$

Similarly,

$$\begin{aligned}\mathbf{C}_{A\to B,0,1} &= a_B \sum_{i=0}^{\ell-1} (\alpha_i r_i + \beta_i r_{\ell+i}) + p \sum_{i=0}^{\ell-1} (\alpha_i \mathbf{E}_{i,1} + \beta_i \mathbf{E}_{\ell+i,1}) \\ &= a_B r_0' + p\mathbf{E}_{0,1}'\end{aligned}$$

Therefore,

$$t = r_0' \left(b_B - a_B s_B\right) + p\left(\mathbf{E}_{0,0}' - s_B \mathbf{E}_{0,1}'\right) + \mu + pe \approx \mu$$

For correct decryption $\|t\|_\infty \leq q/2$. By using central limit theorem on B-bounded discrete Gaussian distribution we arrive at the final correctness constraint:

$$\begin{gathered}\|t\|_\infty \leq p \cdot \left(4n\lceil \log q\rceil B + 2\sqrt{n}\lceil \log q\rceil + B\right) \approx 5pnB\lceil \log q\rceil \\ \text{or, } q \geq 10pnB\lceil \log q\rceil \ \text{[average case bounds]}\end{gathered} \tag{3.13}$$

3.6.4 Key-Switching and Automorphism

We introduce two new operations, KeySwitchGen and SwitchKey to aid the ciphertext transformation process. Assuming we have a ciphertext $\mathbf{C}$ encrypted under secret key sk we describe below the key-switching procedure which results in a ciphertext $\mathbf{C}^*$ which can be decrypted with secret key sk^*:

KeySwitchGen(pp, sk, sk^*): We generate an evaluation key which consists of two RLWE matrices similar to ReKeyGen operation. First we generate a noise free RLWE pair from s^*. Next, to generate each of the RLWE matrix we generate a random vector from ternary distribution $\mathcal{T}_{R_q}$ and an error matrix from discrete

Gaussian distribution and proceed as follows:

$$\text{Sample } a \leftarrow_{\$} \mathcal{U}_{R_q};\ b = as^*;\ \text{Set } \mathbf{A} = [b\ a] \in R_q^{1\times 2}$$

$$\text{Sample } r_i \leftarrow_{\$} \mathcal{T}_{R_q};\ \mathbf{E}_i \leftarrow_{\$} \chi_{R_q,B}$$

$$\mathbf{EK}[i] = r_i\mathbf{A} + p\mathbf{E}_i + (\text{PowerOf2}(sk) \gg i)$$

$$\text{Set } ek = \{\mathbf{EK}[0], \mathbf{EK}[1]\}$$

SwitchKey$(pp, \mathbf{C}, ek)$: This results in a ciphertext $\mathbf{C}^*$ and is analogous to ReKeyGen procedure of Ring-GSW-PRE scheme. We outline the procedure as follows:

$$\mathbf{C}^{*,i} = \text{BitDecomp}\left(\mathbf{C}^{\text{top}}\right) \cdot \mathbf{EK}[i] \in R_q^{\ell\times 2}$$

$$\mathbf{C}^* = \left[\mathbf{C}^{*,0}\ ||^{\intercal}\ \mathbf{C}^{*,1}\right]$$

Next, we discuss the application of Automorphism transformation in conjunction with key-switching operation. Automorphism transformation, denoted by σ, has an effect of rotating or permuting the plaintext slots. While plaintext slots can be easily generated for power of two cyclotomic polynomials by an application of negacyclic NTT, plaintext slots can be rotated only for special cyclotomic polynomials where plaintext modulus satisfies $p \equiv 1 (\mod m)$ and $R = \mathbb{Z}[X]/\Phi_m(X)$. Number of slots that can be used is denoted by $\ell = \varphi(m)$ where each slot element $m_i \in \mathbb{Z}_p$. An Automorphism transformation $\sigma(R, i)$, for $i \in \varphi(m)$, permutes the coefficient of the polynomial. Given a ciphertext $\mathbf{C}$ an Automorphism transformation produces a ciphertext $\mathbf{C}' = \sigma(\mathbf{C}, i)$ which can be decrypted by a secret key $sk' = \sigma(sk, i)$. At this stage, we need to key-switch the ciphertext $\mathbf{C}'$ so that it can be decrypted by the original secret key sk. This is shown as follows:

$$\text{Generate } ek_i = \text{KeySwitchGen}\left(pp, sk_i, sk\right),\ \forall\ i \in \varphi(m)$$

$$\text{Compute } \sigma(\mathbf{C}, i),\quad \text{Output } \mathbf{C}' = \text{SwitchKey}\left(pp, \sigma(\mathbf{C}, i), ek_i\right)$$

3.7 Parameter Selection

A general issue with lattice encryption schemes is that they are more complicated to parameterize than other families of encryption schemes. Parameter selection is governed largely by a correctness condition (which is specific to the scheme being analyzed) and security conditions for the underlying security assumptions.

For NTRU-ABD-PRE, parameter selection is governed by the correctness condition (3.6), the security condition (3.3) accounting for the NTRU immunity against subfield lattice attacks, and RLWE security condition (3.4).

For BV-PRE, GSW-PRE and Ring-GSW-PRE parameter selection is governed by their respective correctness conditions (3.8, 3.11, 3.13) and LWE/RLWE security condition (3.4).

We identify the parameter tradeoffs associated with the correctness constraint and security constraints for PRE schemes in the experimental results section of this paper. Of high importance are the ring dimension n and ciphertext modulus q which have the largest direct impact on the runtimes of the scheme. The value of n should be kept as small as possible as runtime is at least linear in n for all operations. The value of q determines the sizes of integers that need to be manipulated computationally. Ideally q should be kept less than the threshold 2^{32} or the less preferred threshold 2^{64} to utilize native arithmetic operations supported with the processor word sizes in modern processors.

The value of d, the number of hops that the re-encryption scheme supports, can be thought of as an application-specific parameter determined by the number of PRE hops needed.

We begin the process of parameter selection with the security parameter δ, also known as the root Hermite factor. The root Hermite factor is discussed above in the Related Work section with relevant references. A heuristic argument is presented in

[CN11] which suggests that a root Hermite factor of $\delta = 1.006$ could provide adequate security. Therefore, we select to be as close as possible to the ceiling $\delta < 1.006$.

The bound for discrete Gaussian distribution $\chi_i(x)$, where $i \in (k, e)$, is expressed as $B_i = \sigma_i\sqrt{\alpha}$, where σ_i is the standard deviation of the distribution and α determines the effective probability that a coefficient generated using discrete Gaussian distribution (or a product of discrete Gaussians) exceeds the bound B_i [LTV13].

The value of σ_e is usually chosen in the range from 3 to 6, and we set the value of σ_e to 4 as in [GHS12c]. We set α to 9, which for the case of an integer generated using discrete Gaussian distribution corresponds to the theoretic probability of at most 2^{-8} of choosing a value that exceeds the upper bound B_i.

We validated our selection of σ_i and α experimentally. Over 35,000 iterations of encryption/decryption (using different keys) for ring dimensions in the range from 2^9 to 2^{15} (5,000 iterations for each value of ring dimension), we observed no decryption errors. Note that when products of two discrete Gaussians (encryption scheme), three discrete Gaussians (single-hop re-encryption in the case of NTRU-ABD-PRE), and higher number of discrete Gaussians (multi-hop re-encryption in the case of NTRU-ABD-PRE) are considered, the practical probability drops dramatically. This implies that smaller practical values of α may be possible.

Subsequent to the selections of d, δ, σ_e, and α, we can choose n, q, and σ_k (only in the case of NTRU-ABD-PRE) experimentally using appropriate correctness and security constraints to minimize runtime/throughput for various values of the key switching window r and plaintext modulus p.

Tables 3.3 and 3.4 shows the minimum number of bits needed to represent the ciphertext modulus q (which we refer to as $k = \lfloor \log_2 q + 1 \rfloor$), as a function of ring dimension n and re-encryption depth d for the key switching window of unity assuming the other parameters were selected as above. It can be seen that NTRU-ABD-PRE

Table 3.3 Minimum Bits Required to Represent Modulus q for Selections of Ring Dimension n and Multiple Re-encryption Depths d at $p = 2$ and $r = 1$

PRE Scheme	d	**Ring dimension** n					
		512	1024	2048	4096	8192	16384
NTRU-ABD-PRE	1	–	35	36	37	38	39
	2	–	–	–	93	96	99
	3	–	–	–	–	–	–
BV-PRE	1	17	18	18	19	19	20
	2	18	18	19	19	20	20
	3	18	19	19	20	20	21

Table 3.4 Minimum Bits Required to Represent Modulus q for Selections of Ring Dimension n and Multiple Re-encryption Depths d at $p = 2$ and $r = 1$

PRE Scheme	d	n	k	**PRE Scheme**	d	n	k
GSW-PRE	1	511	21	Ring-GSW-PRE	1	1024	22
		1023	22			2048	23
	2	511	22		2	1024	23
		1023	23			2048	24
	3	511	22		3	1024	23
		1023	23			2048	24

requires a ring dimension of at least 4096 and ciphertext modulus of approximately 100 bits to support two re-encryption hops in contrast to a ring dimension of 512 and 18-bit ciphertext modulus for BV-PRE, implying that NTRU-ABD-PRE can be treated as a single-hop scheme for all practical purposes. It should also be noted that all ciphertext moduli for BV-PRE require representations with less than 32 bits, thus enabling efficient implementations based on native integer types (32-bit and 64-bit integers). The growth of the number of ciphertext modulus bits k with increase in ring dimension n for both schemes can be easily estimated from expressions (3.6) and (3.8): for NTRU-ABD-PRE the dependence of q on n is $n^{1+d/2}$ while for BV-PRE, it is $\sqrt{n}$.

Table 3.5 illustrates the effect of increasing the key switching window r and plaintext modulus p on the minimum values of ring dimension n and the number of bits k required to represent the ciphertext modulus q for both schemes. Increase in r reduces the dimension of the re-encryption key (to $\lfloor \log(q)/r \rfloor + 1$) and the number of

Table 3.5 Dependence of Minimum Values of Ring Dimension n and the Number of Bits k Required to Represent the Ciphertext Modulus q, on Plaintext Modulus p and Key Switching Window r for Re-encryption Depth d of Unity

PRE Scheme	p	$r=1$		$r=2$		$r=4$		$r=8$		$r=16$	
		n	k	n	k	n	k	n	k	n	k
NTRU-ABD-PRE	2	1024	35	1024	35	1024	35	1024	38	2048	48
	16	2048	53	2048	53	2048	53	2048	53	2048	57
	256	4096	78	4096	78	4096	78	4096	78	4096	78
	4096	4096	102	4096	102	4096	102	4096	102	4096	102
	65536	4096	126	4096	126	4096	126	4096	126	4096	126
BV-PRE	2	512	17	512	17	512	18	1024	22	1024	29
	16	512	20	1024	21	1024	22	1024	25	1024	32
	256	1024	25	1024	25	1024	26	1024	29	1024	37
	4096	1024	29	1024	29	1024	30	1024	33	2048	41
	65536	1024	33	1024	33	1024	35	1024	37	2048	45

Table 3.6 Dependence of Minimum Values of Ring Dimension n and the Number of Bits k Required to Represent the Ciphertext Modulus q, on Re-encryption Depth d and Key Switching Window r for BV-PRE at $p = 2$

d	$r=1$		$r=2$		$r=4$		$r=8$		$r=16$	
	n	k	n	k	n	k	n	k	n	k
1	512	17	512	17	512	18	1024	22	1024	29
2	512	18	512	18	512	19	1024	23	1024	30
5	512	19	512	19	512	20	1024	24	1024	31
10	512	19	512	20	1024	22	1024	25	1024	32
20	512	20	1024	21	1024	23	1024	25	1024	33
50	1024	22	1024	23	1024	24	1024	28	1024	35
100	1024	23	1024	24	1024	25	1024	29	1024	36

NTT operations (performed for groups of r bits of each coefficient), which effectively reduces the re-encryption runtime. Increase in p improves the plaintext throughput of PRE and reduces the ciphertext expansion factor defined as $k/\lfloor \log_2 p + 1 \rfloor$. More detailed information on these performance metrics is presented in Section 3.9.

The results in Table 3.5 suggest that r can be used to reduce the re-encryption runtime with negligible effect on the encryption/decryption runtimes. For instance, the ring dimension and the number of bits k required to represent the ciphertext modulus q are essentially the same for BV-PRE at $p = 2$ when r is increased from 1 to 4. At the same time, this reduces the runtime of re-encryption by roughly a factor of 4. One can also observe for BV-PRE that increasing the plaintext modulus to 65536 (2 bytes per polynomial coefficient) raises the ring dimension requirement only by a factor of 2 (to 1024), which implies that a much higher plaintext throughput can be achieved for BV-PRE by using large values of plaintext modulus (in applications where runtime/latency is not critical). It can also be seen that the ring dimension / ciphertext modulus requirements are substantially lower for BV-PRE as compared to NTRU-ABD-PRE.

Table 3.6 shows the effect of increasing the re-encryption depth d for different values of key switching window r on the minimum values of ring dimension n and the number of bits k required to represent the ciphertext modulus q for BV-PRE (results for NTRU-ABD-PRE are not presented because the values of ring dimension and ciphertext modulus are impractical for $d = 2$). It can be seen that BV-PRE supports at least 20 re-encryption hops at $n = 512$ (the maximum number is 23 hops). One can also observe that the number of bits k required to represent the ciphertext modulus q changes slowly with increase in re-encryption depth because the noise growth is additive (rather than multiplicative as in the case of NTRU-ABD-PRE), and 100 re-encryption hops can be supported without exceeding the ring dimension of 1024.

3.8 Software Implementation

3.8.1 Software Library Design

We implemented our PRE scheme in PALISADE, a general-purpose portable multi-threaded C++ library designed to support and ease the development of lattice-based encryption prototypes.

The main runtime performance bottleneck (in RLWE based PRE schemes) is conversion between coefficient and evaluation representations. For the power-of-two cyclotomic rings, the most efficient algorithm to perform this operation is the Fermat-Theoretic Transform (FTT) [APS13]. We implemented FTT with NTT as a subroutine in PALISADE. For NTT, the iterative Cooley-Tukey algorithm with optimized butterfly operations was applied. The two slowest sub-operations needed to support NTT operations are multiplication and modulo reduction. For multiplication, we used the standard shift-and-add multiplication algorithm as it performs well for relatively small ciphertext moduli (up to multiple hundreds of bits, but in our case the running bitwidths required to represent ciphertext moduli do not exceed 128 bits). For modulo reduction, we used the generalized Barrett modulo reduction algorithm [DQ00], which requires one pre-computation per NTT run and converts one modulo reduction to roughly two multiplications. For discrete Gaussian sampling, we used the inversion method from [Pei10].

The conventional key switching procedure works with the key switching window r of unity, implying that every coefficient of the ciphertext polynomial c is decomposed into bits. Although this technique dramatically reduces the noise growth (from $\|c\|_\infty$ to 1), it significantly increases both computational and space complexities of re-encryption. As there is no efficient method to extract bits from a polynomial in CRT form, the ciphertext polynomial c has to be first converted to the coefficient form, then decomposed into polynomials over $\mathbb{Z}_2$, and finally all of these bit-level polynomials need to be converted back to CRT form prior to performing the component-wise multiplication with the elements of the re-encryption key. The total computational cost of this operation is $\lfloor \log_2(q) \rfloor + 2$ FTT operations. The size of the re-encryption key is approximately $n \cdot \log_2(q)^2$ bits (or the double of that in the case of BV-PRE).

To reduce the number of FTT operations and size of the re-encryption key, we consider a generalized key switching window of up to 16 bits. It can be seen that in the case of $r = 8$, the number of FTT operations reduces to $\lfloor \log_2(q)/8 \rfloor + 2$ and the re-encryption key size reduces by a factor of 8. At the same time, Table 3.5 suggests that the number of bits required to represent q, which is determined by the correctness constraint, increases only by 3 bits compared to the case of $r = 1$ with the minimum ring dimension n staying at the same level (for NTRU-ABD-PRE at $p = 2$). In view of the above, it can be expected that the re-encryption time for this case will be significantly less than for $r = 1$, which is demonstrated in the next section.

(CPU) Implementation of GSW PRE scheme is found to be rather impractical. The main bottleneck in GSW PRE scheme arises in computations over matrices. Because the parameter m is a function on LWE dimension n and $\lceil \log q \rceil$ we end up with huge matrix dimensions. We dealt with this issue to some extent by dropping the factor of 2 from the computation of m. Furthermore, we kept all ciphertexts in normal form rather than the bit decomposed form.

Given the impractical runtimes of GSW PRE, we opted for a GPGPU implementation. Some of the key features of our GPU implementation are as follows:

Multiprecision Integer: NVIDIA GPU architectures are restricted to 32-bit native integer data types. We implemented a multi precision integer class where large integers are represented internally with an array of 32-bit unsigned integer datatype.

Fast Integer Arithmetic: We provided an implementation of modular reduction using generalized Barrett reduction [DQ98]. Barrett reduction requires an additional constant term which we precompute and keep on the device memory. Large integer multiplication are implemented using shift-and-multiply algorithm and all device kernels make use of device intrinsic methods.

Fast Randomness Generation: Because of large dimensions generating random elements on the fly is quite expensive. We remedy this problem by generating random elements on device memory using cuRAND library. In particular our implementation uses CURAND_RNG_PSEUDO_MTGP32 generators. It is 5x faster than other random number generators of the same family and atleast 10x faster than CPU random number generators.

Minimal Memory Transfers: In our implementation we have eliminated most of the data transfers between CPU. We generate all keys and ciphertexts on device memory thereby eliminating calls to host-to-device and device-to-host data transfers.

Matrix Vector Multiplication Kernel: We optimized the Matrix-Vector multiplication by computing dot products for each row in parallel. Furthermore, we compute the product of integers in parallel followed by a logarithmic reduction phase to compute the final summation.

Matrix Multiplication Kernel: Our matrix multiplcation kernel closely follows the approach presented in NVIDIA guide [Hoc12] with few modifications. We divide the input matrices into smaller non-overlapping blocks and load them into shared memory. At each stage, we compute a partial product from matrix multiplication of smaller blocks and finally add them form an entry of resultant matrix.

3.9 Experimental Evaluation

3.9.1 Methodology

We identify a set of standard metrics, including those used in related work [AFGH06, NAL15] with which we evaluate the performance of our PRE design and implementation. These metrics include:

1. Runtime / Latency: How long it takes to perform the implemented Encryption, Re-Encryption and Decryption operations for various parameter settings.
2. Throughput: How many plaintext bits per unit time can be processed by the implemented operations for various settings.
3. Ciphertext Expansion: How many bits are required to represent ciphertext for every bit in the plaintext.
4. Memory Usage: How much memory is required to run the implemented operations for various settings.

We would normally also use security as a metric to evaluate the performance of our PRE design and implementation, but we assume a ceiling on the security parameter such that $\delta < 1.006$, and we would want δ to be as close as possible to 1.006 to provide as quick runtime performance as possible while providing adequate security. For our experimental analyses, we varied the ring dimension n, key switching window r, plaintext modulus p, and number of hops d to explore tradeoffs in runtime and amortized throughput.

Because we perform all experiments in the single-threaded mode and our implementation does not access disk or networking interfaces, we use latency as a means of determining the temporal overhead of the implementation. Further, runtime performance is useful, for example, when assessing fitness for real-time applications when end-to-end latency is critical. We use the throughput metric to assess how much plaintext data can be processed by the implementation per unit time.

Related to ciphertext expansion is memory usage. Memory intensive operations may not be easily supported on resource-constrained devices, such as embedded systems used for disposable sensor nodes. We therefore, differentiate between the memory requirements of PRE clients (subscribers and publishers) from those of PRE servers (brokers). Memory usage for PRE clients is governed primarily by the size of public/private keys and ciphertext elements. At the same time, the memory requirements for PRE servers are determined primarily by the size of re-encryption keys and decomposed ciphertext ring elements.

We conducted experiments for our PRE implementation on a commodity desktop/laptop computing environment. The evaluation environment for NTRU-ABD-PRE and BV-PRE used an Intel Core i7-3770 CPU rated at 3.40GHz and 16GB of memory running CentOS 7. Our experiments for evaluation of Ring-GSW PRE were run on a Dell Inspiron laptop with Intel Core i7-7700HQ running at 2.8 GHz, on an Ubuntu 18.04 operating system with 16 GB of physical memory, using g++ compiler version 7.4.0. Our GPU experiments for evaluation of GSW PRE were run on an university HPC with host as Intel Xeon Silver-R 4114 Core running at 2.2 GHz, on an Scientific Linux 6.10 operating system with 24 GB of physical memory, using nvcc compiler version 10.0. The GPU device consists of two NVIDIA TITAN RTX processor each of which has 72 multiprocessors and 64 CUDA cores per multiprocessors running at 1.77 GHz. All of our implementations were compiled as single-threaded and used only one core despite our test environment providing multiple cores.

We generated random plaintext samples using discrete uniform distribution from 0 to $p-1$. We ran 100 iterations for a subset of parameter datasets listed in Tables 3.3-3.6 and evaluated the mean runtime of encryption, decryption, and re-encryption operations, with decryption runtime measured before and after re-encryption, and the runtime of multiple re-encryptions. The number of correct decryptions was also recorded, and no decryption errors were observed.

In Tables 3.10 through 3.12 we present experimental results for the dependence of runtime and throughputs of PRE (NTRU-PRE and BV-PRE) operations on variations in key configuration parameters, including the ring dimension, key switching window, plaintext modulus, and number of re-encryption hops. We show throughputs in kilobits per second (Kbps) for encryption, re-encryption, and decryption amortized in terms of the plaintext size.

The ciphertext expansion factor is equal to $k = \lfloor \log_2 q + 1 \rfloor$ in all tables except for 3.11. Although not directly related to the security provided, the key size in bits is equal to the ring dimension n times the number of bits k in ciphertext modulus q.

We consider key generation to be an offline process which is run once for most feasible applications of the PRE capability. For all (except for GSW-PRE) of our experimental configurations we observed key generation and proxy key generation runtime of less than 1 second.

3.9.2 Single-Hop Re-Encryption

Table 3.7 shows the effect of changes in ring dimension n on runtime, amortized throughputs, and ciphertext expansion factors for single-hop re-encryption using both NTRU-ABD-PRE and BV-PRE schemes with security parameter $\delta \leq 1.006$. The highest encryption, re-encryption, and decryption throughputs and lowest runtime are observed for the smallest ring dimension: 1024 and 512 for NTRU-ABD-PRE and BV-PRE, respectively. The ciphertext expansion, which is proportional to the number of bits k required to represent the ciphertext modulus q, and memory usage for both PRE clients and servers, which is proportional to the product of ring dimension and the number of bits k required to represent the ciphertext modulus q, are lowest for the smallest value of ring dimension. This implies that one should always choose the smallest ring dimension satisfying the desired security level.

Note that the runtimes for BV-PRE scheme operations are always lower than for NTRU-ABD-PRE due to lower requirements on the ring dimension and the number of bits required to represent the ciphertext modulus of the former. For the same ring dimension, the runtime improvement factors observed from Table 3.7 are approximately 1.2 for encryption, 1.5 for decryption, and 2.5 for re-encryption operations. Considering that the lowest ring dimension with security parameter $\delta \leq 1.006$ for BV-PRE is 512, the improvement factors for throughputs at smallest ring dimension are 1.3, 1.6, and 2.9 for encryption, decryption, and re-encryption, respectively. The decryption times after regular encryption and proxy re-encryption are approximately the same for all datasets, which also applies to all other experimental results presented in this paper.

Table 3.10 shows the dependence of runtime and throughputs on variations in the key switching window r for single-hop re-encryption with security parameter $\delta \leq 1.006$. The plaintext modulus is kept the same ($p = 2$). It can be seen that for both schemes the highest encryption runtime and throughput are observed for $r = 1$

Table 3.7 Experimental Runtime Performance of Encryption, Decryption, and Re-encryption Operations for Ring Dimension n at r=1, p=2, and d=1

Configuration			Runtime				Throughput		
PRE Scheme	n	k	**Enc (ms)**	**Dec before ReEnc (ms)**	**ReEnc (ms)**	**Dec after ReEnc (ms)**	**Enc (Kbps)**	**ReEnc (Kbps)**	**Dec after ReEnc (Kbps)**
NTRU-ABD-PRE	1024	35	2.13	2.45	67.08	2.44	481.06	15.26	418.85
	2048	36	4.62	5.27	150.95	5.26	443.27	13.57	389.71
	4096	37	9.80	11.07	331.73	11.05	417.94	12.35	370.78
	8192	38	20.69	23.35	724.80	23.29	395.95	11.30	351.70
	16384	39	44.15	49.73	1597.81	49.41	371.14	10.25	331.58
BV-PRE	512	17	0.85	0.76	11.77	0.76	604.37	43.51	674.62
	1024	18	1.81	1.64	27.48	1.63	567.03	37.26	628.04
	2048	18	3.84	3.47	59.83	3.44	533.82	34.23	594.85
	4096	19	7.99	7.24	131.68	7.22	512.70	31.11	566.95
	8192	19	17.00	15.80	296.63	15.33	481.85	27.62	534.30
	16384	20	35.77	32.82	634.71	32.70	458.07	25.81	501.10

Table 3.8 Experimental Runtime Performance of GSW Proxy Re-Encryption on CPU and GPU for Different LWE Dimension and Modulus

PRE Scheme	Configuration				Runtime				
	n	m	k (bits in q)	N	KeyGen (ms)	Encrypt	Decrypt (ms)	ReKeyGen per EK	ReEnc per EK
GSW-PRE (CPU)	511	10731	21	10752	288.23	>10 m	0.01	>10 m	5.45 s
		15841	31	15872	476.25	>10 m	0.01	>10 m	12.24 s
GSW-PRE (GPU)	511	10731	21	10752	57.66	444.5 ms	0.06	420.2 ms	4.9 ms
		15841	31	15872	95.2	1022.8 ms	0.09	990.5 ms	15.3 ms
	1023	22506	22	22528	234.8	3670 ms	0.11	3689 ms	11.5 ms
		38874	38	38912	409.6	12310.7 ms	0.13	12486.6 ms	21.5 ms

For Memory Consideration m is Reduced to $m = n \log q$.

(which uses the smallest the number of bits required to represent the ciphertext modulus.) As the key switching window r increases, the re-encryption time declines until the ring dimension is forced to double by the security constraint. This lowest re-encryption runtime occurs at $r = 8$ and $r = 4$ for NTRU-ABD-PRE and BV-PRE, respectively. Note that the encryption and decryption runtimes for these values of r are approximately the same as for $r = 1$, which implies that $r = 8$ and $r = 4$ are optimal values for all operations of NTRU-ABD-PRE and BV-PRE, respectively, from the runtime/latency perspective. At the same time, the re-encryption throughput at $r = 16$ is highest for both schemes. This implies that in applications where re-encryption througput needs to be maximized and latency requirements are low,

Table 3.9 Experimental Runtime Performance of Ring-GSW Proxy Re-Encryption for Different Ring Dimension, Modulus Bits, $p = 2$ and $r = 1$

PRE Scheme	Configuration		Runtime						Throughput		
	n	k	KeyGen (ms)	Enc (ms)	Dec before ReEnc (ms)	ReKeyGen (ms)	ReEnc (ms)	Dec after ReEnc (ms)	Enc (kbps)	ReEnc (kbps)	Dec (kbps)
Ring-GSW	1024	22	0.38	17.32	0.12	34.28	120.02	0.11	59.12	8.53	9309
		41	0.42	31.56	0.13	61.41	422.86	0.08	32.43	2.42	12325
	2048	23	0.93	37.24	0.29	72.27	275.5	0.19	54.99	7.43	10778
		41	0.85	62.91	0.27	125.37	864.02	0.16	32.55	2.37	12666

Table 3.10 Experimental Runtime Performance of Encryption, Decryption, and Re-encryption Operations on Key Switching Window Size r at p=2 and d=1

Configuration				Runtime				Throughput		
PRE Scheme	r	n	k	**Enc (ms)**	**Dec before ReEnc (ms)**	**ReEnc (ms)**	**Dec after ReEnc (ms)**	**Enc (Kbps)**	**ReEnc (Kbps)**	**Dec after ReEnc (Kbps)**
NTRU-ABD-PRE	1	1024	35	2.13	2.45	67.08	2.44	481.06	15.26	418.85
	2	1024	35	2.13	2.45	36.65	2.44	480.75	27.94	419.03
	4	1024	35	2.13	2.45	21.27	2.44	480.09	48.15	418.86
	8	1024	38	2.11	2.46	13.88	2.44	484.61	73.77	418.95
	16	2048	48	4.72	5.45	19.97	5.42	434.23	102.57	377.71
BV-PRE	1	512	17	0.85	0.76	11.77	0.76	604.37	43.51	674.62
	2	512	17	0.83	0.74	6.38	0.74	615.12	80.21	691.43
	4	512	18	0.84	0.77	4.33	0.76	607.58	118.27	676.02
	8	1024	22	1.78	1.60	6.23	1.60	576.85	164.25	639.50
	16	1024	29	2.00	1.82	5.41	1.82	512.23	189.15	562.60

$r = 16$ could be the preferred choice. It should be noted that the ciphertext expansion grows with r, memory usage by PRE clients increases proportionally to the number of bits required to represent the ciphertext modulus and ring dimension, and memory usage by PRE servers declines as the re-encryption keys are composed of $\lfloor \log_2 (q) / r \rfloor + 1$ ring elements.

Table 3.11 illustrates the effect of plaintext modulus p on performance metrics of PRE operations for both schemes. The key switching window is kept constant ($r = 1$). One can see that runtime increases as p rises due to increased requirements on the number of bits k required to represent the ciphertext modulus q and the ring dimension n. At the same time, plaintext throughputs increase until $p = 4096$ for both schemes. Ciphertext expansion factors, defined as $k/\log_2 p$, are highest at $p = 65536$. This suggests that larger plaintext moduli may be suggested when high

Table 3.11 Experimental Runtime Performance of Encryption, Decryption, and Re-encryption Operations on Plaintext Modulus p at r=1 and d=1

Configuration				Runtime				Throughput		
PRE Scheme	p	n	k	**Enc (ms)**	**Dec before ReEnc (ms)**	**ReEnc (ms)**	**Dec after ReEnc (ms)**	**Enc (Kbps)**	**ReEnc (Kbps)**	**Dec after ReEnc (Kbps)**
NTRU-ABD-PRE	2	1024	35	2.13	2.45	67.08	2.44	481.06	15.26	418.85
	16	2048	53	5.38	7.73	228.30	7.55	1523.62	35.88	1085.06
	256	4096	78	16.20	23.05	1016.58	22.98	2022.23	32.23	1425.92
	4096	4096	102	20.00	28.94	1642.66	28.88	2458.12	29.92	1701.95
	65536	4096	126	21.24	33.66	2141.11	34.03	3085.50	30.61	1925.83
BV-PRE	2	512	17	0.85	0.76	11.77	0.76	604.37	43.51	674.62
	16	512	20	0.95	0.91	13.84	0.92	2156.82	147.93	2221.67
	256	1024	25	2.03	1.90	36.65	1.95	4028.96	223.53	4200.40
	4096	1024	29	2.33	2.15	47.28	2.19	5269.21	259.90	5600.50
	65536	1024	33	2.74	2.47	63.57	2.41	5989.05	257.73	6809.95

throughput and low ciphertext expansion are sought, and latency requirements are secondary. One can also see that the memory usage of both PRE clients and servers increases with p due to requiring more bits to represent the ciphertext modulus and larger ring dimensions, which may be an issue for embedded systems (PRE clients).

All tables for single-hop re-encryption suggest that BV-PRE outperforms NTRU-ABD-PRE for all performance metrics. The best BV-PRE re-encryption runtime of 4.33 ms is almost two orders of magnitudes faster than the runtime reported for comparable conditions (same ring dimension of 512) in the independent work of [NAL15]. Besides being faster than the scheme reported in [NAL15], BV-PRE is unidirectional and is based strictly on the RLWE assumption.

We observed experimentally that as ring dimension n increases for our schemes, the latency due to Encryption, Re-Encryption and Decryption increase, but the ammortized cost and throughput decrease. Furthermore, ciphertext expansion and memory requirements increase. The effect of increasing the key switching window r is similar to the effect of increasing n, except that Re-Encryption latency decreases and throughput increases. The effect of increasing plaintext modulus p is similar to the effect of increasing ring dimension. These results may be used for selecting optimal configuration of these three parameters in practical single-hop PRE applications.

From the GSW-PRE experiments on CPU, we can see that encryption time exceeds over 10 minutes because of the large dimension of matrices. Because of this reason, we chose to generate a single matrix out of the $n + 1$ ReKeyGen matrices. Consequently, we were able to report re-encryption times for a single matrix of the ReKeyGen matrices. Complete runtime of ReKeyGen and ReEncrypt procedure can be calculated by scaling the timings with LWE dimensions. From Table 3.8, we can infer that ReEncryption timings are in the order of seconds and very far from being practical.

Implementation of GSW-PRE on GPU yields significant order of boost in performance. From Table 3.8, we can observe that because of matrix-vector multiplication parallelization, we achieve a speedup of more than 5x on an average. A major performance acceleration is achieved in encryption and re-encryption key generation procedures where the runtimes improve by more than 100x. Furthermore, the actual re-encryption procedure is now reduced to a few milliseconds. We remark that the entire re-encryption procedure can now be completed under a second assuming single threaded host processor.

As compared to GSW-PRE, Ring-GSW-PRE possesses the capability to encrypt and re-encrypt a large number of bits/digits in a single ciphertext and consequently we expect higher throughput per operation. Most of the cases of key generation and decryption are very fast only taking sub-milliseconds. Re-encryption key generation takes nearly twice the amount of time of encryption for a particular ring dimension. This is because of the fact that re-encryption key consists of two matrices which individually consists of an encryption of secret key. Comparing two different ring dimensions we can also infer that as the ring dimension doubles we expect encryption, re-encryption key generation and re-encryption to scale by the same factor. As described earlier, we can benefit further by reducing the size of the ciphertext matrix by using a relinearization window higher than 1. When compared with BV-PRE and

NTRU-ABD-PRE, Ring-GSW-PRE runtimes are significantly higher because of the matrix structure of ciphertext.

3.9.3 Multi-Hop Re-Encryption

Table 3.12 illustrates the dependence of runtime, throughputs, and ciphertext expansion factors on the number of re-encryption hops for PRE-BV with security parameter $\delta \leq 1.006$. The results for NTRU-ABD-PRE are not listed because the scheme supports only two re-encryption hops with the second hop requiring more bits k to represent the ciphertext modulus q, as seen in Table 3.3. It can be seen that PRE-BV scales well with re-encryption depth. For the first 20 hops, the runtime and throughput metrics are approximately the same for encryption and decryption operations, and degrade by at most 20% for the re-encryption operation. For larger re-encryption depths (up to 100), the encryption/decryption throughputs degrade only by at mosty 20% as compared to the single-hop case while re-encryption throughput declines by 40%. It should be noted that the observed enryption/decryption times are still under 2 ms, which may be adequate for many practical applications.

The runtimes for first re-encryption hop and last re-encryption hop are essentially the same for all re-encryption depths, with the latter being slightly lower due to local caching effects of the implementation. The decryption times after regular encryption and proxy re-encryption are approximately the same.

We compared the number of hops of re-encryption and depth of computation after re-encryption of Ring-GSW PRE with BV-PRE [PRSV17] scheme instantiated with same set of parameters. From Figure 3.2, we can see that both the PRE schemes have the ability to perform multiple re-encryptions with Ring-GSW PRE scheme exceeding the number of hops by a large margin. In the next Figure 3.3, we can observe that after a re-encryption operation Ring-GSW scheme still has the ability

Table 3.12 Dependence of Performance Metrics for BV-PRE Encryption, Decryption, and Re-encryption Operations on the Number of Re-encryption Hops d at r=1 and p=2

Configuration			Runtime					Throughput		
d	n	k	**Enc (ms)**	**Dec before ReEnc (ms)**	**First ReEnc (ms)**	**Last ReEnc (ms)**	**Dec after ReEnc (ms)**	**Enc (Kbps)**	**ReEnc (Kbps)**	**Dec after last ReEnc (Kbps)**
1	512	17	0.85	0.76	11.77	–	0.76	604.37	43.51	674.62
2	512	18	0.86	0.77	12.84	12.82	0.77	597.52	39.88	667.48
5	512	19	0.85	0.76	13.74	13.43	0.76	604.31	37.27	672.78
10	512	19	0.85	0.77	13.56	13.57	0.76	602.13	37.75	670.37
20	512	20	0.84	0.76	13.69	13.69	0.75	611.51	37.40	681.35
50	1024	22	1.83	1.67	33.98	33.60	1.67	560.44	30.13	613.60
100	1024	23	2.00	1.87	39.45	39.38	1.86	512.11	25.96	550.19

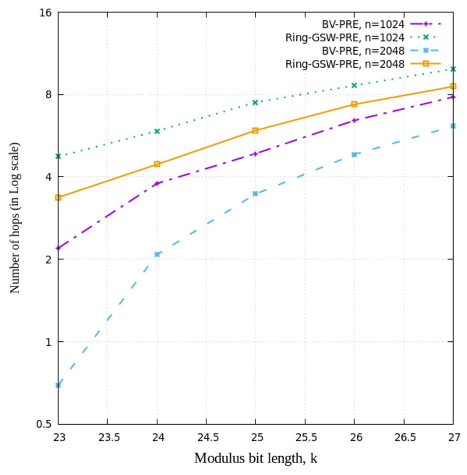

Figure 3.2 Comparison of BV-PRE and Ring-GSW-PRE scheme on multi-hop capability, $p = 5$, $r = 1$.

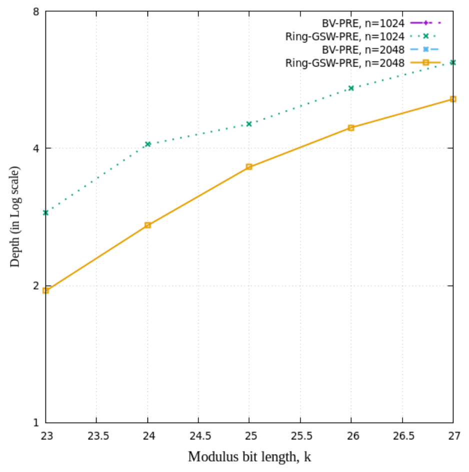

Figure 3.3 Comparison of BV-PRE and Ring-GSW-PRE scheme on depth of computation after re-encryption, $p = 5$, $r = 1$.

to perform a large number of computations owing to its asymmetric noise growth property. In comparison, BV FHE scheme can only perform a single homomorphic multiplication for the bit lengths shown in the figure. This shows robustness of Ring-GSW scheme and viability in practical applications where computations maybe be needed along with re-encryptions.

3.10 Application

A major security challenge for Pub/Sub systems is confidentiality of information which is distributed by the Pub/Sub broker. Existing Pub/Sub systems protect information payloads via encryption that requires either: 1) the publisher and subscriber coordinate to establish the encryption and decryption keys or 2) the Pub/Sub broker decrypts the information payloads from the publishers and then encrypts this information payload again for re-transmission to the subscribers. The first solution contradicts one of the goals of Pub/Sub systems, i.e., the decoupling of publishers and subscribers. The second solution solves this issue, but gives the broker access to the unprotected information. Thus, it makes the broker a ripe target for adversaries to compromise and steal sensitive information.

PRE is a natural fit to support publish-subscribe because PRE maintains data confidentiality even when the broker is compromised and an adversary obtains all re-encryption keys and observes all communications between the publisher, broker and subscriber. These features reduce the need for special, difficult to use security-enabled hardware and software for high-assurance applications, such as in military settings. A compromised PRE-enabled broker would at most allow the adversary to learn which subscribers are allowed to receive information from which publishers based on the existence of re-encryption keys.

In this section, we are particularly interested in understanding how to parameterize the PRE schemes for three application use cases to illustrate the adaptability of our design and implementation.

3.10.1 Enterprise Security

PRE could be very useful in enterprise-style computing environments such as for medical file sharing. Enterprise environments are characterized by high resource availability - both computational power at the publishers, subscribers and PRE servers, but also bandwidth availability. The primary concern would be overall throughput.

For single-hop applications, the goal is to maximize re-encryption throughput. As Tables 3.10 and 3.11 suggest, re-encryption throughput can be maximized by increasing the key swtching window r or increasing the plaintext modulus p (up to certain limits, until the ciphertext modulus bit length and ring dimension start to significantly slow down the runtime). In the case of the BV-PRE scheme, the plaintext throughputs can reach 250 Kbps. The combined effect of increased plaintext modulus and key switching window can produce even higher plaintext outputs but this analysis should be performed based on the requirements of a specific application. The BV-PRE scheme can also provide a multi-hop capability without significantly increasing parameter requirements if the value of key switching window r in expression (3.8) does not exceed 8.

3.10.2 Embedded Support

At the opposite end of the resource availability spectrum is the use case of embedded sensors that collect, encrypt and publish data to a PRE server. To set up the environment, point-to-point communication approvals need to be established, namely that:

- The sensors would need to have appropriate encryption keys.
- The sensors would need to be paired with the PRE server.
- The approval for subscribers to receive data would need to be received to approve the generation of a re-encryption key hosted at the PRE server.

PRE addresses the above measures to encrypt data at the sensor, transmit the data to a cloud storage environment where processing is done, and the encrypted results shared with intended recipients, without ever decrypting the data or sharing decryption keys. Recent results [LSSR$^+$15, BGG$^+$16] show that it is possible to implement public key lattice encryption schemes, very similar to our PRE schemes, and run them on very resource-limited devices, inlcuding devices using 8-bit AVR processors [LSSR+15]. These results also provide general design guidelines to port our designs into limited hardware.

Because embedded use cases require computationally intense operations at low-powered sensor nodes, encryption throughput is paramount. It is feasible that multi-hop encryption would be needed so that the encrypted information can aggregate from the sensors to local PRE servers which send data to a centralized encrypted information clearinghouse. In this situation, the use of BV-PRE with $r = 1$ and a large plaintext modulus, for example, $p = 65536$, would maximize encryption throughput.

An alternative formulation of this use-case for especially low-powered sensor devices might rely on processors with 32-bit words, or less. In this scenario it is generally important for modulus bit-widths to be within a power-of-2 rather than without for increased performance. If the modulus bit-width is larger than bit-width of the processor, then extra shuffling of data and at least a factor-of-2 decrease in performance is likely to result. Selecting BV-PRE at $n = 512$, $r \in (1, 2, 4)$, and a ciphertext modulus bit-width of 17-18 bits is recommended to maximize encryption throughput. It should be noted that the ciphertext bit-width of up to 20 and ring dimension of 512 can support up to 23 re-encryption hops of BV-PRE at $p = 2$ and $r = 1$, as can be seen from Tables 3.6 and 3.12.

3.10.3 Hybrid Deployment with AES

This work is motivated by the problem of sharing data across coalition partners who do not interact directly, including across administrative boundaries, yet want to control data access within each coalition partner by policy. While encryption and policy enforcement solutions are available, a major challenge is the lack of suitable techniques to generate or share encryption keys. For example, streaming video, images and text data are often transmitted when encrypted by AES, because AES is considered both secure and highly efficient. PRE can be used in these scenarios as an AES key distribution mechanism.

Single-hop application operation of PRE would provide the most control for users to limit the spread of restricted data. Based on the RLWE security constraint and PRE correctness constraint, we should keep q as small as possible to guarantee correctness and use the lowest value of n that satisfies the security requirements.

3.11 Conclusion

In this chapter, we present four new lattice-based PRE schemes. We experimentally evaluate the performance of the PRE schemes. Our lattice encryption library is an important aspect of our implementation performance in that its modularity and extensibility enables us to further improve performance with either improved mathematical libraries or even hardware acceleration as these technologies become available.

A benefit of our PRE approach is that it supports applications on commodity computing hardware and improves the overall security of information sharing in practical pub/sub systems. Taken together, this could greatly reduce the operational costs of highly regulated industries such as health-care where regulatory compliance restricts the ability to outsource computation to low cost cloud computing environments.

Although we have focused our discussion on PRE for situations with one producer (Alice) and one consumer (Bob), there is no theoretical limit to the number of producers and subscribers that can be used for PRE operations. With PRE we can support general many-to-many operations where data from many producers is securely shared with many consumers through the PRE prototype, by generating multiple re-encryption keys, one for every permitted publisher-subscriber information sharing pair. A possible approach to address scalability is to distribute the operations of the PRE servers across many computation nodes, and we seek to address this in follow-on research.

CHAPTER 4

EFFICIENT AND SCALABLE BOOTSTAPPING OF BV-LWE FHE SCHEME

4.1 Introduction

In his breakthrough seminal work, Gentry [Gen09] showed the first theoretical construction of a fully homomorphic encryption (FHE) scheme. The idea first introduced as "privacy homomorphism" by Rivest *et al.* [RAD$^+$78] had the capability to outsource computation and get back encrypted results without revealing any information to the outsourcing party. In recent years it has been shown through a series of work that FHE can be pragmatically deployed for a wide variety of applications such as large scale statistical analysis [WH12], spam filtering [KGV16], machine learning [BLN14, BHHH19, CGH$^+$18] etc.

Gentry's initial construction of FHE scheme was based on ideal lattices however, initial implementation of the scheme [GH11] revealed an impractical runtime in the order of seconds to minutes. Since then many FHE schemes have been developed [DGHV10, SV10, BV14a, BV14b, BGV14, GSW13] which are much efficient, have shorter memory footprints and based on hardness assumption of worst case lattice problems.

Almost all of the above mentioned schemes are based on the "noisy" encryption technique as they base security on hardness assumption of LWE and its variant RLWE. In other words, a small amount of noise or error is added to the ciphertexts for the security of the scheme. Evaluation of functions on ciphertexts of these schemes leads to asymptotic noise growth. Furthermore, noise growth of the ciphertexts is directly proportional to the depth of computation. Once the noise contained in ciphertext reaches a certain threshold there is no room for further homomorphic evaluation. Any homomorphic operation beyond this point amounts to incorrect result upon decryption. Such encryption schemes are said to be ”somewhat homomorphic” (SHE) encryption schemes, a choice by design. Following Gentry's blueprint, these schemes can be then converted into a fully homomorphic scheme by Bootstrapping. In a nutshell, Bootstrapping implies evaluating the decryption circuit of the scheme via homomorphic operations. As a result, Bootstrapping is possible if and only if the homomorphic capacity is higher than the depth of decryption circuit of the scheme.

Bootstrapping algorithms demonstrated in cryptology literature can be broadly classified into two body of work. In the first line of work, bootstrapping algorithms [GHS12a, HS15, CH18] are applicable to FHE schemes where multiple messages can be potentially packed into a ciphertext. Such bootstrapping algorithms are often associated with very large runtimes, typically in the order of minutes, however their amortized runtime is quite comparable to other LWE based bootstrapping approaches. In the second line of work, bootstrapping algorithms [DM15, BR15, CGGI16, BDF18, MP20] work on FHEW-like cryptosystems where ciphertexts are in present in LWE form, typically encrypt single bit messages and homomorphic decryption is performed via *Ring-GSW* FHE scheme. In practice, this family of bootstrapping algorithm performs much faster than the former with runtimes in the order of milliseconds.

In this work, our goal is to bridge this gap between the two family of bootstrapping algorithms by supporting larger plaintext modulus and arbitrary depth of computation before the need to bootstrap. We present techniques for bootstrapping *BV-LWE* [BV14a] and *BV-GSW* FHE scheme in a symmetric key setting. To begin with, *BV-LWE* FHE scheme allows elementary operations of addition and multiplication on encrypted data. Security of the scheme is based on learning with errors (LWE) assumption which is known to be reducible to solving hard problems in general lattices. A message $m \in \mathbb{Z}_p$ can be encrypted in the scheme where $p\,(\geq 2)$ is a relatively small plaintext modulus. This work advances on other LWE cryptosystems such as FHEW [DM15] and TFHE [CGGI16] that can only encrypt binary messages and evaluate arbitrary boolean circuits. One of the similarity found in both these schemes is that the noise and plaintext modulus evolve after evaluation of a boolean gate. Bootstrapping the ciphertext then restores it back to original plaintext modulus and noise levels. In contrast to this, plaintext modulus in *BV-LWE* scheme remains unaffected by homomorphic operations and consequently we can evaluate circuits till the depth of instantiated parameters allow.

One of the drawbacks of *BV-LWE* scheme is that multiplication of ciphertexts leads to quadratic blow up in ciphertext size ($n + 1$ to $n^2/2$ roughly). Application of re-linearization technique then reduces the ciphertext size back to $n + 1$ however, at the cost of publishing encryptions of all the quadratic terms as well as individual terms of the secret key. We remedy this problem in a simple and effective way by adding *GSW* extensions to the original *BV-LWE* ciphertext of the form $c = (\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle + pe + m) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. The outcome of this modification leads to *BV-GSW* FHE scheme where noise grows asymmetrically instead of quadratically in LWE dimension and hence larger depth of computations can be evaluated. Naturally, *BV-GSW* FHE scheme inherits all the advantages of *GSW*[GSW13] FHE scheme. We summarize our contributions as follows:

• **Larger plaintext modulus**: Following approaches presented in [ASP14, DM15, CGGI16] we demonstrate bootstrapping algorithms that are applicable to *BV-LWE* and *BV-GSW* FHE schemes and work with plaintext modulus, $p \geq 2$. In these set of algorithms we retain the usage of ternary or root of unity encoding where an integer (mod-q) is represented as a polynomial in $\{-1, 0, 1\}$. The core of our *BV-GSW* bootstrapping technique relies on a RLWE coefficient extraction procedure on polynomial rings. We show an efficient implementation of this extraction procedure which uses $\log_2(n)$ Automorphism operations without introducing any significant noise growth.

• **Arbitrary secret key**: In addition to secret keys generated from binary ($\mathcal{B}$) distributions, we extend our bootstrapping algorithms to work efficiently with secret keys generated from ternary ($\mathcal{T}$) or discrete Gaussian (χ_e) distributions. In other words, our bootstrapping algorithms meets the specification of *Homomorphic Encryption standardization* document [ACC$^+$18] which suggests the usage of Gaussian distribution over the interval $\{\pm 8\}$ for generation of secret key. To work with arbitrary secret key distributions our bootstrapping algorithms uses Automorphism maps instead of Galois field isomorphisms which preclude the necessity of generating lookup tables and lead to efficient generation of bootstrapping keys. Lastly, we discuss memory trade-offs in bootstrapping key generation that can reduce the bootstrapping key size further.

• **Arbitrary ciphertext modulus and Gridstrapping**: We introduce a new bootstrapping procedure which works on a large finite field without increasing the ring dimension of *Ring-GSW* FHE scheme. The finite field is described as a multi-dimensional grid or matrix where the position of "1" indicates the integer value. Contrary to our first bootstrapping procedure we encode integers as binary $\{0, 1\}$ polynomial and represent grid pointers as ciphertexts. This finite field can be scaled further by increasing the grid-dimension parameter ζ (for a two dimensional grid or matrix $\zeta = 2$).

Chapter Organization: We start by describing some of the earlier works done on Bootstrapping in Section 4.2. In Section 4.4 we present *BV-GSW* scheme in symmetric key setting. Section 4.5 outlines the *Ring-GSW* scheme along with an analysis on noise growth of homomorphic operations. We use *Ring-GSW* FHE scheme as well as BV FHE scheme as primary tools for bootstrapping *BV-LWE* and

BV-GSW FHE schemes. In Section 4.6 we present our bootstrapping procedures while Section 4.7 shows extension of bootstrapping technique to large modulus sizes.

4.2 Related Work

In [BV14b], Brakerski and Vaikuntanathan describe a generalized bootstrapping procedure for LWE based cryptosystems. The central idea in their construction rests on the application of Barrington's [Bar89] circuit sequentialization theorem which allows to transform any depth d circuit into a polynomial length, width-5 permutation branching program. Evaluation of the circuit translates to sequential homomorphic multiplication of ℓ 5-by-5 encrypted permutation matrices where ℓ is the length of the branching program. Next, to show that such evaluations can be performed within polynomially bounded noise in security factor the authors describe a new LWE based FHE scheme roughly similar to GSW FHE scheme. Finally, they reduce the ciphertext with successive application of *dimension-modulus* reduction technique and prove that the bootstapping technique meets the optimal approximation factor for lattice problems under quantum reductions. This theoretical bootstrapping technique appears to be satisfactory but its implementation would lead to very large runtimes because of the hidden constant factor in run-time complexity and $\mathcal{O}(n^3)$ complexity of GSW FHE operations.

To reduce the time complexity and approximation factor of [BV14b], Sheriff and Peikert introduced a new bootstrapping algorithm [ASP14] which works on injective homomorphism of permutation matrices and removes the necessity to transform decryption circuit into boolean circuit. More specifically, their bootstrapping algorithm generalizes the LWE decryption circuit as a rounded inner product between secret key vector and binary ciphertext. The bootstrapping key consists of encryption of each coordinate of the secret key vector after mapping it to a binary indicator vector. To evaluate the decryption circuit as an arithmetic function, the authors used the additive embedding property of permutation matrices to a finite field element in $\mathbb{Z}_q$. To evaluate an addition, the indicator vector needs to be expanded into a permutation matrix and then multiplied with another indicator vector leading to $\mathcal{O}\left(q^2\right)$ homomorphic operations, q being the ciphertext modulus. After the evaluation of inner product, rounding function is performed through a homomorphic equality test. For ciphertext evaluations the authors use GSW FHE scheme because of its quasi-additive error growth property and show the total number of homomorphic computations to be $\tilde{\mathcal{O}}\left(\lambda\right)$ where λ is the security factor. However, no direct implementation of this algorithm has been reported in literature.

Building upon the work of [ASP14], Ducas and Micciancio [DM15] demonstrated the first concrete implementation of a bootstrapping technique on a LWE based FHE scheme capable of evaluating boolean functions. In their work, the authors eliminated the need to map integers to permutation matrices by utilizing the finite field properties of cyclotomic polynomials. A major outcome of their work is in demonstrating that cyclic groups can be directly embedded into cyclotomic polynomials by encoding the cyclic group $\mathbb{Z}_q$ into the group of roots of unity: $i \rightarrow X^i$. Further the isomorphic operations of bootstrapping can be directly evaluated using the appropriate homomorphic operations of a RLWE based FHE scheme. In their implementation, they instantiated the RLWE scheme with Ring-GSW FHE scheme and reported a bootstrapping runtime under a second for a small parameter set. A major bottleneck in their implementation is the updating procedure of homomorphic accumulator which stores the intermediate ciphertext during bootstrapping. To accelerate this updating procedure bootstrapping key needs to be represented as a lookup table with a base, $b > 2$ which increases the size of bootstrapping key roughly by a factor $b/\log_2 b$ and $q/\log_2 q$ in the extreme case. In contrast to this, our bootstrapping procedure doesn't incur such memory overheads as homomorphic accumulator is always updated in a single step via Automorphism transforms.

In another arithmetic bootstrapping technique, TFHE, Chillotti *et al.* [CGGI16] takes a different approach. In particular, the authors follow the generalized bootstrapping approach based on mux gates discussed in [GINX16]. In their implementation, the underlying arithmetic element is Torus, defined over real numbers modulo 1. Further, the FHE schemes used in their work are scale invariant Torus analogues of LWE, BV FHE [BV11b] and *Ring-GSW* FHE [KGV16] schemes. In comparison to FHEW, TFHE bootstrapping runtime and bootstrapping key size is significantly reduced. This is mainly because two reasons: **1**. Use of binary mux gates for updating the FHEW accumulator. **2**. Use of matrix-vector multiplications (external product) of ciphertexts instead of matrix multiplication of polynomial ciphertexts (internal product). One of the drawbacks of TFHE scheme is that the usage of binary mux gates places a constraint on generation of secret keys from a binary distribution. TFHE bootstrapping can still be adapted to arbitrary secret keys as shown in [MP20] however, this approach scales the bootstrapping runtime and evaluation key size linearly with the number of bits in secret keys.

A more rigorous and practical comparison between FHEW and TFHE bootstrapping procedures was presented in [MP20]. For direct comparison between both the schemes, the authors chose a unified approach by implementing them in PALISADE crypto software library using modular arithmetic. Further, the TFHE scheme was extended to support ternary and arbitrary secrets. It was found that TFHE is faster than FHEW roughly by a factor 2 when working with binary secret keys. For ternary secret keys, performance of both the bootstrapping procedures are nearly equivalent. However, for the case of arbitrary secret keys FHEW bootstrapping outperforms TFHE in terms of running time. In these evaluations, it can be noted that a large base ($b \approx q$) is used resulting in larger bootstrapping key. Our bootstrapping procedure is a variation of the FHEW scheme however works without any dependence on a base decomposition parameter b. As such, when working with ternary and arbitrary secret keys we achieve similar or better performance in runtimes when compared with TFHE scheme. Further, we achieve these results without any expansion in bootstrapping key size.

In a separate work, [BR15] presented a generalization of FHEW bootstrapping procedure by extending it to larger plaintext modulus $t > 2$. It should be noted that here the underlying LWE scheme is same as that used in FHEW and ciphertexts are encryption of binary messages. The extended modulo t allows the computation of multiple gates or gates with several inputs and outputs, and hence, amortizes the cost of single gate FHEW bootstrapping. The number of gates that can be evaluated before bootstrapping is roughly equal to the parameter t. One of the drawbacks of this scheme is usage of power-of-p cyclotomics that is required in accumulator operations. As non power of 2 cyclotomics are known to incur extra overhead in computation of FFTs, bootstrapping runtime of [BR15] is significantly higher. In comparison to [BR15], our bootstrapping procedures work on a LWE scheme which directly supports encryption of integer rather than binary messages and still retain the usage of power of 2 cyclotomics. Additionally, finite field arithmetic are supported by our schemes and have the ability to amortize cost of bootstrapping by instantiating larger ciphertext modulus.

In other related work, improvement of FHEW bootstrapping scheme has been pursued in [MS18] to reduce the amortized runtime by packing multiple bits in a ciphertext, extension of FHEW scheme to larger gates has been shown in [BDF18] by CRT polynomial tensoring techniques.

4.3 Design

4.3.1 Syntax of a Fully Homomorphic Encryption Scheme

A non-interactive FHE scheme is an ensemble of PPT algorithms $\Pi = (\mathit{ParamsGen}$, KeyGen, $\mathit{Encrypt}$, $\mathit{EvalAdd}$, $\mathit{EvalMult}$, $\mathit{Decrypt}$, $\mathit{BootstrapKeyGen}$ and $\mathit{Bootstrap})$ described as follows:

- **ParamsGen**$\left(1^{\lambda}\right)$: It takes in the security parameter λ and returns the corresponding public parameters pp.
- **KeyGen**$\left(pp, 1^{\lambda}\right)$: KeyGen takes the public parameters pp and returns the public key and secret key pair (pk, sk).
- **Encrypt**(pp, pk, m): Given public key pk, public parameters pp and message $m \in \mathcal{M}$, it encrypts the message m and returns a ciphertext c.
- **EvalAdd**(pp, c_0, c_1): Given two ciphertexts c_0 and c_1 encrypting messages $m_0, m_1 \in \mathcal{M}$, EvalAdd produces a ciphertext c_{add} encrypting message $m_0 + m_1$.
- **EvalMult**(pp, c_0, c_1): Given two ciphertexts c_0 and c_1 encrypting messages $m_0, m_1 \in \mathcal{M}$, EvalMult produces a ciphertext c_{mult} which encrypts the message $m_0 \cdot m_1$.
- **Decrypt**(pp, sk, c) Decrypt recovers the message m from the ciphertext c.
- **BootstrapKeyGen**$(pp_{\Pi'}, pp_{\Pi}, pk_{\Pi'}, sk_{\Pi})$: Given a FHE scheme Π' where Π' may or may not be equal to Π, BootstrapKeyGen generates an evaluation key, bk which consists of encryptions of secret key sk under the public key of scheme Π'.
- **Bootstrap**$(pp_{\Pi'}, bk, c)$ Given a bootstrapping key bk and ciphertext $c \in \Pi$, Bootstrap generates a ciphertext $c' \in \Pi$ with reduced noise that c.

4.4 BV-GSW FHE Scheme

BV-GSW [BV14a] is a symmetric-key encryption scheme and its security is based on standard LWE [Reg09] assumption. To circumvent the generation of $\mathcal{O}\left(\lambda^2\right)$ keys and quadratic noise growth, we present a modified encryption scheme by adding *GSW* [GSW13] extensions. The scheme has a message space of $\mathcal{M} \in \mathbb{Z}_p$ where p is the plaintext modulus. The scheme is parameterized by:

- Security parameter λ and dimension $n = n\left(\lambda\right)$.
- Working modulus q of $\kappa = \kappa\left(\lambda, d\right)$ bits and plaintext modulus p.

- maximum depth of evaluation, d.
- Discrete random uniform noise generator $\mathbf{x} \leftarrow_{\$} \mathcal{U}_q^n$ and B-bounded discrete Gaussian noise generator $\mathbf{y} \leftarrow_{\$} \chi_B^\ell, \mid \mathbf{x} \in \mathbb{Z}_q^n, \mathbf{y} \in \mathbb{Z}_q^\ell$.

  The scheme is an ensemble of following operations:

- **ParamsGen**$\left(1^\lambda\right)$: Choose positive integers q, n and p such that $p > 2$, $\gcd(p, q) = 1$, $q \gg p$ and $n = (\lambda)$. Set $N = (n+1)\,\ell$ and $\ell = \lceil \log q \rceil$. Return public parameter set $pp = (p, q, n, \ell, N)$.
- **SecretKeyGen**(pp): Generate $\mathbf{s} \leftarrow_{\$} \chi_B^n$ from discrete Gaussian distribution and set $sk = \mathbf{t} = (1, -\mathbf{s}) \in \mathbb{Z}_q^{n+1}$.
- **Encrypt**(pp, m, sk): To produce an encryption of message $m \in \mathbb{Z}_p$ we produce a LWE matrix, $\mathbf{A}$ and simply add $m \cdot \mathbf{G}$ to it. To generate **LWE** matrix $\mathbf{A}$ first we sample uniformly random matrix $\mathbf{B}$, binary random matrix $\mathbf{R}$ and discrete Gaussian noise vector $\mathbf{e}$ and set $\mathbf{b} = \mathbf{Bs} + p \cdot \mathbf{e}$. LWE matrix $\mathbf{A}$ is a concatenation of $\mathbf{b}$ and $\mathbf{B}$. This procedure is shown as:

$$\mathbf{B} \leftarrow_{\$} \mathcal{U}_q^{N \times n},\ \mathbf{R} \leftarrow_{\$} \mathcal{U}_{\{0,1\}}^{N \times N},\ \mathbf{e} \leftarrow_{\$} \chi_B^N,\ \mathbf{b} = \mathbf{Bs} + p \cdot \mathbf{e},\ \mathbf{A} = [\mathbf{b} \mid\mid \mathbf{B}]$$
$$\mathbf{C} = m \cdot \mathbf{G} + \mathbf{RA} \in \mathbb{Z}_q^{N \times (n+1)},\ c \leftarrow \mathbf{C}$$

- **EvalAdd**(c_0, c_1): Homomorphic addition is given by addition of the input matrices represented by c_0 and c_1.

$$\mathbf{C}_{add} = \mathbf{C}_0 + \mathbf{C}_1,\ c_{add} \leftarrow \mathbf{C}_{add}$$
$$\mathbf{C}_{add} = (m_0 + m_1) \cdot \mathbf{G} + \mathbf{R}_{add}\mathbf{A}_{add},\ \mathbf{R}_{add}\mathbf{A}_{add} = \mathbf{R}_0\mathbf{A}_0 + \mathbf{R}_1\mathbf{A}_1$$

- **EvalMult**(c_0, c_1): Homomorphic multiplication of ciphertexts is achieved by first bit decomposing the first ciphertext and then we proceed to multiply it with the second ciphertext. It is shown as:

$$\begin{aligned}
\mathbf{C}_{mult} &= \mathrm{G}^{-1}(\mathbf{C}_0) \cdot \mathbf{C}_1 = \mathrm{G}^{-1}(\mathbf{C}_0) \cdot (m_1\mathbf{G} + \mathbf{R}_1\mathbf{A}_1) \\
&= m_1 \cdot \mathbf{C}_0 + \mathbf{G}^{-1}(\mathbf{C}_0)\,\mathbf{R}_1\mathbf{A}_1 \\
&= m_0 m_1 \cdot \mathbf{G} + \left(m_1\mathbf{R}_0\mathbf{A}_0 + \mathbf{G}^{-1}(\mathbf{C}_0)\,\mathbf{R}_1\mathbf{A}_1\right) \\
&= m_0 m_1 \cdot \mathbf{G} + \mathbf{A}_{mult}
\end{aligned}$$

- **Decrypt**(sk, c): We recover message m' by performing the following two operations:

  1. Compute $\mathbf{w} = \mathbf{Ct} \in \mathbb{Z}_q^N$, Note that,
     $\mathbf{w} = m \cdot \mathbf{g} \otimes \mathbf{t} + p \cdot \mathbf{e}$.
  2. Since, the first ℓ rows of $\mathbf{g} \otimes \mathbf{t}$ are powers of two we output $m' = w_0 \bmod p$

4.4.1 Correctness Constraints

In order to correctly recover the message, we have to ensure that there in no wrap around mod-q. Since, our working modulus range is $\left[-\frac{q}{2}, \frac{q}{2}\right]$, noise has to be strictly within this bound. We derive noise bounds as follow:

$$\|w_0\| < q/2, \text{ or } \|m + pe\| < q/2$$

In case of homomorphic multiplication, we derive noise bound as follows:

$$\begin{aligned}
\mathbf{w} = \mathbf{C}_{mult}\mathbf{t} &= m_{mult} \cdot \mathbf{Gt} + \mathbf{A}_{mult}\mathbf{t} \\
&= m_{mult} \cdot \mathbf{g} \otimes \mathbf{t} + m_1\mathbf{A}_0\mathbf{t} + \mathbf{G}^{-1}\left(\mathbf{C}_0\right)\mathbf{A}_1\mathbf{t} \\
&= m_{mult} \cdot \mathbf{g} \otimes \mathbf{t} + pm_1\mathbf{e}_0 + p\mathbf{G}^{-1}\left(\mathbf{C}_0\right)\mathbf{e}_1
\end{aligned}$$

Using central limit heuristics on noise bounds, we can correctly decrypt for the following condition:

$$\begin{gathered}
\|w_0\| = p\left(m_1 \|e_0\|_\infty + \sqrt{N}\,\|\mathbf{e}_1\|_\infty\right) < q/2 \\
\text{or, } q > 2pB\sqrt{N}
\end{gathered}$$

where B is bound for B-bounded discrete Gaussian generator.

4.4.2 Modulus Switching

(Adapted from [DM15]) We use modulus switching operation not only as a noise reduction technique but also to align the current parameters with the Bootstrapping scheme. Given a ciphertext c with modulus q we can transform the ciphertext into c' with modulus q'. To switch modulus we use the following rounding function $[x]_{q \to q'}$

which is defined as:

$$[x]_{q\to q'} = \beta + \gamma,\ \beta = [q'x/q],\ \gamma = (x - \beta) \mod p$$

We apply this scaled rounding function to every entry of the ciphertext matrix to complete the modulus switching procedure.

$$\textbf{ModSwitch}\,(c) = \left[\mathbf{C}_{i,j}\right]_{q\to q'},\ i \in [0, N),\ j \in [0, n]$$

Since, $\gamma \in [0, p-1]$ and integer rounding is uniformly distributed in $[\frac{-1}{2}, \frac{1}{2}]$ the rounding error is a Gaussian distribution with a variance of $\sigma^2 = \frac{p^2-1}{12}$.

More specifically, we use modulus switching prior to initiating Bootstrapping procedure with modulus parameter $q' = 2^{\lfloor \log q \rfloor}$. Furthermore, we use only the first row of ciphertext matrix to "align" with Bootstrapping parameters.

4.4.3 Key-Switching

Key-switching operations transform a LWE ciphertext $\mathbf{C}$ encrypted under a key $\mathbf{s} \in \mathbb{Z}_q^n$ to be transformed into another ciphertext $\mathbf{C}^*$ which can be decrypted under a new key $\mathbf{s}^*$. To facilitate this transformation process, we generate an evaluation key, *ek* which consists of a sequence of $(n+1)$ matrices, $\mathbf{EK}[i],\ i \in [0, n]$. These matrices are then used in the re-encryption phase to complete the transformation process. We segregate the key-switching process into *KeySwitchGen* and *SwitchKey* procedures which are described as follows:

KeySwitchGen(pp, sk, sk^*): To generate the matrices $\mathbf{EK}[i]$, first we generate $\mathbf{B}_i$ from discrete uniform distribution $\mathcal{U}_q^{N\times n}$ and set b_i using the new secret key s^*.

Next, we add a shifted column matrix of the old secret key. This is described as:

$$\text{Sample } \mathbf{B}_i \leftarrow_{\$} \mathcal{U}_q^{N\times n},\ \mathbf{e}_i \leftarrow_{\$} \chi_B^N$$

$$\text{Set } \mathbf{b}_i = \mathbf{B}_i s^* + p\cdot \mathbf{e}_i,\ \mathbf{A}_i = [\mathbf{b}_i \mid\mid \mathbf{B}_i]$$

$$\mathbf{EK}[i] = \mathbf{A}_i + (\mathbf{g}\otimes \mathbf{t} \gg i)\,, ek = \{\mathbf{EK}[\mathbf{0}], \cdots, \mathbf{EK}[n]\}$$

SwitchKey$(pp, \mathbf{C}, ek)$: This results in a ciphertext $\mathbf{C}^*$ and the procedure is described as follows:

$$\mathbf{C}^{*,i} = \mathbf{G}^{-1}\left(\mathbf{C}^{\text{top}}\right)\cdot \mathbf{EK}[i] \in \mathbb{Z}_q^{\ell\times n+1}$$

$$\mathbf{C}^* = \left[\mathbf{C}^{*,0} \mid\mid^{\intercal} \cdots \mid\mid^{\intercal} \mathbf{C}^{*,n}\right] \in \mathbb{Z}_q^{N\times n+1}$$

Here $\mathbf{C}^{\text{top}}$ represent the top-ℓ rows of the ciphertext matrix $\mathbf{C}$.

4.5 Ring-GSW Bootstrapping Scheme

Ring-GSW encryption scheme is a RLWE [LPR10, KGV16] adaptation of the GSW [GSW13] FHE scheme based on LWE security assumption. We primarily use it for bootstrapping or evaluating the decryption circuit of *BV-LWE* or *BV-GSW* scheme. However, it can be noted that leveled *BGV* [BGV14] FHE scheme can also be used instead but at a greater expense of noise growth and larger modulii.

The scheme has a message space of $\mathcal{M} \in R_p$ where $p > 2$. Construction of the scheme is based on underlying cyclotomic polynomial arithmetic and parameterized by:

- Security paramter λ and ciphertext modulus q.
- Plaintext modulus p such that $2 < p \ll q$ and ring dimension n.
- B-bounded discrete Gaussian distribution χ_B.
- Ternary distribution $\mathcal{T}$ which generates a ring polynomial with coefficients uniformly sampled from $\{-1, 0, 1\}$.

- Discrete uniform distribution $\mathcal{U}_{R_q}$.
- Maximum depth of homomorphic evaluation d.

Ring-GSW scheme encapsulates the following operations:

- **ParamsGen**$\left(1^\lambda\right)$: Choose positive integers $q = q(\lambda, d)$ and $n = n(\lambda)$. Return parameter set $pp = (\ell, N, p, q, n)$ where $\ell = \lceil \log q \rceil$ and $N = 2\ell$.
- **SecretKeyGen**(pp): Secret key is a low norm polynomial sampled randomly from discrete Gaussian distribution χ_{R_q}. However, it has been shown that generating secret key from sparse distributions such as binary ($\mathcal{B} \in \{0, 1\}$) [GKPV10, BLP+13] or ternary distributions, $\mathcal{T}$ [Mic18] still leads to instantiation of LWE problem that are as hard as standard LWE and hence we show secret key generation using ternary distribution, $\mathcal{T}$ as follows:

$$s \leftarrow_{\$} \mathcal{T}_{R_q},\ sk = [1; -s] \in R_q^{2\times 1}$$

- **PublicKeyGen**(pp, sk): Public key, pk is a RLWE pair (b, a) represented as a matrix $\mathbf{A}$. To generate b, we sample polynomials a and e from uniform and discrete Gaussian distributions respectively and proceed as follows:

$$a \leftarrow_{\$} \mathcal{U}_{R_q}, e \leftarrow_{\$} \chi_{B,R_q},\ b = as + pe$$
$$\text{Set public key, } pk = \mathbf{A}^{1\times 2} = [b\ a]$$

- **Encrypt**(pp, pk, m): For encryption we sample a random matrix $\mathbf{R}$ from uniform ternary distribution $\mathcal{T}_{R_q}^{N}$ and an error matrix $\mathbf{E} \in R_q^{N\times 2}$ from discrete Gaussian distribution. Encryption of message polynomial $m \in R_p$ is then set as follows:

$$\mathbf{R} = (r_0, r_1, \cdots, r_{N-1}) \leftarrow_{\$} \mathcal{T}_{R_q}^{N\times 1},\ \mathbf{E} \leftarrow_{\$} \chi_{B,R_q}^{N\times 2}$$
$$\mathbf{C} = m \cdot \mathbf{G} + \mathbf{R} \otimes \mathbf{A} + p\mathbf{E},\ \text{Set } c \leftarrow \mathbf{C}$$

- **Decrypt**(pp, sk, c): Message m' is recovered by multiplying the first row of ciphertext $\mathbf{C}$ by secret key sk. This is shown as:

$$m' = (\mathbf{C}_0 \times sk \bmod q) \bmod p$$

  Message m' is recovered correctly as long as the coefficients of the noise polynomial t are within modulus $q \left(= [-\frac{q}{2}, \frac{q}{2}]\right)$ where t is given as:

$$t = \mathbf{C}_0 \times sk \bmod q$$
$$\text{For correct decryption, } \|t\|_\infty \leq q/2, \text{ or, } q \geq 2 \cdot \|t\|_\infty$$

Next, we discuss the homomorphic operations of the scheme, namely, **EvalAdd** and **EvalMult** and analyze their noise growth.

EvalAdd(c_0, c_1): Ciphertext addition is quite straightforward. To produce c_{add} we simply add each polynomial entry of c_0 with corresponding entry of c_1. This is shown as:

$$\mathbf{C}_{add} = \mathbf{C}_0 + \mathbf{C}_1, \text{ Set } c_{add} \leftarrow \mathbf{C}_{add}$$

EvalMult(c_0, c_1): Homomorphic multiplication is a two step procedure. In the first step we use the bit decomposition subroutine to expand the first operand matrix. In the next step we multiply this expanded matrix with the second operand. This is shown as:

$$\mathbf{C}' = \mathbf{BitDecomp}\left(\mathbf{C}_0\right) \in R_q^{N \times N}$$
$$\mathbf{C}_{mult} = \mathbf{C}' \cdot \mathbf{C}_1 \in R_q^{N \times 2}, \; c_{mult} \leftarrow \mathbf{C}_{mult}$$

To determine correctness of decryption after homomorphic multiplication, we have to ensure that the noise term t is within the modulus bounds. Noise after homomorphic multiplication is given as:

$$t = \mathbf{C}_{mult,0} \times sk = \mathbf{C}_{mult,0,0} - s \cdot \mathbf{C}_{mult,0,1}$$

Breaking it down further, $\mathbf{C}_{mult,0,0}$ and $\mathbf{C}_{mult,0,1}$ are given by:

$$\begin{aligned}
\mathbf{C}_{mult,0,0} &= \sum_{i=0}^{\ell-1} \left(\alpha_i \cdot \mathbf{C}_{1,i,0} + \beta_i \cdot \mathbf{C}_{1,i+\ell,0}\right) \\
&= \sum_{i=0}^{\ell-1} \alpha_i \left(r_i b + 2^i m_1 + p\mathbf{E}_{i,0}\right) + \beta_i \left(r_{i+\ell} b + p\mathbf{E}_{i+\ell,0}\right) \\
&= \sum_{i=0}^{\ell-1} \left[b\left(r_i \alpha_i + r_{i+\ell}\beta_i\right) + p\left(\alpha_i \mathbf{E}_{i,0} + \beta_i \mathbf{E}_{i+\ell,0}\right)\right] + \alpha m_1
\end{aligned}$$

where α_i and β_i are given by:

$$\alpha_i = \mathbf{BitDecomp}\left(\mathbf{C}_{0,0,0}\right),\ \beta_i = \mathbf{BitDecomp}\left(\mathbf{C}_{0,0,1}\right)$$

where $i \in [0, \ell)$. Similarly,

$$\begin{aligned}
\mathbf{C}_{mult,0,1} &= \sum_{i=0}^{\ell-1} \left(\alpha_i \cdot \mathbf{C}_{1,i,1} + \beta_i \cdot \mathbf{C}_{1,i+\ell,1}\right) \\
&= \sum_{i=0}^{\ell-1} \alpha_i \left(r_i a + p\mathbf{E}_{i,1}\right) + \beta_i \left(r_{i+\ell} a + p\mathbf{E}_{i+\ell,1} + 2^i m_1\right) \\
&= \sum_{i=0}^{\ell-1} \left[a\left(r_i\alpha_i + r_{i+\ell}\beta_i\right) + p\left(\alpha_i\mathbf{E}_{i,1} + \beta_i\mathbf{E}_{i+\ell,1}\right)\right] + \beta m_1
\end{aligned}$$

The noise term, t can then be shown as follows:

$$\begin{aligned}
t = (b - as) \cdot \sum_{i=0}^{\ell-1} \left(r_i\alpha_i + r_{i+\ell}\beta_i\right) + (\beta - s\alpha)\, m_1 + \\
\sum_{i=0}^{\ell-1} p\left[\left(\alpha_i\mathbf{E}_{i,0} + \beta_i\mathbf{E}_{i+\ell,0}\right) - s\left(\alpha_i\mathbf{E}_{i,1} + \beta_i\mathbf{E}_{i+\ell,1}\right)\right] \\
\approx m_0 m_1
\end{aligned}$$

Using central limit heuristics, we finally arrive at the following noise bound and correctness constraint:

$$\begin{aligned}
&\|t\| \approx 6pnB\log_2 q \\
&\therefore q \geq 12pnB\log_2 q, \text{ average case bounds}
\end{aligned}$$

4.5.1 Key-Switching

As defined in Section 4.4, key switching procedure allows one to transform the ciphertext so that it can be decrypted by a more desirable secret key sk^*. The procedure is divided into two phases, a static phase called *KeySwitchGen* where we generate roughly an encryption of the secret key and a dynamic phase *SwitchKey* which is responsible for the ciphertext transformation. Given a *Ring-GSW* ciphertext $\mathbf{C}$ encrypted under a secret key sk we describe the key-switching procedure as follows:

KeySwitchGen(pp, sk, sk^*): : We generate an evaluation key which consists of two RLWE matrices. First, we generate a noise free RLWE pair from sk^*. Next, to generate each of the matrix we generate a random vector from ternary distribution $\mathcal{T}_{R_q}$ and an error matrix from discrete Gaussian distribution and proceed as follows:

$$\begin{aligned}
&\text{Sample } a \leftarrow_{\$} \mathcal{U}_{R_q},\ b = as^*,\ \text{Set } \mathbf{A} = [b\ a] \in R_q^{1\times 2}\\
&\text{Sample } \mathbf{r}_i \leftarrow_{\$} \mathcal{T}_{R_q}^{N},\ \mathbf{E}_i \leftarrow_{\$} \chi_{R_q,B}^{N\times 2}\\
&\mathbf{EK}[i] = \mathbf{r}_i \otimes \mathbf{A} + p\mathbf{E}_i + (\mathbf{PowersOf2}(sk) \gg i)\\
&\text{Set } ek = \{\mathbf{EK}[0], \mathbf{EK}[1]\}
\end{aligned}$$

SwitchKey$(pp, \mathbf{C}, ek)$: This procedure is similar to homomorphic multiplication however only considers the upper ℓ-rows of the input ciphertext for bit decomposition and represented as $\mathbf{C}^{\text{top}}$. We outline the procedure as follows:

$$\begin{aligned}
&\mathbf{C}^{*,i} = \mathbf{BitDecomp}\left(\mathbf{C}^{\text{top}}\right) \cdot \mathbf{EK}[i] \in R_q^{\ell\times 2}\\
&\mathbf{C}^{*} = \left[\mathbf{C}^{*,0} \,||^{\intercal}\, \mathbf{C}^{*,1}\right] \in R_q^{N\times 2}
\end{aligned}$$

Noise growth in a ciphertext after *key-switching* is comparatively lower when compared to that in homomorphic multiplication.

4.6 RLWE Bootstrapping Procedure

In this section, we describe the complete mechanics of bootstrapping *BV-LWE* or *BV-GSW* ciphertexts. For simplicity, we represent the *BV-LWE* and *BV-GSW* ciphertexts in the following form:

$$c = (\mathbf{a}, b) \in \mathbb{Z}_q^{n+1} \text{ where } b = \langle \mathbf{a}, \mathbf{s} \rangle + pe + m$$

We assume that the noise in ciphertext is in the form $pe \in [-q/2, q/2]$ and large in magnitude. Our goal is to homomorphically decrypt this ciphertext and output a *BV-GSW* ciphertext in modified (LWE matrix) form or original LWE form. Recall that the decryption circuit is represented as $b - \langle \mathbf{a}, \mathbf{s} \rangle \bmod q$. This arithmetic circuit can then be transformed in terms of homomorphic encryption $E(\cdot)$ of a scheme Π' as follows:

$$\Delta_q = E(b) - \sum_{i=0}^{n-1} \mathbf{a}_i \cdot E\left(\mathbf{s}_i\right) \bmod q$$

It can be noted that Δ_q may or may not be the final refreshed ciphertext. Further, we need to publish the encryption of the secret key's individual entries $\mathbf{s}_i$ as $E\left(\mathbf{s}_i\right)$. A collection of such encryption of secret key is referred to as *bootstrapping key* and is considered to be secure under hardness assumption of the underlying scheme Π'.

Next, we can observe that such homomorphic decryption can be performed trivially by instantiating an encryption scheme Π' with a large plaintext modulus, p_{RLWE} such that $p_{\text{RLWE}} = q_{\text{BV-LWE}}$. This automatically leads to a very large ciphertext modulus q_{RLWE} given the condition $p_{\text{RLWE}} \ll q_{\text{RLWE}}$. Furthermore, to maintain a particular security level RLWE dimension needs to be increased accordingly which can have detrimental effects on overall efficiency. Nonetheless, in a different line of work [GHS12a, HS15] such large parameters were proven useful to evaluate the decryption formula homomorphically.

To avoid such "parameter bloating", we represent an integer (in mod q) using a low norm encoding technique derived from the work of [DM15].

4.6.1 Message Encoding

We can observe that the cyclotomic polynomial $R_q = \mathbb{Z}_q[X]/\left(X^n + 1\right)$ acts as a finite field where a small integer i can be encoded simply as a monomial $X^i \in R_q$. This encoding technique works for any two integers $a, b < n/2$ and addition of a and b simply maps to multiplication of cyclotomic polynomials as $(a + b) \rightarrow X^a \cdot X^b = X^{a+b}$. In group theory this is also referred to as embedding an additive group $\mathbb{Z}_q$ into a multiplicative group $\mathcal{G}$ of some order $|\mathcal{G}|$. Because of the negacyclic nature of cylotomic polynomials this multiplicative group changes sign when $(a + b) \geq n$. To account for this negative cycle, the first n integers are mapped to X^i while the next n integers are mapped to $-X^i$. In summary, a cyclotomic polynomial R_q acts a finite field of order $2n$ and shown as follows:

$$\mathbb{Z}_{2n} \rightarrow \mathcal{G}\langle X\rangle = \left\{1, X, \cdots, X^{n-1}, -1, -X, \cdots, -X^{n-1}\right\}$$

It is not much difficult to prove that $\mathcal{G}\langle X\rangle$ obeys the group properties as it has an identity element, $e = 1$, every element has an inverse, $X^a \rightarrow X^{-a} \in R_q$ and lastly a group action defined as $X|X^{2n} = e = 1$. Further, the group is also Abelian because of the commutative nature of multiplication.

4.6.2 Homomorphic Decryption with Ring-GSW FHE Scheme

In this section, we describe four different variants of our bootstrapping algorithm. The first two are applicable to *BV-LWE* ciphertext and the next two are applicable to *BV-GSW* ciphertext in LWE matrix form. We start with a brief overview of [DM15] and [CGGI16] bootstrapping procedure.

One of the key differences in bootstrapping procedure of both schemes lies in computation of the product term $a_i \cdot s_i \in \mathbb{Z}_q,\ (a_i, s_i \in \mathbb{Z}_q)$ required for updating the homomorphic accumulator. In [DM15], to compute the product term the authors resorted to bit decomposition of a_i followed by homomorphic multiplication with corresponding powers of 2 of s_i. Such an approach is chosen due to the fact that scalar multiplications in $\mathbb{Z}_q$ map to homomorphic exponentiation in cyclotomic polynomials. This results in a total cost of $\mathcal{O}(\log_2 q)$ homomorphic multiplications to produce a single product $a_i \cdot s_i$. It was shown that by moving to a larger base $b > 2$ the number of homomorphic multiplications can be reduced further at the cost of generating a larger bootstrapping key. Using this generalized base b the entire bootstrapping key, bk in closed form can be shown as follows:

$$bk = E\left(k \cdot s_i 2^j\right) \; \forall i \in [0, n), \; j \in [0, \lceil \log_b q \rceil), \; k \in [0, b)$$

Applying the [DM15] technique, homomorphic decryption of *BV-LWE* ciphertext along with the above mentioned bootstrapping key, bk can be expressed as follows:

$$\begin{aligned}\Delta_q &= E(b) + \sum_{i=0}^{n-1} (-a_i) \cdot E(s_i) \\ &= E(b) + \sum_{i=0}^{n-1} \sum_{j=0}^{\ell-1} bk(a_{i,j}, i, j) = E(m + pe)\end{aligned}$$

To compute the product $a_i \cdot s_i$ efficiently, the authors in [CGGI16] place a stronger security assumption on generation of secret keys from binary distributions ($\mathcal{B}$). Because of this assumption, the product simply reduces to $(0, a_i)$ for $s_i = (0, 1)$. In this case the product can be evaluated by picking appropriate values using a binary homomorphic multiplexer where the control bit is given by the secret key s_i. The product can be expressed using homomorphic computations as follows:

$$E(a_i \cdot s_i) = \mathbf{G} + (X^{a_i} - 1) \cdot E(s_i)$$

Bootstrapping key, bk here is simply a collection of encryptions of individual secret key terms, $s_i \in \{0, 1\}$. We can observe that the bootstrapping key generated here is smaller than that of [DM15] by a factor of $b\log_b(q)$. Applying the [CGGI16] technique, homomorphic decryption of *BV-LWE* ciphertext can be expressed as follows:

$$\Delta_q = E(b) + \sum_{i=0}^{n-1} \left(\mathbf{G} + \left(X^{-a_i} - 1\right) \cdot E(s_i)\right) = E(m + pe)$$

We can observe that for secret keys generated from ternary distributions ($\mathcal{T}$) or B_e-bounded discrete Gaussian distributions (χ_e) computation of each product term would scale linearly by a factor of $2\log_2(B_e)$ in number of homomorphic multiplexer operations. To circumvent such inefficiency we introduce a new technique for computing the product term by homomorphic exponentiation in cyclotomic rings or scalar multiplication in finite field space. More specifically, we rely on Frobenius automorphisms for achieving such scalar multiplications.

Particularly for a power of 2 cyclotomic polynomial with unpacked coefficients, Automorphism transformation with an index k maps a polynomial $z(X) \in R_Q$ to a RLWE polynomial $z\left(X^k\right) \pmod{\Phi_{m=2N=q}(X)} \in R_Q$. These Automorphism maps, denoted by $\tau_k : R_Q \mapsto R_Q$, are however only defined for odd indices k which are relative primes of $2N\,(=q)$. It is now known that the complete set of transformation maps form a group under composition and isomorphic to $(\mathbb{Z}/2N\mathbb{Z})^*$ [LPR10, GHS12c]. With regards to our monomial encoding for an integer $i \in \mathbb{Z}_q$ or $X^i \in R_Q$ Automorphism map τ_k produces another mononomial $X^{i\cdot k} \in R_Q$ which is equivalent to scalar multiplication in finite field $\mathbb{Z}_q$. To apply Automorphism on a ciphertext, c we apply the transformation maps on individual elements of the ciphertext after which the cphertext $\tau_k(c)$ can be decrypted by the transformed secret key $\tau_k(sk)$. To get back the ciphertext in native form, we *key-switch* the ciphertext *w.r.t* a pre-computed evaluation key.

Extending scalar multiplication to $\mathbb{Z}/2N\mathbb{Z}$ and trade-off in bootstrapping key size: Since, the scalar multiplications are not defined for even valued indices we apply the following simple tweaks to cover the scalar multiplication over the entire range of $\mathbb{Z}/2N\mathbb{Z}$.

- For $a_i = 2\kappa, a_i > 2$ we apply Automorphism $(\tau_k(s_i))$ with index $k = a_i - 1$ followed by a multiplication with null (noise less) ciphertext encrypting X.
- For $a_i = 2\kappa,\ a_i > 1$ we apply Automorphism $(\tau_k(2^j s_i))$ with index $k = a_i/2^j$ where j is the rightmost bit set in a_i.

The first approach is nearly as efficient as the second one however, memory overhead of bootstrapping key is very different. In the first approach bootstrapping key consists of encryptions of individual components of secret key whereas in the second one we generate all powers of 2 of secret key components. Additionally, in both approaches we also populate the bootstrapping key with evaluation keys $ks_i \leftarrow \textbf{KeySwitchGen}(\tau_i(sk), sk)\ \forall i \in (\mathbb{Z}/2N\mathbb{Z})^*$ to transform the intermediate ciphertexts to native form. We remark that in our implementation we used the second approach for the sake of efficiency. Applying this technique, homomorphic decryption of *BV-LWE* ciphertext can be expressed as follows:

$$\begin{aligned}\Delta_q &= E(b) + \sum_{i=0}^{n-1} \textbf{KeySwitch}\left[\tau_j\left(E(s_{i,j}), ks_i\right)\right] \\ &= E(m+pe),\ \text{where } j = \log_2(a_i \wedge -a_i)\end{aligned}$$

Testing polynomial and Post processing: At this stage, if homomorphic decryption is correct then Δ_q decrypts to $X^{m+pe} \in R_p$. To extract a *BV-LWE* or *BV-GSW* ciphertext deterministically from the constant term of RLWE ciphertext we have to sum up the coefficients in Δ_q ciphertext. This is done by multiplying Δ_q

with a special polynomial called *testing* polynomial, t and shown as follows:

$$t_0 = 1 + (Q-1)X + \cdots + (Q-1)X^{N-1} \in R_Q$$
$$t_1 = 1 + X + \cdots + X^{N-1} \in R_Q,$$
$$\gamma \equiv 0 \bmod p,\ |\gamma \geq q/4,\ t = \gamma t_0 t_1,\ \Delta'_q = \Delta_q t$$

As noted in [CGGI16], we apply the testing polynomial in the beginning to avoid an additional factor of $\sqrt{N}$ in final noise term. The factor of γ is introduced to pin down the noise in the range $[0, q/2)$ resulting in a positive coefficient polynomial. Consequently noise limit in *BV-LWE* scheme is reduced to $|q/4|$. Now, Δ'_q can be shown to be a ciphertext encrypting the plaintext message $N - 2\,(m + pe)$ in constant term. With additional linear transformations, the message is finally transformed to $m + pe$. At this stage, ciphertext Δ'_q is in a more desirable form, however, it can't be decrypted with original secret key. Assuming LWE dimension is less than RLWE dimension, $n \leq N$, we generate the key switching evaluation key $ks_{R_Q \to \mathbb{Z}^n}$ and *key-switch* the ciphertext.

$$\text{RLWE secret key } sk \in R_Q,\ \text{BV-LWE secret key } z \in \mathbb{Z}_q^n$$
$$z^* = \{z_0, -z_{n-1}, \cdots, -z_1\} \in \mathbb{Z}_Q^n$$
$$ks_{R_Q \to \mathbb{Z}^n} \leftarrow \textbf{KeySwitchGen}\,(sk, z^*)$$

Finally, we run a *mod-switch* procedure to reduce the ciphertext modulus Q in $\Delta_q \in R_Q^2$ to a very small value q and get back a ciphertext $w = (w_0, w_1) \in R_Q^2$. Bootstrapped *BV-SLWE* ciphertext is given by $c_{\text{refresh}} = (w_0[0], w_1[0], w_1[1], \cdots, w_1[n-1]) \in \mathbb{Z}_q^n$.

BV-LWE [$\mathcal{B}$] Bootstrapping Using the above described techniques, we proceed to describe our first bootstrapping Algorithm 1 applicable to *BV-LWE* scheme in

LWE form with binary secret keys. For this specific case, we retain the usage of homomorphic multiplexer for computing the product terms $a_i s_i$. Further, we keep the accumulator in BV [BV11b] ciphertext form.

Algorithm 1: *BV-LWE* $[\mathbf{z} \leftarrow_{\$} \mathcal{B}]$ Bootstrapping procedure

```
Input  : Ciphertext $c = (\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$, $bk_i$ s.t.
         $bk_i \leftarrow \mathbf{RGSW.Enc}(z_i, pk),\ i \in n.$
Output: Refreshed $c_{\text{ref}} = (\mathbf{a}_{\text{ref}}, b_{\text{ref}}) \in \mathbb{Z}_q^{n+1}$.
1 ACC $\leftarrow X^b \cdot t,$ ACC $\in R_p^2$ ;
2 for $i = 0$ to $n - 1$ do
3     $\bar{c} \leftarrow \mathbf{G} + (X^{-a_i} - 1) \cdot bk_i$ ;
4     ACC $\leftarrow$ ACC $\times \bar{c}$;
5 end for
6 ACC $\leftarrow 2^{-1}$negate(ACC $- N$);
7 $w_Q = (w_0, w_1) \leftarrow \mathbf{KeySwitch}(\text{ACC}, ks_{sk \to \mathbf{z}^*})$ ;
8 $w_q = \mathbf{ModReduce}_{Q \to q}(w_Q)$ ;
9 return $c_{ref} = \textbf{\textit{LWE-Extract}}(w_q)$
```

BV-LWE [$\mathcal{T}$ or χ_e] Bootstrapping Our next bootstrapping Algorithm 2 is applicable to *BV-LWE* scheme in LWE form with secret keys generated from ternary, $\mathcal{T}$ or discrete Gaussian distributions χ_e. Again, we keep the intermediate accumulator as BV scheme ciphertext however, at the greater cost of 2 key-switching operations per iteration.

BV-GSW Bootstrapping Bootstrapping *BV-GSW* ciphertext in LWE matrix form is a little more involved and needs more operations to refresh the ciphertext. From the construction of *BV-GSW* ciphertext, we can see that the top ℓ rows of the LWE matrix can be trivially obtained by applying LWE extraction procedure on the top ℓ rows of a *Ring-GSW* ciphertext. This even applies to the next ℓ rows but only for the cases when the message is a constant term, *i.e.*, remaining coefficients must be 0. Similarly, remaining $(n-1)\ell$ rows of LWE matrix can be formed by multiplication with $X \in R_q$ followed by LWE extraction. However, as shown in Algorithms 1 and 2 our accumulator, ACC stores random values in all coefficients except for the constant

Algorithm 2: *BV-LWE* [$\mathbf{z} \leftarrow_{\$} \mathcal{T}$ or χ_e] Bootstrapping procedure

```
Input  : Ciphertext $c = (\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$, $bk_{i,j}$ s.t.
         $bk_{i,j} \leftarrow \mathbf{RGSW.Enc}\left(2^j z_i, pk\right),\ i \in n, j \in \log_2(2N)$.
         $ks_{\rightarrow,i} = \mathbf{BV.KeySwitchGen}\left(sk, \tau_i\left(sk\right)\right)$ and
         $ks_{\leftarrow,i} = \mathbf{BV.KeySwitchGen}\left(\tau_i\left(sk\right), sk\right) \forall i \in \left(\mathbb{Z}/2N\mathbb{Z}\right)^*$
Output: Refreshed $c_{\text{ref}} = (\mathbf{a}_{\text{ref}}, b_{\text{ref}}) \in \mathbb{Z}_q^{n+1}$.
1  ACC $\leftarrow X^b \cdot t$, ACC $\in R_p^2$ ;
2  for $i = 0$ to $n - 1$ do
3      $a' \leftarrow q - a_i,\ j \leftarrow \log_2(a' \wedge -a')$;
4      $k \leftarrow (a' \gg j)$;
5      $\bar{c_k} \leftarrow \tau_k\left(bk_{i,j}\right)$;
6      ACC$_k \leftarrow$ KeySwitch$\left(\text{ACC}, ks_{\rightarrow,k}\right)$ ;
7      ACC$_k \leftarrow$ ACC$_k \times \bar{c_k}$;
8      ACC $\leftarrow$ KeySwitch$\left(\text{ACC}_k, ks_{\leftarrow,k}\right)$ ;
9  end for
10 ACC $\leftarrow 2^{-1}\text{negate}\left(\text{ACC} - N\right)$;
11 $w_Q = (w_0, w_1) \leftarrow$ KeySwitch$\left(\text{ACC}, ks_{sk \rightarrow \mathbf{z}^*}\right)$ ;
12 $w_q =$ ModReduce$_{Q \rightarrow q}\left(w_Q\right)$ ;
13 return $c_{ref} =$ LWE-Extract$\left(w_q\right)$
```

term. To resolve this issue, we rely on a RLWE coefficient extraction (RCE, discussed in Section 4.6.3) procedure which nullifies these random coefficients. As an aftermath of RCE procedure, the value of constant term increases by a factor equal to RLWE dimension. This factor can be canceled out by simply scaling the testing polynomial appropriately. We remark that the intermediate accumulator ACC can no longer be kept as a BV ciphertext because of the limitation in multiplication properties of BV FHE scheme. Further, accumulator (in *Ring-GSW* form) can only be multiplied by powers of 2 right after the RCE procedure to obtain the ℓ rows of *BV-GSW* ciphertext. Algorithm 3 and 4 outlines the bootstrapping procedure for different modes of secret key generation.

4.6.3 RLWE Coefficient Extraction Procedure

RLWE coefficient extraction (RCE) procedure discussed in this section allows us to selectively filter out the "garbage" values from the RLWE polynomial that is accrued during the bootstrapping procedure. Specifically, given a RLWE ciphertext

Algorithm 3: *BV-GSW* $[\mathbf{z} \leftarrow_{\$} \mathcal{B}]$ Bootstrapping procedure

```
Input  : Ciphertext $\mathbf{C} \in \mathbb{Z}_q^{O\times(n+1)}$, $b = \mathbf{C}_{0,0}$, $\mathbf{a} = (\mathbf{C}_{0,1} \cdots \mathbf{C}_{0,n+1})$, $bk_i$ s.t.
         $bk_i \leftarrow \mathbf{RGSW.Enc}\,(z_i, pk)\,,\ i \in n$.
Output: Refreshed $\mathbf{C}_{\text{ref}} \in \mathbb{Z}_q^{O\times(n+1)}$.
1  $\text{ACC} \leftarrow X^b \cdot t \cdot (N^{-1} \;\%\; Q)$ ;
2  for $i = 0$ to $n-1$ do
3      $\bar{c} \leftarrow \mathbf{G} + (X^{-a_i} - 1) \cdot bk_i$ ;
4      $\text{ACC} \leftarrow \text{ACC} \times \bar{c}$;
5  end for
6  $\text{ACC} \leftarrow \mathbf{RCE}\,(\text{ACC})$ ;
7  for $i = 0$ to $\ell - 1$ do
8      $\text{ACC}_i \leftarrow \text{ACC} \times 2^i,\ \text{ACC}_i \in R_Q^2$ ;
9      $\text{ACC}_i \leftarrow 2^{-1}\text{negate}\,(\text{ACC}_i - 2^i N)$ ;
10     $\mathbf{W}_{Q,i} \leftarrow \mathbf{KeySwitch}\,(\text{ACC}_i, ks_{sk\to\mathbf{z}^*})$ ;
11     $\mathbf{W}_{q,i} = \mathbf{ModReduce}_{Q\to q}\,(\mathbf{W}_{Q,i})$ ;
12 end for
13 return $\mathbf{C}_{ref} = \boldsymbol{GSW\text{-}Extract}(\mathbf{W}_{q,0}, \cdots, \mathbf{W}_{q,\ell-1})$
```

Algorithm 4: *BV-GSW* $[\mathbf{z} \leftarrow_{\$} \mathcal{T}$ or $\chi_e]$ Bootstrapping procedure

```
Input  : Ciphertext $\mathbf{C} \in \mathbb{Z}_q^{O\times(n+1)}$, $b = \mathbf{C}_{0,0}$, $\mathbf{a} = (\mathbf{C}_{0,1} \cdots \mathbf{C}_{0,n+1})$, $bk_{i,j}$
         s.t. $bk_{i,j} \leftarrow \mathbf{RGSW.Enc}\,(2^j z_i, pk)\,,\ i \in n, j \in \log_2(2N)$.
         $ks_{\leftarrow,i} = \mathbf{RGSW.KeySwitchGen}\,(\tau_i\,(sk)\,, sk)\,\forall i \in (\mathbb{Z}/2N\mathbb{Z})^*$
Output: Refreshed $\mathbf{C}_{\text{ref}} \in \mathbb{Z}_q^{O\times(n+1)}$.
1  $\text{ACC} \leftarrow X^b \cdot t \cdot (N^{-1} \;\%\; Q)$ ;
2  for $i = 0$ to $n-1$ do
3      $a' \leftarrow q - a_i,\ j \leftarrow \log_2(a' \wedge -a')$;
4      $k \leftarrow (a' \gg j),\ \bar{c_k} \leftarrow \tau_k\,(bk_{i,j})$;
5      $\bar{c} \leftarrow \mathbf{KeySwitch}\,(\bar{c_k}, ks_{\leftarrow,k})$ ;
6      $\text{ACC} \leftarrow \text{ACC} \times \bar{c}$;
7  end for
8  $\text{ACC} \leftarrow \mathbf{RCE}\,(\text{ACC})$ ;
9  for $i = 0$ to $\ell - 1$ do
10     $\text{ACC}_i \leftarrow \text{ACC} \times 2^i,\ \text{ACC}_i \in R_Q^2$ ;
11     $\text{ACC}_i \leftarrow 2^{-1}\text{negate}\,(\text{ACC}_i - 2^i N)$ ;
12     $\mathbf{W}_{Q,i} \leftarrow \mathbf{KeySwitch}\,(\text{ACC}_i, ks_{sk\to\mathbf{z}^*})$ ;
13     $\mathbf{W}_{q,i} = \mathbf{ModReduce}_{Q\to q}\,(\mathbf{W}_{Q,i})$ ;
14 end for
15 return $\mathbf{C}_{ref} = \boldsymbol{GSW\text{-}Extract}(\mathbf{W}_{q,0}, \cdots, \mathbf{W}_{q,\ell-1})$
```

$c = \textbf{Encrypt}\left(m\right)$ where $m = (m_0, g_1, \cdots, g_{n-1})$ the RCE procedure on c results in a ciphertext $c' = \textbf{Encrypt}\left(m'\right)$ where $m = (nm_0, 0, \cdots, 0)$ and n is the ring dimension.

Coefficient extraction procedure shown in [DM15] is very efficient but only results in a LWE ciphertext which cannot be converted to RLWE form. Alternatively, after extraction of coefficient in LWE form one can decrypt it using $\mathcal{O}\left(n\right)$ homomorphic operations, however, such method is accompanied with significant noise growth. We describe an efficient RCE procedure which requires $\mathcal{O}\left(\log_2\left(n\right)\right)$ homomorphic operations with very little noise growth.

Our RCE procedure mainly comprises of Automorphism operations on RLWE polynomials with unpacked coefficients. Automorphism $(\tau_k : R_q \mapsto R_q)$ with an index $k \in \left(\mathbb{Z}/2n\mathbb{Z}\right)^*$ produces a transformation on polynomial m as follows:

$$\tau_k\left(m(X)\right) \mapsto m\left(X^k\right) = \bar{m}\left(x\right)$$

The effect of Automorphism is that the j-th coefficent of $\bar{m}$ is now related to the i-th coefficient of the original polynomial by the equation $(2j-1) = (2i-1)\,k \bmod 2n$ for any $i, j \in [n]$. It was observed in[GHS12c, GHS12b] that coefficients embedded in slots of $\bar{m}$ shift cyclically when the cyclotomic (not a power of 2) polynomial is in packed form. However, in the case of power of 2 cyclotomic polynomials all the coefficients except for the constant term permute in a particular order depending upon the value of k. Another interesting property of Automorphism is that we can selectively negate coefficients of $\bar{m}$ by choosing appropriate k values. Using the idempotentcy and negation property of Automorphism, we can filter out half of the original coefficients of m. However, after each Automorphism we need to *key-switch* the transformed ciphertext to bring it back to original form so that it can be added to the original ciphertext c. Repeating this process $\log_2\left(n\right)$ times gives us the required ciphertext. RCE procedure is outlined in Algorithm 5.

Algorithm 5: RLWE Coefficient Extraction (RCE) Procedure

```
Input  : A ciphertext $c = \textbf{Encrypt}(m), m \in R_q$ of dimension $n$ and
         cyclotomic order $2n$. $m = (m_0, g_1, \cdots, g_{n-1})$. Evaluation keys,
         $ks_i$ such that $ks_i \leftarrow \textbf{KeySwitchGen}(\tau_i(sk), sk)\ \forall i \in (\mathbb{Z}/2n\mathbb{Z})^*$.
Output: Ciphertext $c' = \textbf{Encrypt}(m'), m' \in R_q$. $m' = (nm_0, 0, \cdots, 0)$.
1  $b \leftarrow \log_2(n)$ ;
2  $c' \leftarrow c$ ;
3  for $i = 0$ to $b - 1$ do
4      $p := 2^i$;
5      $\kappa := n/p + 1$;
6      $\bar{c}_{\tau_\kappa(sk)} \leftarrow \tau_\kappa(c')$ ;
7      $\bar{c}_{sk} \leftarrow \textbf{KeySwitch}(\bar{c}_{\tau_\kappa(sk)}, ks_\kappa)$ ;
8      $c' \leftarrow c' + \bar{c}_{sk}$ ;
9  end for
10 return $c'$
```

4.7 Gridstrapping

In this section, we describe the work on bootstrapping that builds on the above work and other common FHE tools and techniques. Our starting point is the additive group structure $\mathbb{Z}_q$ mentioned in the work by Sheriff and Pikert [ASP14] on bootstrapping. Recall that the additive group is defined as an embedding of $\mathbb{Z}_q$ into the symmetric group S_q of permutation matrices. Such an embedding allows us to express an integer $x \in \mathbb{Z}_q$ as a q-by-q permutation matrix. Alternatively, the permutation matrix can be represented succinctly by an indicator vector $\{0,1\}^q$ where the position of 1 denotes the integer value. Given such a multiplicative group represented by permutation matrices, each addition translates to $\mathcal{O}(q^2)$ homomorphic operations.

We forgo many other details and focus on this multiplicative group S_q. In order to extend this field over a large modulus q, the authors [ASP14] split the modulus into many small terms by a direct application of Chinese Remainder Transform. Since arithmetic on $\mathbb{Z}_q$ is isomorphic to $q = \prod_i \mathbb{Z}_{r_i}$ an integer is now expressed in terms of a group of indicator vectors $\{0,1\}^{r_i}$. As a natural consequence the number of homomorphic operation reduces to $\mathcal{O}(r_i^2)$ for each integer addition. However, it is not clear how to compute an inverse CRT on these ensemble of homomorphically encrypted indicator vectors.

4.7.1 Message Encoding

Building further on the above mentioned idea of indicator vectors, we express an integer $x \in \mathbb{Z}_{q^\zeta}$ as a tuple of indicator vectors $\{0,1\}^q$ where ζ is the grid dimension parameter. Collectively, integer x can be shown as an indicator vector $\{0,1\}^{q^\zeta}$ where the position of 1 denotes its value. For the simple case of $\zeta = 2$, integer x is represented by a sparse grid or matrix. In this case, position of 1 is indicated by a row pointer and a column pointer. Figure 4.1 shows a multi dimensional grid where the position of 1 is shown by k_0, k_1 and k_2 pointers. Integer x can be shown to be equal to $x = k_2 \cdot q^2 + k_1 \cdot q + k_0$. Generalizing this over a multi dimensional grid with a grid dimension parameter ζ an integer in the grid is represented as follows:

$$x = k_{\zeta-1} \cdot q^{\zeta-1} + k_{\zeta-2} \cdot q^{\zeta-2} + \cdots + k_0 \in \mathbb{Z}_{q^\zeta}$$

The above equation is reminiscent of representation of a large number in base-q and indeed this idea converges with our goal to bootstrap ciphertexts with large modulus. We represent each of the indicator vector $k_i = \{0,1\}^q$ as cyclotomic polynomial with RLWE dimension $n = q$. Now, it only remains to show the addition of integers using grid representation.

Grid Addition: Addition of grid integers quickly lands us in problems because of the negacyclic nature of cyclotomic polynomials. Concretely, for two integers i_0 and i_1 addition of grid integers translates to homomorphic multiplication of ciphertexts, $E(i_0)$ and $E(i_1)$. If the sum is greater than n, then the resultant polynomial attains a negative sign and falls out of encoding domain. To resolve this issue, we need to evaluate absolute value of the resultant polynomial. For this purpose, we extract the sign ($\text{sgn} \in \{-1,+1\}$) after every ciphertext multiplication and again multiply the

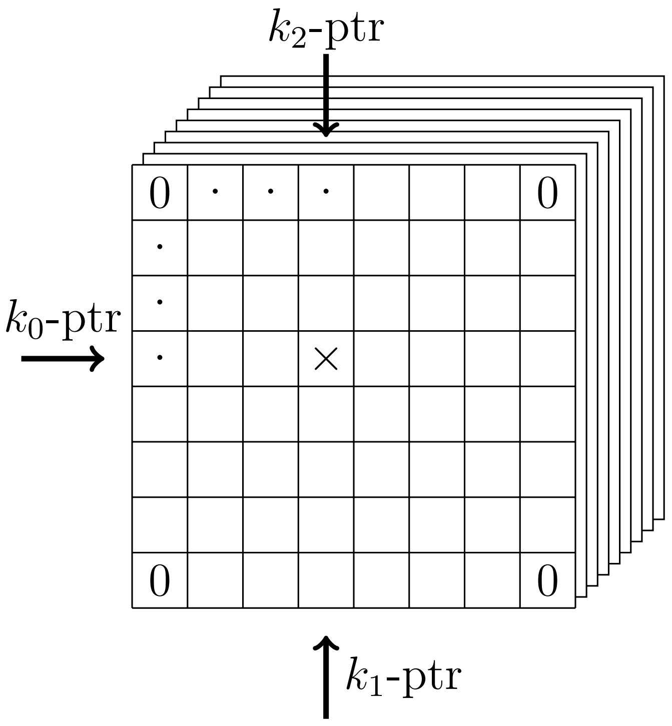

Figure 4.1 Figure shows a multi dimensional grid representation of an integer, $\zeta = 3$, $q = n^{\zeta}$. k_0-ptr, k_1-ptr and k_2-ptr are ciphertexts capturing different grid dimensions.

ciphertext with the sign. Pseudocode for grid addition is outlined in Algorithm 6. For sign extraction, we apply the RCE procedure on the product term, R_i's after multiplying it with the testing polynomial t_0, however, this step can be omitted if the message polynomial is already present in a rotated form (by a multiplication with t_1).

Carryover Next, we have to generate and propagate a carry-over for each of the indicator ciphertexts. We define a homomorphic carry-over generation circuit, **EvalCarry** as a map from sign, sgn to $\{1, X\}$, defined in the following equation:

$$\textbf{EvalCarry} : \text{sgn} \mapsto \{1, X\}$$

$$\text{carry} = \{(1 - X) \times sgn + (1 + X)\} \times 2^{-1}\mathbf{G}$$

Bootsrapping keys: We assume *BV-LWE* secret keys are generated from binary ($\mathcal{B}$) distributions but it can be extended to other distributions at the expense of

Algorithm 6: Grid Add procedure

```
Input  : Grid ciphertexts, $P = (P_{\zeta-1}, \cdots, P_0)$ and $Q = (Q_{\zeta-1}, \cdots, Q_0)$,
         grid dimension, $\zeta$.
Output: Grid ciphertexts, $R = (R_{\zeta-1}, \cdots, R_0)$.
1 carry $\leftarrow X^0$ ;
2 for $i = 0$ to $\zeta - 1$ do
3     $R_i \leftarrow P_i \times Q_i \times$ carry;
4     sgn $\leftarrow$ RCE $(R_i \times t_0)$;
5     carry $\leftarrow$ EvalCarry $(sgn)$ ;
6     $R_i \leftarrow R_i \times$ sgn ;
7 end for
8 return $R = (R_{\zeta-1}, \cdots, R_0)$.
```

more number of evaluations. Bootstrapping keys are simply published as encryption of individual secret key components so that the product term $(a_i s_i)$ remains as a positive coefficient indicator vector. We use homomorphic multiplexer function for generating grid product terms. Gridstrapping procedure is described in Algorithm 7.

4.8 Experimental Results

We evaluated our bootstrapping Algorithms 1-3 on open source C++ lattice crypto library PALISADE [1]. Our testing platform consist of a 64-bit quad core (i7-7700HQ) processor clocked at 2.80GHz. Further, we remark that our implementations were run entirely on a single thread with no hardware optimizations.

Unlike previous works [DM15, CGGI16, CGGI17] which worked on complex FFTs for switching between RLWE polynomial representation, our work relies on finite field FFT or NTT with mod-q reductions done with optimized Barrett reductions. Further, we believe that our runtimes will improve by a factor of 4 or more by using AVX vectorized instructions in FFT implementation as demonstrated by these previous works. We treat hardware optimization as an independent topic and reserve it for future works. Since runtime of bootstrapping algorithms are mostly dominated by the relinearization factor, r and NTT operations we kept most of the

[1]. PALISADE homomorphic encryption software library is currently in version 1.9.1 and available to download from `https://palisadecrypto.org`

Algorithm 7: *BV-LWE* [$\mathcal{B}$] Gridstrapping procedure

```
Input  : Ciphertext $c = (\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$, $bk_i$ s.t. $q = N^{\zeta}$,
         $bk_i \leftarrow \mathbf{RGSW.Enc}(z_i, pk)$, $i \in [0, n-1]$.
Output: Refreshed $c_{\text{ref}} = (\mathbf{a}_{\text{ref}}, b_{\text{ref}}) \in \mathbb{Z}_q^{n+1}$.
1  $\text{ACC}_{grid} = (\text{ACC}_{\zeta-1}, \cdots, \text{ACC}_0) \leftarrow \mathbf{GridEncode}(b)$ ;
2  $\text{ACC}_{grid} \leftarrow \text{ACC}_{grid} \cdot N^{-1} \cdot t_1$ ;
3  for $i = 0$ to $n-1$ do
4      $a_i \leftarrow N^{\zeta} - a_i$ ;
5      $a_{i,j} \leftarrow \mathbf{GridEncode}(a_i)$, $j \in [0, \zeta-1]$ ;
6      for $j = 0$ to $\zeta - 1$ do
7          $p_j \leftarrow \mathbf{G} + (X^{a_{i,j}} - 1) \cdot E(s_i)$ ;
8      end for
9      $P \leftarrow (p_{\zeta-1}, \cdots, p_0)$ ;
10     $\text{ACC}_{grid} \leftarrow \mathbf{GridAdd}(P, \text{ACC}_{grid})$ ;
11 end for
12 $\text{ACC} \leftarrow 0$ ;
13 for $i = 0$ to $\zeta - 1$ do
14     $\text{ACC}_i \leftarrow \text{ACC}_i \times t_0$ ;
15     $\text{ACC}_i \leftarrow \text{RCE}(\text{ACC}_i)$;
16     $\text{ACC}_i \leftarrow 2^{-1}\text{negate}(\text{ACC}_i - N)$ ;
17     $\text{ACC} \leftarrow \text{ACC} + \text{ACC}_i \times (N^i \bmod p)$;
18 end for
19 $w_Q \leftarrow \mathbf{KeySwitch}(\text{ACC}, ks_{sk \to \mathbf{z}^*})$ ;
20 $w_q = \mathbf{ModReduce}_{Q \to q}(w_Q)$ ;
21 return $c_{ref} = \textbf{\textit{LWE-Extract}}(w_q)$
```

RLWE elements in evaluation form and r considerably large. For the same reason, we performed Automorphism operations in evaluation representation making the generalized secret key bootstrapping very efficient.

4.8.1 Parameter Selection and Results

In our experiments, we used a static parameter set from [DM15] to fix the parameters for *BV-LWE* FHE and *BV-GSW* FHE schemes. Parameters for RLWE cryptosystems are usually set from security and correctness constraints. However, in our experiments we chose standard RLWE dimensions of $N = 512,\ 1024$ and then determine correctness empirically. While security level (bits of security) for some of the parameters have been set to a relatively low value, we remark that these runtimes help in ease of comparison with other bootstrapping techniques and can be easily extended for more secure parameter settings.

BV-LWE/GSW Parameters: LWE dimension, $n = 500$, ciphertext modulus, $q = 512$, plaintext modulus, $p = 5$.

From the results, we observe that we get the fastest bootstrapping runtime from Algorithm 1 where the secret keys are generated from binary distribution. Algorithm 2 has a runtime roughly twice that of Algorithm 1 but has the added benefit that it can work with arbitrary distribution secret keys. Lastly, Algorithm 3 has a relatively higher runtime because of the fact that the ciphertext is in LWE matrix form. Moreover, the runtimes of these algorithms grow proportionately as the RLWE dimension is increased.

4.9 Conclusion and Future Directions

In this chapter, we presented an improved variant of *BV-LWE* scheme and discussed several novel techniques for refreshing a ciphertext of the scheme. Building on prior works we have shown that a LWE homomorphic scheme capable of computing on

Table 4.1 Experimental Results Showing Bootstrapping Runtimes for *BV-LWE* Scheme using Different Algorithms and Parameters

Algorithm	N	r	Q bits	Bootstrapping KeyGen (ms)	Bootstrapping Runtime (ms)
1	512	11	32	308	133
		8	31	410	166
2		11	32	2820	233
		8	31	3680	295
3		14	41	310	721
		11	43	412	1187
1	1024	13	37	675	282
		11	43	852	362
2		13	37	6250	495
		11	43	8310	615
3		11	43	832	2477

RLWE plaintext modulus $p = 5$, $\sigma = 4$.

integers can be bootstrapped efficiently resulting in a fully homomorphic scheme. In contrast to prior work. which convert arithmetic functions to binary circuits, our LWE FHE scheme can be applied directly resulting in several orders of improvement in performance.

In the future, we plan to improve our bootstrapping algorithm runtimes by porting over to heterogeneous execution platforms such as GPUs and FPGAs. We also plan to bring in other FHE tools such as RLWE Field switching which can decouple the LWE and RLWE parameters more effectively. Lastly, we believe our techniques will benefit the most if they can be extended to bootstrap RLWE FHE schemes.

CHAPTER 5

ACCELERATING LATTICE BASED PRIMITIVES ON GPUS

5.1 Introduction

Over the last decade, lattice based cryptography has revolutionized the field of cryptology by introducing many powerful cryptographic primitives such as Fully Homomorphic Encryption (FHE) schemes, Proxy Re-Encryption (PRE) schemes, digital signature, functional encryption and many more. In theory, many of these lattice based primitives are considered to be more efficient than other traditional cryptosystems which rely on modular exponentiation. This is mainly because evaluating modular exponentiation of very large integers (depending on security factor) is a time consuming operation and further the most optimized algorithms for exponentiation are not amenable to parallelization. On the other hand, it is also evident from numerous implementation of lattice-based cryptographic schemes on modern computing platforms that they still suffer from inefficiencies incurred due to intensive computational workloads. Some of the inefficiencies can still be mitigated by careful design choices and other trade-off in parameters however such choices often come with a limitation on improvement in performance. A large number of these implementations can be efficiently realized in practice by mapping the schemes to appropriate hardware platform or synthesizing, part or all of a scheme, on special purpose hardware. Additionally, performance of the system is boosted tremendously if the computational workload can be split up into parallel constructs and mapped to individual hardware units.

Fortunately, most of the lattice based primitives belong to the category of problems which are in *NC*. Particularly, the class *NC* captures the set of problems for which there are efficient parallel algorithms or in other words problems that are *massively parallelizable.* Therefore, in theory many lattice based schemes can be efficiently implemented in parallel using a parallel computer with $n^{\mathcal{O}(1)}$ processors and in time $\log^{\mathcal{O}(1)}(n)$ where n is the size of input and assuming availability of infinite number of processors. For example, many operations on matrices such as computation of product, rank and determinant have been shown to be computable in parallel as the depth of circuits corresponds to a parallel time. Furthermore, the trend in semiconductor industries is continuously shifting focus on high performance multi-core architectures instead of building power hungry single-core processors running at significantly higher clock speeds. This strongly suggests and justifies the need to investigate the implementation of lattice based primitives on high performance computational platforms such as ASICS, FPGAs, GPUs etc. In this chapter, we describe the acceleration of RLWE based Proxy Re-Encryption schemes and bootstrapping procedure by implementing them on heterogeneous computing platforms such as GPUs.

5.1.1 Motivation for Accelerating PRE Schemes on GPUs

First introduced in the work of Blaze, Bleumer and Strauss [BBS98], Proxy Re-Encryption (PRE) is a powerful cryptographic primitive that allows a subscriber (Bob) to exchange and interpret encrypted data received from a publisher (Alice) without ever exchanging any secret key. To interpret the messages, Alice creates and gives to Proxy (Polly) a re-encryption key which then allows Polly to transform messages encrypted with Alice's public key into an encrypted message that can be decrypted by Bob's secret key. Furthermore, semantic security of proxy re-encryption guarantees that the proxy, Polly doesn't learn anything about Alice's secret key or messages.

Proxy re-encryption can be immensely useful in brokering information exchanges in untrusted environments such as cloud computing platforms. Users can choose to store contents in encrypted form on cloud and then register re-encryption keys of other users they want to interact with. Now, the cloud acting as a proxy, can re-encrypt messages on the fly and deliver encrypted messages that can be read by the desired users. Even if the cloud is corrupted by a malicious adversary and all data stored on cloud is compromised, the adversary cannot retrieve meaningful information out of it. Further, the adversary in possession of re-encryption keys cannot deduce secret keys of either the producer or consumer. Additionally, if the PRE scheme in deployment is *key private* secure then the adversary cannot even trace the identities of Alice and Bob.

Building PRE primitive with FHE schemes presents a simple yet powerful approach for extending them for information exchanges in untrusted environment. Further these PRE schemes mitigate some of the problems associated with traditional PRE schemes as they are inherently endowed with *uni-directional* and *multi-hop* nature. Construction of a PRE scheme based on BV [BV11b] FHE scheme was presented in [PRSV17] where the authors exploited the key-switching procedure for achieving the re-encryption functionality. Similarly, in [SR20] the authors presented two PRE schemes based on GSW [GSW13] FHE scheme and it's RLWE variant Ring-GSW [KGV16] FHE scheme. Evaluation of the both BV-PRE and Ring-GSW-PRE (with $r = 1$) with standard parameter set and security factor (100-bits) shows that re-encryption procedure can be completed in the order of $10 - 100$ milliseconds.

Nowadays, owing to the ever increasing computational power and network efficiency most of the applications hosted on cloud are expected to work in real time. An application involving a PRE primitive is no different to this. Being a low level key primitive, the PRE scheme in deployment is expected to deliver least possible latency. The above mentioned PRE schemes share the similarity of using the algebraic structure of ideal lattices as polynomial rings (RLWE) or finite field vectors

(LWE). As a result, many of the computational bottlenecks in implementation of these PRE schemes arise due to large ring dimensions, arbitrary precision multiplications, polynomial tensor, number theoretic transforms, matrix multiplication etc. Over the years many of these problems have been remedied by switching over to better algorithms or optimizations. However, in order to provide additional speedups better hardware architectures have to be considered. In a heterogeneous computing model, these hardware accelerators acting as co-processors can be orchestrated by CPUs to achieve the desired levels of throughput. Among the common hardware accelerators such as FPGAs, ASICs and GPUs the most common and readily available solution is provided by GPUs. Modern GPUs consist of streaming multi-core processors which can be utilized to accelerate parts of computations that can be processed in parallel.

Lattice based cryptography and more specifically RLWE based FHE schemes being amenable to such parallelism have shown significant folds of improvement in performance when implemented on GPUs. For example, in [DDS14] the authors presented an implementation of NTRU FHE scheme on GPUs and evaluated AES and Prince block ciphers resulting in 2.5-7.6x factors of speedup over CPUs. Similarly, [DS15] present a homomorphic encryption accelerator library, cuHE targeting LTV [LATV12], BGV [BGV14] and DHS [DHS14] FHE schemes. Speedups of 12-41x were reported by the authors for homomorphic sorting of ciphertexts. In another work [KGV16] leveraged the power of GPUs towards construction of homomorphic Bayesian spam filter, secure multiple keyword search, and evaluation of binary decision trees based on Ring-GSW FHE scheme. For the same security settings [KGV16] reported a factor of 10x improvement in performance when compared with IBM HeLib [HS14] software library. Continuing in this line work, we present an implementation of PRE schemes based on BV and Ring-GSW FHE schemes.

5.1.2 Motivation for Accelerating Bootstrapping Procedures for LWE FHE Scheme on GPUs

Nearly a decade ago, Gentry proposed the first Fully Homomorphic Encryption (FHE) scheme, a powerful cryptographic primitive that enables computations on encrypted data. FHE is the most versatile tool that can be directly applied to outsource storage and computation to a remote server, enable private queries on a database or a search engine and other secure two-party computations. Because of its wide spectrum of application and strong security guarantees, fully homomorphic encryption is sometimes dubbed as the "holy grail of cryptography". Although for all practical purposes a fixed depth homomorphic encryption scheme or SHE can be used however, such instantiations are rather found to be inefficient due to the requirement of very large modulus to contain the noise and satisfy the correctness constraints. In the FHE approach, noise in the ciphertext is allowed to grow during computation phase and once it reaches a certain threshold the ciphertext is refreshed by applying a bootstrapping procedure. First described by Gentry [Gen09, $G^{+}09$], a repeated application of this bootstrapping procedure converts a fixed depth HE scheme to FHE in all currently known schemes.

In all practical implementations [HS15, GH11], bootstrapping procedure is often found to be computationally expensive because of homomorphic evaluation of the decryption circuit which in turn can bring down the efficiency of the entire FHE scheme or the application depending on it. To circumvent such inefficiency, it is much desirable to offload the entire bootstrapping procedure to a hardware accelerator such as GPU. In this chapter, we explore hardware acceleration of *BV-LWE* like FHE schemes using NVIDIA GPUs interfaced on devices ranging from high end HPCs to resource constrained embedded systems (NVIDIA Jetson AGX Xavier). A key advantage of using *BV-LWE* FHE scheme over *FHEW*-like [DM15] cryptosystem is that messages other than bits ($m > 2$) can be encrypted and operated on. Further, we remark that our GPU implementation can be easily extended for implementation of FHEW [DM15] or TFHE [CGGI16, CGGI17] FHE schemes.

Our Contributions: We enumerate our main contributions and scope of the chapter as follows.

- We present a GPU implementation of number theoretic transforms (NTT) based on finite field arithmetic where butterfly operations are performed in parallel. In order to extend the finite field over large numbers we keep integers in CRT representation. Our NTT implementation targets both small ($n \leq 1024$) and large ($n > 1024$) polynomial dimensions.
- Armed with a parallel implementation of NTT operation, next we target parallelization of bit decomposition procedure. Relinearization operation along with bit/digit decomposition is considered to be the most critical procedure in many homomorphic encryption schemes and accelerating this operation imparts overall efficiency to the FHE scheme.
- Utilizing the above implementations we demonstrate acceleration of BV-PRE and Ring-GSW PRE schemes. In our implementation we have reduced number of memory transfers between host and device to a minimum by storing most of the dynamic elements on GPU memory. Another key feature of our implementation is the use of cuda streams which allow concurrent execution of kernels thereby minimizing latency.
- Similarly, we present an implementation of bootstrapping procedure 1 for *BV-LWE* FHE scheme on NVIDIA GPUs. To minimize the latency of bootstrapping procedure we execute the optimized SHE operations of Ring-GSW FHE scheme on GPUs directly. Furthermore, to reduce the number of memory transfers we generate and store the bootstrapping keys on GPU memory.

Chapter Organization: Section 5.2 discusses the implementation of underlying arithmetic layer, number theoretic transforms and bit decomposition procedures on GPU along with other optimizations. We omit the description of PRE schemes and bootstrapping procedure 1 for *BV-LWE* FHE scheme as they have been discussed in previous chapters 3 and 4. In Section 5.3, we provide the parameters selected for implementation. Section 5.4 discusses the evaluation platforms, salient features of our software implementation and overall speedups achieved for these implementations.

5.2 Number Theoretic Transform and Bit-Decomposition

5.2.1 Number Theoretic Transform

Cryptosystems based on RLWE security assumption are defined over a polynomial ring $R = \mathbb{Z}[X]/\Phi_m(X)$ where $\Phi_m(X)$ is an irreducible monic polynomial, typically a cyclotomic polynomial of order m. This notation is extended to a polynomial R_q modulo an integer q where the coefficients of the polynomial are in the interval $(-q/2, q/2]$. Alternatively, an element $a \in R_q$ is simply considered to be a coefficient vector $\vec{\mathbf{a}} \in \mathbb{Z}_q^{\varphi(m)}$. While addition of these polynomials is quite efficient, multiplication leads to quadratic time complexity. To circumvent this inefficiency we represent polynomial rings in the so called "Evaluation" representation. For a polynomial $a \in R_q$, the coefficients can be converted to evaluation domain $\bar{a}$ by evaluating $a(X)$ at each of the m-th primitive roots of unity modulo q. Coefficients of $\bar{a}$ are related to polynomial a through the relation $\bar{a}_i = a(\omega^i) \mod q$ where $(i, m) = 1$ and ω is m-th primitive root of unity modulo q.

This back and forth conversion of a polynomial can be achieved efficiently by using number theoretic transforms (NTT) which is roughly similar to the classical n-dimensional fast fourier transform where finite field is used instead of complex numbers. Concretely, in our implementation we use power of two cyclotomics ($m = 2^k$) where $\Phi_m(X)$ is maximally sparse and ring dimension $n = \varphi(m) = m/2$ is also a power of two. Power of two cyclotomics along with NTT have become so pervasive in lattice based cryptography that overall efficiency of the cryptosystem depends upon latency of NTT procedure. For this reason, we chose to implement NTT as iterative Cooley-Tukey algorithm. More specifically, we implemented NTT routine with Fermat-Theoretic Transform (FTT) optimization which eliminates the need for interleaved zero padding when using the conventional NTT procedure. The pseudo-code describing the NTT decimation in time procedure targeting CPU platforms is shown in Algorithm 8.

5.2.2 Parallel NTT

Exploiting NVIDIA GPU architecture we reduce the NTT latency further by mapping the butterfly computations of each of the $\log n$ stages to an independently processing thread of a thread block. In NVIDIA CUDA architecture, each kernel can be potentially divided into a 3-dimensional array of blocks where a block further consists of a number of threads. CUDA runtime schedules these threads in groups of 32 threads (called warps) which execute concurrently on a streaming-multiprocessor (SM). Because of hardware restrictions, a maximum of 1024 threads can be assigned to a block. Further, these threads within a block have the capability to share data and more importantly synchronize with each other. In our implementation, for small polynomials ($n \leq 1024$) we map the coefficients entirely to a thread block and synchronize the thread block after completion of a stage as shown in Figure 5.1. We use shared memory for storing the intermittent results as latency associated with data retrieval for global memory is higher than that of shared memory which reside on chip. After completion of the entire NTT procedure, we transfer data back to the global memory. For larger polynomial rings ($n > 1024$) we use a combination of block level synchronization and stream level synchronization to avoid data race conditions. Because stream level synchronization avoids data race conditions via global memory synchronization, we pay the penalty of using slower memory but only for a fraction of the NTT procedure call.

Evaluation of the proposed NTT procedure on GPU platforms and CPU platforms is shown in Figure 5.2. From the figure, we can see that CPU platform running on single thread achieves slightly better performance for smaller ring dimensions. As the ring dimensions grow higher, we can see that the GPU platform start showing significant improvement in performance.

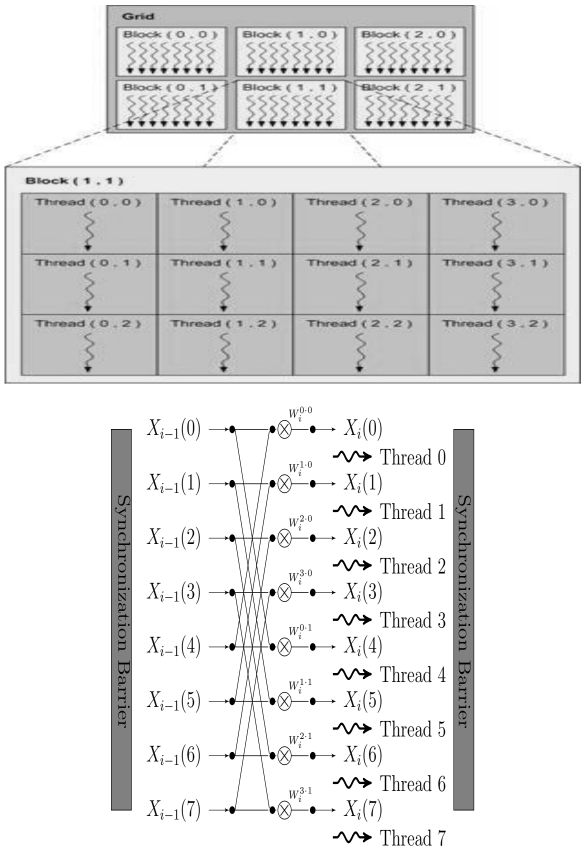

Figure 5.1 Figure demonstrating parallel implementation of the i-th stage of NTT on GPU, $N = 8$.

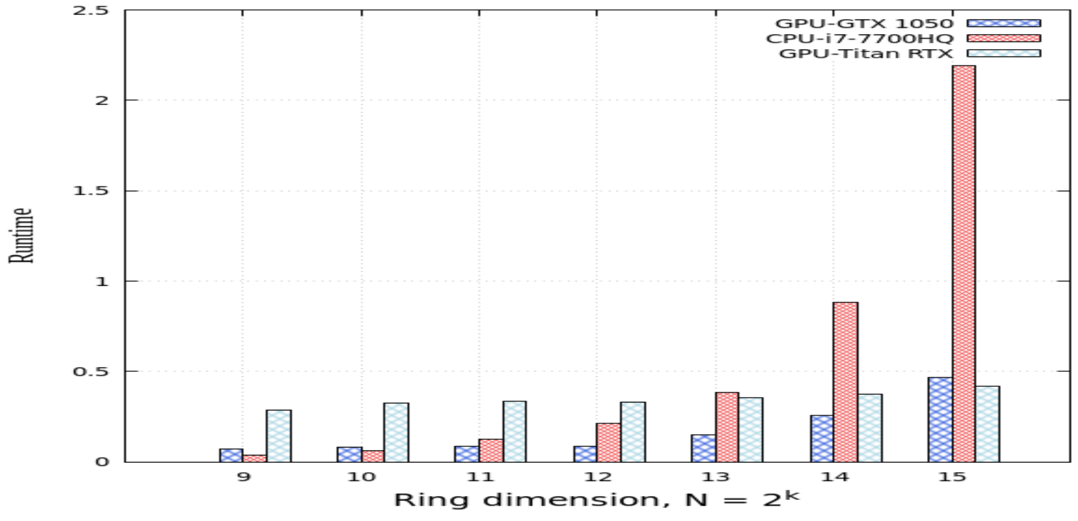

Figure 5.2 Comparison of CPU and GPU runtimes of NTT algorithm.

5.2.3 Barrett Modulo Reduction and Arbitrary Precision Support

For modulo reduction we used a variation of the generalized Barrett modulo reduction presented in [Bar86, DQ98], outlined in Algorithm 10. Barrett modulo reduction requires a pre-computation term $\mu = \lfloor 2^{2b}/q \rfloor$ for a particular modulus q and it's bit width, $b = \lceil \log_2 q \rceil$. We pre-compute these terms and transfer them to GPU global memory for read only access to any kernel. NVIDIA GPUs are restricted to 32-bit architecture however 64-bit arithmetic is supported by assembly code emulation. Because of this reason, we limit our modulus to 32-bits and prefer modulus with bit width closer to 32-bits so that the precomputation term, μ fits into word size. Concretely, we used $29 - 30$ bit width modulus in our implementation.

Lattice based cryptosystems, and particularly those related to homomorphic encryption employ the addition of low norm noise terms to base their security on Ring-LWE and LWE assumptions. As a result, noise term in ciphertexts grow upon homomorphic evaluation such as addition, multiplication and other operations. For preserving the correctness constraints so that ciphertexts are decrypted correctly, the modulus q should be chosen large enough such that the the final accumulated error terms do not "wrap around" modulo q. To extend support for larger modulus, we store a set of increasing prime moduli q_i by an application of Chinese Remainder Theorem (CRT). We only reconstruct back the coefficients into larger terms, for the purpose of decryption or bit-decomposition where the polynomial needs to be represented in terms of the larger modulus, $q = \prod_{i=0}^{t-1} q_i$.

Evaluation of the NTT procedure on GPU with varying number of moduli, t and ring dimension, n can be seen in Figure 5.3. It can be seen from the figure that for most of the ring dimensions the runtimes vary a little. This is due to the fact that GPUs have the capability to improve throughput by hiding latency with concurrent execution of NTT procedure on different polynomials. On a CPU platform

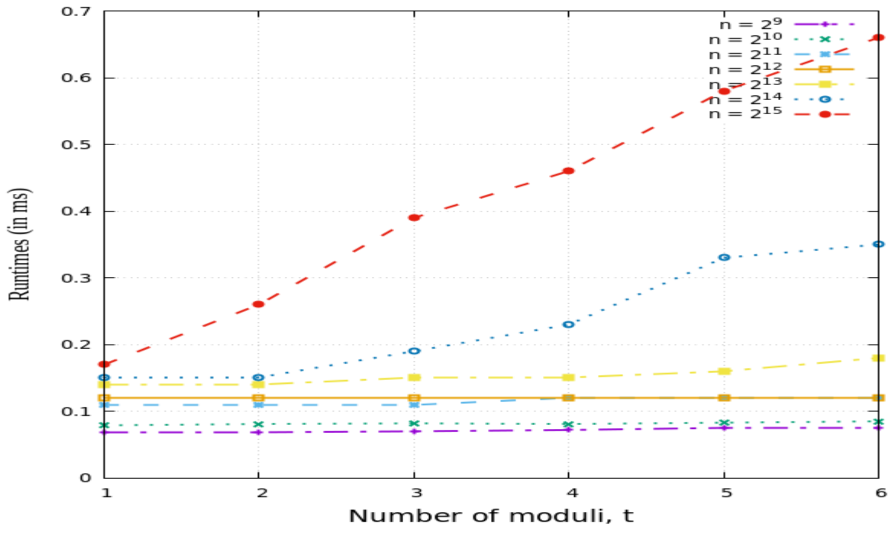

Figure 5.3 GPU runtimes of NTT algorithm with varying ring dimension and moduli, t.

the runtimes is estimated to scale linearly with the number of moduli assuming a single thread execution environment.

5.2.4 Bit Decomposition

Bit decomposition along with relinearization procedure forms the backbone of lattice based cryptography. Relinearization procedure has proved to be a crucial primitive for many cryptographic operations such as key-switching, re-encryption, homomorphic multiplication and ciphertext length reduction. While bit decomposing elements is a simple procedure in finite field arithmetic, it is accompanied with additional computational overhead in Ring-LWE based cryptosystems. In Ring-LWE based cryptosystems, ciphertexts and other key elements are mostly present in evaluation representation. In order to bit decompose polynomial ring elements, it needs to switched back to coefficient representation by an application of inverse NTT. At this stage, bit decomposition of the polynomial results in a vector of b polynomials, where b is the bit length of the modulus q. To perform further computations, these polynomials needs to be converted back to evaluation representation by a series of NTT calls.

Algorithm 8: NTT DIT Procedure

```
Input  : A polynomial $a \in R_q$ of dimension $n$ and cyclotomic order $m$.
         Modulus $q$, $m$-th primitive root of unity, $\omega$ mod $q$.
Output: Polynomial $\bar{a}$ of dimension $n$ in evaluation representation.
for $i \leftarrow 0$ to $n-1$ do
    $a_i \leftarrow a_i \cdot \omega^i \bmod q$;
end for
$\bar{a} \leftarrow \mathrm{BitReversal}(a)$;
for $i = 1$ to $\log n$ do
    $\ell \leftarrow 2^i$;
    for $j = 0$ to $n-1$ by $\ell$ do
        for $k = 0$ to $\ell/2 - 1$ do
            $p = (m/2^i) \cdot k$ ;
            $\zeta = \omega^p \% q$ ;
            $idxEven = j + k$ ;
            $idxOdd = idxEven + \ell/2$ ;
            $\bar{a}_{idxEven} = (\bar{a}_{idxEven} + \zeta \cdot \bar{a}_{idxOdd}) \% q$ ;
            $\bar{a}_{idxOdd} = (\bar{a}_{idxEven} - \zeta \cdot \bar{a}_{idxOdd}) \% q$;
        end for
    end for
end for
return $\bar{a}$
```

Algorithm 9: INTT DIT Procedure

```
Input  : A polynomial $\bar{a} \in R_q$ of dimension $n$ and cyclotomic order $m$.
         Modulus $q$, $m$-th primitive root of unity, $\omega$ mod $q$.
Output: Polynomial $a$ of dimension $n$ in coefficient representation.
$a \leftarrow \mathrm{BitReversal}(\bar{a})$;
$a \leftarrow \mathrm{NTT}(a,\ \omega^{-1},\ q)$;
for $i \leftarrow 0$ to $n-1$ do
    $a_i \leftarrow a_i \cdot \omega^{-i} \bmod q$;
end for
return $a$
```

Algorithm 10: Mod Barrett Reduction

```
Input  : $x \in [0, (q-1)^2]$, modulus $q$, bit-width $b = \lceil \log q \rceil$, $\bar{q} = 2 \cdot q$ and
         $\mu = \lfloor 2^{2b}/q \rfloor$.
Output: $z \leftarrow x\%\, q$
1  $z \leftarrow x \gg b$ ;
2  $z \leftarrow z \cdot \mu$ ;
3  $z \leftarrow x \gg b$ ;
4  $z \leftarrow z \cdot q$ ;
5  $z \leftarrow x - z$ ;
6  if  $z >= \bar{q}$ then
7      $z \leftarrow z - \bar{q}$;
8  end if
9  if  $z >= q$ then
10     $z \leftarrow z - q$;
11 end if
12 return $z$
```

Since the bit decomposed polynomials are independent of each other, we can apply NTT procedures on them in parallel. For this purpose, in our GPU implementation we map the three dimensional CUDA grid as follows:

- **grid**.X $\rightarrow$ mapped to individual bit of decomposed polynomial with grid dimension set to bit length b of modulus q.
- **grid**.Y $\rightarrow$ mapped to individual modulus q_i with grid dimension set to number of moduli, t.
- **grid**.Z $\rightarrow$ mapped to polynomial coefficients in excess of the maximum number of threads allowed to exist in a block.

In order to avoid race conditions in NTT procedure, we provide the kernel with appropriate synchronization. From Figure 5.4, we can observe that GPU implementation of bit decomposition outperforms runtimes of CPU platform for all ring dimensions and further the speedups are more pronounced in case of higher ring dimensions.

5.3 Parameter Selection

Estimating parameters for LWE or RLWE based encryption schemes is a significantly challenging task. On one hand, we have to ensure that the chosen parameters generate an underlying RLWE instance that is hard to solve as per known attacks while on the other hand we have to also meet the correctness constraint. Correctness constraint

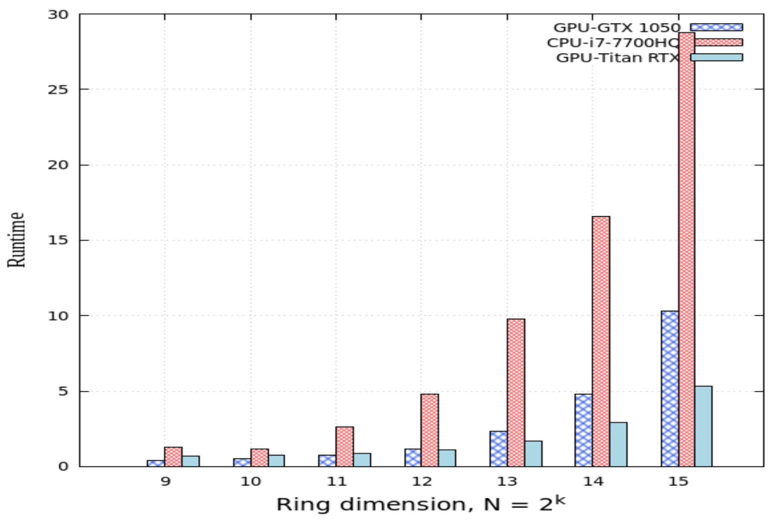

Figure 5.4 GPU vs CPU runtimes of bit decomposing polynomial ring with varying ring dimension, N.

can be trivially achieved by choosing an arbitrarily large modulus q. However, such a strategy is not suitable for efficient implementation and further leads to insecure LWE instances. It was shown in [LL15] when the modulus is exponential in LWE dimension, $q \geq 2^{\mathcal{O}(n)}$ and error distribution is narrow enough, secret key can be recovered in polynomial time using standard lattice basis reduction algorithms such as LLL [LLL82] and Babai's nearest planes method [Bab86]. These LWE attacking algorithms which work by reducing the lattice basis (LLL and BKZ), are often quantified by a parameter called root Hermite factor, δ which represents the quality of resultant basis. A smaller δ factor leads to better lattice basis and improved security. In order to lay out concrete parameter, Lindner and Peikert [LP11] gave a heuristic relation that computes runtime of BKZ lattice reduction algorithm for a particular root Hermite factor. This is shown as:

$$\log_2(t_{BKZ}) \geq \frac{1.8}{\log_2(\delta)} - 110$$

Further, Gentry *et al.* [GHS12c] gave a relation which computes minimum LWE dimension secure for a particular modulus q, standard deviation of error distribution

σ_e and root Hermite factor, δ. Combining the two we can express the minimum LWE dimension to support κ-bits of security as follows:

$$n \geq \frac{\log_2(q/\sigma_e)(\kappa + 110)}{7.2}$$

In particular, we used the runtime of BKZ2.0 [ACF+15] (considered to be a improved version of BKZ algorithm) given by the relation

$$\log_2(t_{BKZ2.0}) \geq \frac{0.009}{\log_2^2 \delta} - 27$$

Again, combining this with minimum LWE dimension relation from [GHS12c] we arrive at our final security constraint as follows:

$$n \geq \frac{\log_2(q/\sigma_e)\sqrt{\kappa + 27}}{0.379}$$

5.3.1 PRE Parameter Selection

In our implementation, we targeted for $\kappa = 128$-bits of security for both BV-PRE and Ring-GSW-PRE scheme. Using the correctness constraint from the respective schemes, we generated the working modulus for various ring dimensions as shown in Tables 5.1 and 5.2.

5.3.2 BV-LWE Bootstrapping Parameter Selection

In this case, we need to set parameters for both *BV-LWE* and *Ring-GSW* schemes separately. We fix the parameters for *BV-LWE* scheme as follows:

BV-LWE Parameters: LWE dimension, $n = 500$, ciphertext modulus, $q = 512$, plaintext modulus, $p = 5$.

Table 5.1 Minimum Modulus Bits that Satisfies 128-bits of Security for BV-PRE scheme

PRE Scheme	n	log(q) based on average case error bounds
BV-PRE	512	18
	1024	18
	2048	19
	4096	19
	8192	20
	16384	20

$B = 15$ and $\sigma = 4$.

Table 5.2 Minimum Modulus Bits that Satisfies 128-bits of Security for Ring-GSW-PRE Scheme

PRE Scheme	n	log(q) based on average case error bounds
Ring-GSW-PRE	1024	25
	2048	26
	4096	27
	8192	28
	16384	29

$B = 15$ and $\sigma = 4$.

For *Ring-GSW* FHE scheme, we set the ring dimension to $N = \{512, 1024\}$, use small relinearization factor r and increment them gradually. We set the modulus values (Q-bits) empirically by verifying the correctness of decryption.

5.4 GPU Implementation and Results

5.4.1 Software Implementation

We evaluated both BV-PRE and Ring-GSW-PRE scheme on two NVIDIA GPU devices, namely, GeForce GTX-1050 and Titan-RTX as shown in Table 5.3. While GPU1 is a commodity grade notebook GPU, GPU2 is a much more powerful GPU targeted towards compute intensive applications. Our software implementation [1] follows the modular structure of PALISADE [2] homomorphic encryption library

[1]PRE scheme implementations available to download from `https://git.njit.edu/grs22/pre-on-gpu`

[2]**PALISADE** homomorphic encryption software library is currently in version 1.9.1 and available to download from `https://palisade-crypto.org`

Table 5.3 Configuration and Features of GPU with their Corresponding CPU used for Evaluation of PRE Schemes

Feature	CPU1	CPU2	GPU1	GPU2
Model	Intel i7-7700HQ	Intel Silver 4114	NVIDIA GeForce GTX 1050	NVIDIA Titan RTX
Cores	4	10	640	4608
Clock Rate	2.8 GHz	2.2 GHz	1.49 GHz	1.77 GHz
Multiprocessors	-	-	5	72
RAM Memory	16 GBytes	181 GBytes	4042 MBytes	24190 MBytes
CUDA Capability	-	-	6.1	7.5

Table 5.4 Configuration and Features of GPU with their Corresponding CPU used for Evaluation of BV-LWE Bootstrapping Procedure

Feature	CPU1	CPU2	CPU3	GPU1	GPU2	GPU3
Model	Intel i7-7700HQ	Intel Silver 4114	ARM v8	NVIDIA GeForce GTX 1050	NVIDIA Titan RTX	NVIDIA Volta
Cores	4	10	8	640	4608	512
Clock Rate	2.8 GHz	2.2 GHz	2.26 GHz	1.49 GHz	1.77 GHz	1.38 GHz
Multiprocessors	-	-	-	5	72	8
RAM Memory	16 GBytes	181 GBytes	32 GBytes	4042 MBytes	24190 MBytes	15823 MBytes
CUDA Capability	-	-	-	6.1	7.5	7.2

separating crypto implementations from lower level math layers. Our implementation is compiled with CUDA 10.0 NVCC compiler along with C++14 support. Our execution environment consist of 64-bit x86 architecture with operating system for GPU1 and GPU2 as Ubuntu 18.04 and Scientific Linux 6.10 (available on university HPC). Further, we remark that we strictly focus on a single threaded CPU implementation with no hardware optimization. Additionally, on the GPU side our implementations make use of a single GPU device in spite of availability of multiple GPU devices on GPU2.

In the evaluation of Algorithm 1 for bootstrapping of *BV-LWE* scheme with binary secret keys, we used both GPU1 and GPU2. In addition, we used GPU3, NVIDIA Jetson AGX Xavier GPU to investigate the performance on a embedded systems platform. The embedded system is a heterogeneous platform which is interfaced with a ARMv8 processor. The device works on three different pre-configured power modes. We note that our evaluation used the highest power mode for better performance. Complete details of devices are shown in Table 5.4.

In our implementation, we primarily aimed to improve the runtimes of operations pertaining to encryption scheme, PRE scheme and homomorphic operations. To do so, we identified the critical operations in encryption, re-encryption, decryption, re-keygen, addition and multiplication and switched them to CUDA kernel calls. We outline the main aspects of our GPU optimizations as follows:

• **Optimized pre-computation phase**: Our implementation consists of an optimized pre-computation phase wherein we compute most of the cryptosystem parameters and NTT related parameters on CPU and transfer them to GPU global memory. For faster memory transfers, we used coalesced memory spaces which require fewer memory transfer calls. On the CPU side we mostly used memory allocation using pinned host memory. Memory transfer using pinned memory is generally faster than pageable host memory because GPU can directly access such memory spaces. Finally, we remark that we do not make use of constant or texture memory as it reported little or no improvement in performance. Moreover, such memory spaces are available in very limited number and not scalable for higher ring dimensions.

• **Memory related optimizations**: Being a heterogeneous platform, one should expect a significant amount of data transfers between CPU and GPU memory. However, it is known that such data transfers are generally slower (because of lower bandwidth PCIe) and can sometimes degrade the overall performance of application. To get a more realistic performance estimate of various operations, we eliminated most of the data transfers by allocating memory for most of the cryptosystem elements on GPU memory directly.

• **Fast Random Generators**: Our implementation relies on CUDA random number generation library cuRAND for generating random polynomials. The distributions targeted in our application are uniform random distribution $\mathcal{U}_q$, discrete Gaussian distribution χ_e with standard deviation σ_e, binary and ternary uniform distributions. Except for uniform distribution, all other distributions were generated using continuous Gaussian distribution on pre-allocated memory and then launching appropriate kernels to reduce them in a particular range. More specifically, we used the CURAND RNG PSEUDO MTGP32 set of generators which is 5x faster than other random number generators of the same family and atleast 10x faster than CPU random number generators.

• **Relinearization** As described in Section 5.2.4, we benefit significantly by mapping the bit-decomposed polynomials to a three dimensional CUDA kernel wherein NTT procedure can be invoked in parallel. Once the polynomials are evaluated they need to be reduced back into a single polynomial. To do this efficiently, we launch a parallel reduction kernel with grid dimension roughly equal to number of bit-decomposed polynomials. After each kernel call, we obtain the partial results in half the polynomials than we start with. Repeating this process until we reach a single reduced polynomial makes the relinearization process very efficient with only $\mathcal{O}(\log(\log(q)))$ reduction kernel calls.

• **Streams**: On GPUs we can increase the throughput of kernels by launching them simultaneously on independent streams. For example, NVIDIA K20 has the ability to support upto 32 concurrent kernels launched on a separate stream. In our implementation, we use streams mainly for parallel generation of noise, synchronizing NTTs, asynchronous memory transfers and kernel parallelization. For taking advantage of stream APIs, we always keep the ciphertext components independent; for example in Ring-GSW PRE scheme we keep ciphertext as a vector of column polynomials in row major order.

• **Cache optimized multiplication**: We optimized the homomorphic multiplication of a *Ring-GSW* ciphertext with a BV ciphertext by storing the intermediate results on shared memory. These shared memories are on-chip memory spaces with very fast access time and hence reduce the evaluation runtime considerably.

Table 5.5 Experimental Runtime Performance of Encryption, Decryption, and Re-encryption Operation for Ring Dimension n at $r = 1$, $p = 5$, and 128-bits of Security

PRE Scheme	Parameters		Runtime			Throughput		
	n	b	**Enc (ms)**	**Dec (ms)**	**ReEnc (ms)**	**Enc (Kbps)**	**Dec (Kbps)**	**ReEnc (Kbps)**
BV-PRE	512	18	2.14	0.46	0.29	239.25	1113.04	1765.51
	1024	18	2.18	0.52	0.45	469.72	1969.23	2275.55
	2048	19	2.21	0.55	0.85	926.69	3723.63	2409.41
	4096	19	2.28	0.66	1.26	1796.49	6206.06	3250.79
	8192	20	2.76	1.18	2.76	2968.11	6942.37	2968.11
Ring-GSW-PRE	1024	25	3.92	0.6	39.75	261.22	1706.66	25.76
	2048	26	5.34	0.55	61.18	383.52	3723.63	33.47
	4096	27	9.85	0.63	118.86	415.83	6501.58	34.46
	8192	28	19.89	1.08	259.02	411.86	7585.18	31.62

Evaluation Data Reported for GPU1.

Table 5.6 Experimental Runtime Performance of Encryption, Decryption, and Re-encryption Operation for Ring Dimension n at $r = 1$, $p = 5$, and 128-bits of Security

PRE Scheme	Parameters		Runtime			Throughput		
	n	b	**Enc (ms)**	**Dec (ms)**	**ReEnc (ms)**	**Enc (Kbps)**	**Dec (Kbps)**	**ReEnc (Kbps)**
BV-PRE	512	18	4.16	0.47	0.31	123.07	1089.36	1651.61
	1024	18	4.04	0.57	0.4	253.46	1796.49	2560
	2048	19	4.24	0.61	0.69	483.02	3357.37	2968.11
	4096	19	4.3	0.63	0.72	952.55	6501.58	5688.88
	8192	20	4.99	0.68	1.3	1641.68	12047.05	6301.53
Ring-GSW-PRE	1024	25	4.75	0.62	35.46	215.57	1651.61	28.87
	2048	26	5.22	0.67	49.5	392.33	3056.71	41.38
	4096	27	7.98	0.71	87.24	513.28	5769.01	46.95
	8192	28	12.2	0.71	166.67	671.47	11538.03	49.15

Evaluation Data Reported for GPU2.

5.4.2 Experimental Results

For experimental analysis of our PRE schemes, we use latency and throughput as primary yardsticks. A PRE scheme can be divided into a static and dynamic phase. Static phase takes into account all sorts of key generation, pre-computations and parameter setup. Performance of any real time system that uses PRE scheme is largely determined by this dynamic phase where encryption, re-encryption and decryption operations are performed on the fly. Therefore, we only report the runtimes and throughput for dynamic phase operations by varying the ring dimension, n and modulus bit length, b determined as per 128-bit security setting. For recording throughput, we consider messages as binary string with length equal to the ring dimension.

From Tables 5.5 and 5.6, we can observe that encryption and decryption runtimes for BV-PRE scheme vary in very small amount with increasing ring dimensions. We remark that encryption runtimes for smaller ring dimensions are still slower with break even occuring for $n = 2048$ and hence encryption for smaller ring dimensions are more suitable for CPU platforms. Re-encryption runtimes for BV-PRE scheme increases linearly with ring dimensions but still doesn't grow more than twice as observed on CPU implementations. Comparing our results with Table 6 [PRSV17] for re-encryption runtimes, we get a performance improvement by a factor of 39x to 228x. Similarly, we get a peak throughput of 6.3 Mbps for BV-PRE re-encryption procedure from GPU2. We remark that the actual runtimes of BV-PRE in the current release of PALISADE has been improved since the publication of [PRSV17].

For Ring-GSW PRE scheme, decryption runtimes are slightly higher than that of BV PRE scheme because of relatively large modulus bit lengths. Further, decryption runtimes vary very little and can be treated as constant for all practical purposes. Re-encryption runtimes are drastically higher and consequently throughput reduced when compared to BV PRE re-encryption runtimes and throughput. This is due to the fact that relinearization procedure is performed over multiple rows of ciphertext matrix. However, when compared to CPU implementation we still get performance improvement of 3.5x to 11x. Runtimes of both BV PRE and Ring-GSW PRE schemes can be further brought down by considering a larger relinearization window but is accompanied with larger noise growth and error bounds.

Table 5.7 Experimental Runtime Performance of BV-LWE Bootstrapping Algorithm 1 for Varying Ring Dimension, N, Relinearization Factor, r and GPUs

N	r	Q bits	Bootstrapping KeyGen (in s)			Bootstrapping Runtime (in s)		
			GPU1	GPU2	GPU3	GPU1	GPU2	GPU3
512	1	30	1.75	1.35	3.36	0.57	0.25	2.25
	2	30	1.43	1.32	3.32	0.41	0.22	1.97
	4	30	1.27	1.32	2.67	0.32	0.21	1.21
1024	1	32	2.58	1.44	5.9	0.93	0.32	2.65
	2	32	1.94	1.42	3.52	0.65	0.28	1.66
	4	32	1.57	1.42	2.62	0.48	0.26	1.24

In the evaluation of bootstrapping algorithm for *BV-LWE* scheme, we used very low relinearization values, $r = \{1, 2, 4\}$. These small relinearization factors use relatively low base 2^r to decompose polynomials and hence the noise growth is considerably less. The overall impact is that a lower ciphertext modulus (within machine word size) can be used however the runtimes is increased. In such cases, a hardware accelerator such as GPU can be more useful. From Table 5.7, we can observe that for GPU1 and GPU2 the overall runtimes stay under a second. In case of the embedded GPU platform, GPU3, we get reduced performance on account of power and architecture constraints. For these small relinearization factors, the runtimes are faster than CPU runtimes by a factor of 4x or more. Further, with very high relinearization factors ($r > 9$) CPU implementation are slightly faster but use higher ciphertext modulus values.

5.5 Conclusion

In this work, we explored GPU acceleration of BV-PRE, Ring-GSW PRE schemes and bootstrapping algorithm for BV-LWE schemes and showed that GPUs are indeed capable of improving performance by more than an order of magnitude. Moreover, from our experiments we found that GPUs are more effective in working with larger ring dimensions. In future we would like to explore hardware acceleration of other FHE applications and primitives through our optimized and parallel software library.

REFERENCES

[ABB10a] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient Lattice (H)IBE in the Standard Model. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 553–572, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[ABB10b] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice Basis Delegation in Fixed Dimension and Shorter-Ciphertext Hierarchical IBE. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 98–115, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[ABD16] Martin Albrecht, Shi Bai, and Léo Ducas. A Subfield Lattice Attack on Overstretched NTRU Assumptions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 153–178, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[ABPW13] Yoshinori Aono, Xavier Boyen, Le Trieu Phong, and Lihua Wang. Key-Private Proxy Re-encryption under LWE. In Goutam Paul and Serge Vaudenay, editors, *Progress in Cryptology – INDOCRYPT 2013*, pages 1–18, Cham, 2013. Springer International Publishing.

[ACC$^+$18] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic Encryption Security Standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[ACF$^+$15] Martin R Albrecht, Carlos Cid, Jean-Charles Faugere, Robert Fitzpatrick, and Ludovic Perret. On the Complexity of the BKW algorithm on LWE. *Designs, Codes and Cryptography*, 74(2):325–354, 2015.

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. PIR with Compressed Queries and Amortized Query Processing. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 962–979, 2018.

[AD97] Miklós Ajtai and Cynthia Dwork. A Public-key Cryptosystem with Worst-case/Average-case Equivalence. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, pages 284–293, 1997.

[AFFP11] Martin R Albrecht, Pooya Farshim, Jean-Charles Faugere, and Ludovic Perret. Polly Cracker, Revisited. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 179–196. Springer, 2011.

[AFGH06] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. *ACM Transactions on Information and System Security (TISSEC)*, 9(1):1–30, 2006.

[Ajt96] Miklós Ajtai. Generating Hard Instances of Lattice Problems. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, pages 99–108, 1996.

[Ajt98] Miklós Ajtai. The Shortest Vector Problem in L2 is NP-hard for Randomized Reductions. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 10–19, 1998.

[AMBFK16] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. Xpir: Private Information Retrieval for Everyone. *Proceedings on Privacy Enhancing Technologies*, 2016(2):155–174, 2016.

[APS13] Aydin Aysu, Cameron Patterson, and Patrick Schaumont. Low-cost and Area-efficient FPGA Implementations of Lattice-based Cryptography. In *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, pages 81–86, June 2013.

[ASP14] Jacob Alperin-Sheriff and Chris Peikert. Faster Bootstrapping with Polynomial Error. In *Annual Cryptology Conference*, pages 297–314. Springer, 2014.

[Bab86] László Babai. On Lovász Lattice Reduction and the Nearest Lattice Point Problem. *Combinatorica*, 6(1):1–13, 1986.

[Bar86] Paul Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 311–323. Springer, 1986.

[Bar89] David A Barrington. Bounded-width Polynomial-size Branching Programs Recognize Exactly those Languages in NC1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.

[BBC$^+$10] Mauro Barni, Tiziano Bianchi, Dario Catalano, Mario Di Raimondo, Ruggero Donida Labati, Pierluigi Failla, Dario Fiore, Riccardo Lazzeretti, Vincenzo Piuri, Fabio Scotti, et al. Privacy-preserving Fingercode Authentication. In *Proceedings of the 12th ACM Workshop on Multimedia and Security*, pages 231–240, 2010.

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible Protocols and Atomic Proxy Cryptography. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 127–144. Springer, 1998.

[BDF18] Guillaume Bonnoron, Léo Ducas, and Max Fillinger. Large FHE Gates from Tensored Homomorphic Accumulator. In *International Conference on Cryptology in Africa*, pages 217–251. Springer, 2018.

[BGG$^+$16] Johannes Buchmann, Florian Göpfert, Tim Güneysu, Tobias Oder, and Thomas Pöppelmann. High-Performance and Lightweight Lattice-Based Public-Key Encryption. In *Proceedings of the 2nd ACM International Workshop on IoT Privacy, Trust, and Security (IoTPTS'16)*, pages 2–9, 2016.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In *Theory of Cryptography Conference*, pages 325–341. Springer, 2005.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6:13, 2014.

[BHHH19] Flavio Bergamaschi, Shai Halevi, Tzipora T Halevi, and Hamish Hunt. Homomorphic Training of 30,000 Logistic Regression Models. In *International Conference on Applied Cryptography and Network Security*, pages 592–611. Springer, 2019.

[BLLN13] Joppe W Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved Security for a Ring-based Fully Homomorphic Encryption Scheme. In *IMA International Conference on Cryptography and Coding*, pages 45–64. Springer, 2013.

[BLN14] Joppe W Bos, Kristin Lauter, and Michael Naehrig. Private Predictive Analysis on Encrypted Medical Data. *Journal of Biomedical Informatics*, 50:234–243, 2014.

[BLP$^+$13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical Hardness of Learning with Errors. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, pages 575–584. ACM, 2013.

[BPTG15] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine Learning Classification over Encrypted Data. In *NDSS*, volume 4324, page 4325, 2015.

[BR15] Jean-François Biasse and Luis Ruiz. FHEW with Efficient Multibit Bootstrapping. In *International Conference on Cryptology and Information Security in Latin America*, pages 119–135. Springer, 2015.

[Bra12] Zvika Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 97–106, Washington, DC, USA, 2011. IEEE Computer Society.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In *Annual Cryptology Conference*, pages 505–524. Springer, 2011.

[BV14a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. *SIAM Journal on Computing*, 43(2):831–871, 2014.

[BV14b] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as Secure as PKE. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, pages 1–12. ACM, 2014.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster Fully Homomorphic Encryption: Bootstrapping in less than 0.1 Seconds. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–33. Springer, 2016.

[CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 377–408. Springer, 2017.

[CGH$^+$18] Jack LH Crawford, Craig Gentry, Shai Halevi, Daniel Platt, and Victor Shoup. Doing Real Work with FHE: The case of Logistic Regression. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 1–12. ACM, 2018.

[CH07] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 185–194. ACM, 2007.

[CH18] Hao Chen and Kyoohyung Han. Homomorphic Lower Digits Removal and Improved FHE Bootstrapping. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 315–337. Springer, 2018.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai Trees, or How to Delegate a Lattice Basis. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 523–552. Springer, 2010.

[CJL16] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An Algorithm for NTRU Problems and Cryptanalysis of the GGH Multilinear Map without a Low-level Encoding of Zero. *LMS Journal of Computation and Mathematics*, 19(A):255–266, 2016.

[CM15] Michael Clear and Ciaran McGoldrick. Multi-identity and Multi-key Leveled FHE from Learning with Errors. In *Annual Cryptology Conference*, pages 630–656. Springer, 2015.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better Lattice Security Estimates. In *ASIACRYPT*, pages 1–20, 2011.

[DDS14] Wei Dai, Yarkın Doröz, and Berk Sunar. Accelerating NTRU based Homomorphic Encryption using GPUs. In *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2014.

[DGHV10] Marten Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully Homomorphic Encryption over the Integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 24–43. Springer, 2010.

[DHS14] Yarkin Doröz, Yin Hu, and Berk Sunar. Homomorphic AES Evaluation using NTRU. *IACR Cryptology ePrint Archive*, 2014:39, 2014.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less than a Second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 617–640. Springer, 2015.

[DQ98] J-F Dhem and J-J Quisquater. Recent Results on Modular Multiplications for Smart Cards. In *International Conference on Smart Card Research and Advanced Applications*, pages 336–352. Springer, 1998.

[DQ00] Jean-François Dhem and Jean-Jacques Quisquater. Recent Results on Modular Multiplications for Smart Cards. In Jean-Jacques Quisquater and Bruce Schneier, editors, *Smart Card Research and Applications*, volume 1820 of *Lecture Notes in Computer Science*, pages 336–352. Springer Berlin Heidelberg, 2000.

[DS15] Wei Dai and Berk Sunar. cuhe: A Homomorphic Encryption Accelerator Library. In *International Conference on Cryptography and Information Security in the Balkans*, pages 169–186. Springer, 2015.

[EFG$^+$09] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving Face Recognition. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 235–253. Springer, 2009.

[EHKM11] David Evans, Yan Huang, Jonathan Katz, and Lior Malka. Efficient Privacy-preserving Biometric Identification. In *Proceedings of the 17th Conference Network and Distributed System Security Symposium, NDSS*, volume 68, 2011.

[ElG85] Taher ElGamal. A Public key Cryptosystem and a Signature Scheme based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[FL16a] Xiong Fan and Feng-Hao Liu. Various Proxy Re-Encryption Schemes from Lattices. *IACR Cryptology ePrint Archive*, 2016:278, 2016.

[FL16b] Xiong Fan and Feng-Hao Liu. Various Proxy Re-Encryption Schemes from Lattices. Cryptology ePrint Archive, Report 2016/278, 2016. `http://eprint.iacr.org/2016/278`.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[G$^+$09] Craig Gentry et al. Fully Homomorphic Encryption using Ideal Lattices. In *STOC*, volume 9, pages 169–178, 2009.

[GBDL$^+$16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.

[Gen09] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, Stanford, CA, USA, 2009.

[GH11] Craig Gentry and Shai Halevi. Implementing Gentrys Fully-homomorphic Encryption Scheme. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 129–148. Springer, 2011.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P Smart. Better Bootstrapping in Fully Homomorphic Encryption. In *International Workshop on Public Key Cryptography*, pages 1–16. Springer, 2012.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P Smart. Fully Homomorphic Encryption with Polylog Overhead. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 465–482. Springer, 2012.

[GHS12c] Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic Evaluation of the AES Circuit. In *Annual Cryptology Conference*, pages 850–867. Springer, 2012.

[GINX16] Nicolas Gama, Malika Izabachène, Phong Q Nguyen, and Xiang Xie. Structural Lattice Reduction: Generalized Worst-case to Average-case Reductions and Homomorphic Cryptosystems. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 528–558. Springer, 2016.

[GKPV10] Shafi Goldwasser, Tauman Yael Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the Learning with Errors Assumption. *International Conference on Supercomputing*, pages 230–240, 2010.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption &amp; how to play Mental Poker keeping Secret all Partial Information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 365–377, 1982.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for Hard Lattices and New Cryptographic Constructions. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, pages 197–206. ACM, 2008.

[GR15] Paolo Gasti and Kasper B Rasmussen. Privacy-preserving User Matching. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*, pages 111–120, 2015.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-simpler, Asymptotically-faster, Attribute-based. In *Advances in Cryptology–CRYPTO 2013*, pages 75–92. Springer, 2013.

[HG01] Nick Howgrave-Graham. Approximate Integer Common Divisors. In *International Cryptography and Lattices Conference*, pages 51–66. Springer, 2001.

[Hoc12] Robert Hochberg. Matrix Multiplication with CUDA-a basic Introduction to the CUDA Programming Model. *Internet": http://www. shodor. org/media/content/petascale/materials/UPModules/matrixMultiplication/moduleDocument. pdf,[August 11 2012]*, 2012.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. Ntru: A Ring-based Public Key Cryptosystem. In *International Algorithmic Number Theory Symposium*, pages 267–288. Springer, 1998.

[HS14] Shai Halevi and Victor Shoup. Algorithms in HeLib. In *Annual Cryptology Conference*, pages 554–571. Springer, 2014.

[HS15] Shai Halevi and Victor Shoup. Bootstrapping for HeLib. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 641–670. Springer, 2015.

[ID03] Anca-Andreea Ivan and Yevgeniy Dodis. Proxy Cryptography Revisited. In *NDSS*, 2003.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A Low latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1651–1669, 2018.

[KGV16] Alhassan Khedr, Glenn Gulak, and Vinod Vaikuntanathan. SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers. *IEEE Transactions on Computers*, 65(9):2848–2858, 2016.

[KHP06] Himanshu Khurana, Jin Heo, and Meenal Pant. From Proxy Encryption Primitives to a Deployable Secure-mailing-list Solution. In *International Conference on Information and Communications Security*, pages 260–281. Springer, 2006.

[Kir14] Elena Kirshanova. Proxy Re-encryption from Lattices. In *International Workshop on Public Key Cryptography*, pages 77–94. Springer, 2014.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC 12, page 12191234, New York, NY, USA, 2012. Association for Computing Machinery.

[LdVMPT09] Françoise Levy-dit Vehel, Maria Grazia Marinari, Ludovic Perret, and Carlo Traverso. A Survey on Polly Cracker Systems. In *Gröbner Bases, Coding, and Cryptography*, pages 285–305. Springer, 2009.

[LL15] Kim Laine and Kristin Lauter. Key Recovery for LWE in Polynomial Time. *IACR Cryptology ePrint Archive*, October 2015.

[LLL82] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische annalen*, 261(ARTICLE):515–534, 1982.

[LP11] Richard Lindner and Chris Peikert. Better Key Sizes (and attacks) for LWE-based Encryption. In *Cryptographers Track at the RSA Conference*, pages 319–339. Springer, 2011.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A Toolkit for Ring-LWE Cryptography. In *Advances in Cryptology (EUROCRYPT'13)*, volume 7881, pages 35–54. Springer, 2013.

[LSSR$^+$15] Zhe Liu, Hwajeong Seo, Sujoy Sinha Roy, Johann Großschädl, Howon Kim, and Ingrid Verbauwhede. *Efficient Ring-LWE Encryption on 8-Bit AVR Processors*, pages 663–682. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[LTV13] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. Multikey Fully Homomorphic Encryption and On-the-Fly Multiparty Computation. *IACR Cryptology ePrint Archive*, 2013:94, 2013. Full Version of the STOC 2012 paper with the same title.

[Mic01] Daniele Micciancio. The Shortest Vector in a Lattice is hard to Approximate to within some Constant. *SIAM Journal on Computing*, 30(6):2008–2035, 2001.

[Mic10] Daniele Micciancio. Duality in Lattice Cryptography. In *Public Key Cryptography (PKC'10)*, 2010. Invited talk.

[Mic18] Daniele Micciancio. On the Hardness of Learning with Errors with Binary Secrets. *Theory of Computing*, 14(1):1–17, 2018.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 700–718. Springer, 2012.

[MP20] Daniele Micciancio and Yuriy Polyakov. Bootstrapping in FHEW-like Cryptosystems. *IACR Cryptology. ePrint Arch.*, 2020:86, 2020.

[MR07] Daniele Micciancio and Oded Regev. Worst-Case to Average-Case Reductions Based on Gaussian Measures. *SIAM J. Comput.*, 37(1):267–302, 2007. Preliminary version in FOCS 2004.

[MR09] Daniele Micciancio and Oded Regev. Lattice-based Cryptography. In *Post-quantum Cryptography*, pages 147–191. Springer, 2009.

[MS18] Daniele Micciancо and Jessica Sorrell. Ring Packing and Amortized FHEW Bootstrapping. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two Round Multiparty Computation via Multi-key FHE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 735–763. Springer, 2016.

[NAL15] David Nuñez, Isaac Agudo, and Javier Lopez. Ntrureencrypt: An Efficient Proxy Re-encryption Scheme based on NTRU. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pages 179–189. ACM, 2015.

[NLV11] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can Homomorphic Encryption be Practical? In *Proceedings of the 3rd ACM workshop on Cloud Computing Security Workshop*, pages 113–124, 2011.

[Pai99] Pascal Paillier. Public-key Cryptosystems based on Composite Degree Residuosity Classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[Pei09] Chris Peikert. Public-key Cryptosystems from the Worst-case Shortest Vector Problem. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, pages 333–342. ACM, 2009.

[Pei10] Chris Peikert. An Efficient and Parallel Gaussian Sampler for Lattices. In Tal Rabin, editor, *Advances in Cryptology CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 80–97. Springer Berlin Heidelberg, 2010.

[Pei16] Chris Peikert. A Decade of Lattice Cryptography. *Foundations and Trends® in Theoretical Computer Science*, 10(4):283–424, 2016.

[PRSV17] Yuriy Polyakov, Kurt Rohloff, Gyana Sahu, and Vinod Vaikuntanathan. Fast Proxy Re-encryption for Publish/Subscribe Systems. *ACM Transactions on Privacy and Security (TOPS)*, 20(4):14, 2017.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A Framework for Efficient and Composable Oblivious Transfer. In *Annual International Cryptology Conference*, pages 554–571. Springer, 2008.

[PWA$^+$16] L Phong, L Wang, Y Aono, M Nguyen, and X Boyen. Proxy Re-encryption Schemes with Key Privacy from LWE. Technical report, IACR Cryptology ePrint Archive 2016/327, 2016.

[RAD$^+$78] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On Data banks and Privacy Homomorphisms. *Foundations of Secure Computation*, 4(11):169–180, 1978.

[Reg04] Oded Regev. Quantum Computation and Lattice Problems. *SIAM Journal on Computing*, 33(3):738–760, 2004.

[Reg09] Oded Regev. On Lattices, Learning with Errors, Random Linear codes, and Cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[Sho99] Peter W Shor. Polynomial-time algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM review*, 41(2):303–332, 1999.

[SR20] Gyana Sahu and Kurt Rohloff. Construction and Evaluation of Proxy Re-Encryption on GSW FHE Scheme and other Primitives. *IEEE Transactions on Dependable and Secure Computing*, 2020. Manuscript submitted to IEEE journal of Transactions on Dependable and Secure Computing.

[SS11] Damien Stehlé and Ron Steinfeld. Making NTRU as Secure as Worst-case Problems over Ideal Lattices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 27–47. Springer, 2011.

[SSW09] Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Efficient Privacy-preserving Face Recognition. In *International Conference on Information Security and Cryptology*, pages 229–244. Springer, 2009.

[SV10] Nigel P Smart and Frederik Vercauteren. Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. In *International Workshop on Public Key Cryptography*, pages 420–443. Springer, 2010.

[SV14] Nigel P Smart and Frederik Vercauteren. Fully Homomorphic SIMD Operations. *Designs, Codes and Cryptography*, 71(1):57–81, 2014.

[TCG06] Gelareh Taban, Alvaro A Cárdenas, and Virgil D Gligor. Towards a Secure and Interoperable DRM Architecture. In *Proceedings of the ACM Workshop on Digital Rights Management*, pages 69–78. ACM, 2006.

[vdP12] Joop van de Pol. Quantifying the Security of Lattice-Based Cryptosystems in Practice. In *Mathematical and Statistical Aspects of Cryptography*, 2012.

[WH12] David Wu and Jacob Haven. Using Homomorphic Encryption for Large Scale Statistical Analysis, 2012.

[Yao82] Andrew C Yao. Protocols for Secure Computations. In *23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)*, pages 160–164. IEEE, 1982.

[Yao86] Andrew Chi-Chih Yao. How to Generate and Exchange Secrets. In *27th Annual Symposium on Foundations of Computer Science (SFCS 1986)*, pages 162–167. IEEE, 1986.

[YBG$^+$15] Xun Yi, Athman Bouguettaya, Dimitrios Georgakopoulos, Andy Song, and Jan Willemson. Privacy Protection for Wireless Medical Sensor Data. *IEEE Transactions on Dependable and Secure Computing*, 13(3):369–380, 2015.

[ZZZ$^+$13] Rui Zhang, Jinxue Zhang, Yanchao Zhang, Jinyuan Sun, and Guanhua Yan. Privacy-preserving Profile Matching for Proximity-based Mobile Social Networking. *IEEE Journal on Selected Areas in Communications*, 31(9):656–668, 2013.