Homomorphic Encryption

by

Brandon Weir

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2013

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

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

Abstract

In this thesis, we provide a summary of fully homomorphic encryption, and in particular, look at the BGV encryption scheme by Brakerski, Gentry, and Vaikuntanathan; as well the DGHV encryption scheme by van Dijk, Gentry, Halevi, and Vaikuntanathan. We explain the mechanisms developed by Gentry in his breakthrough work, and show examples of how they are used.

While looking at the BGV encryption scheme, we make improvements to the underlying lemmas dealing with modulus switching and noise management, and show that the lemmas as currently stated are false. We then examine a lower bound on the hardness of the Learning With Errors lattice problem, and use this to develop specific parameters for the BGV encryption scheme at a variety of security levels.

We then study the DGHV encryption scheme, and show how the somewhat homomorphic encryption scheme can be implemented as both a fully homomorphic encryption scheme with bootstrapping, as well as a leveled fully homomorphic encryption scheme using the techniques from the BGV encryption scheme. We then extend the parameters from the optimized version of this scheme to higher security levels, and describe a more straightforward way of arriving at these parameters.

Acknowledgements

I would like to thank my supervisor Dr. Edlyn Teske-Wilson for her support and continued guidance. Her efforts and help have not only led me in the right direction to succeed, but have also provided me with the tools and insight that I needed to progress. I would not have been able to accomplish this task without her.

I would also like to thank the Combinatorics and Optimization Department, and specifically the Cryptography Group, for giving me the mathematical foundations that I was required to build upon while completing my thesis.

I would also like to thank my friends and family for their support throughout my time at the University of Waterloo.

Dedication

I would like to dedicate this thesis to my parents, and my brother Nathan. They have been a source of encouragement to me whenever I needed it.

Table of Contents

List of Tables

Chapter 1

Introduction

Homomorphisms in Encryption Schemes

A group homomorphism is a function f from a group $(G, \cdot)$ to another group $(H, \cdot)$ that is structure preserving. That is, for $g_1, g_2 \in G$ such that $f(g_1) = h_1 \in H$ and $f(g_2) = h_2 \in H$, we have $f(g_1 \cdot g_2) = f(g_1) \cdot f(g_2)$. We say that an encryption scheme is homomorphic if its encryption scheme is a homomorphism. There are a number of homomorphic encryption schemes, including ElGamal, RSA, and the Paillier cryptosystem. Given two RSA ciphertexts $c_1 = m_1^e \bmod n$ and $c_2 = m_2^e \bmod n$, we are able to compute $c = c_1 \cdot c_2 = m_1^e \cdot m_2^e = (m_1 \cdot m_2)^e \bmod n$, which is the value of $m_1 \cdot m_2$ encrypted. Similarly for the ElGamal cryptosystem, given two ciphertexts $c_1 = (g^{r_1}, m_1 \cdot (g^s)^{r_1})$ and $c_2 = (g^{r_2}, m_2 \cdot (g^s)^{r_2})$, we can compute $c_1 \cdot c_2 = (g^{r_1} \cdot g^{r_2}, m_1 \cdot (g^s)^{r_1} \cdot m_2 \cdot (g^s)^{r_2}) = (g^{(r_1+r_2)}, (m_1 \cdot m_2) \cdot (g^s)^{(r_1+r_2)})$, which is the ciphertext corresponding to $m_1 \cdot m_2$. These schemes are called multiplicatively homomorphic because we can compute the product of two ciphertexts, and this gives us the encrypted value of the product of the original messages.

Just as there are multiplicatively homomorphic encryption schemes, there are also additively homomorphic encryption schemes, such as the Paillier cryptosystem [5]. An additively homomorphic cryptosystem is able to compute the sum of two ciphertexts, and have this correspond to the encrypted value of the sum of the original plaintexts. For instance, given two Paillier ciphertexts $c_1 = g^{m_1} \cdot r_1^N \bmod N^2$ and $c_2 = g^{m_2} \cdot r_2^N \bmod N^2$, one can compute $c = c_1 \cdot c_2 = g^{m_1} \cdot r_1^N \cdot g^{m_2} \cdot r_2^N \bmod N^2 = g^{(m_1+m_2)} \cdot (r_1 \cdot r_2)^N \bmod N^2$, which is the encrypted value of $m_1 + m_2$.

Fully Homomorphic Encryption Schemes

An encryption scheme that is both multiplicatively homomorphic and additively homomorphic is called fully homomorphic. Such an encryption scheme can be very useful because this would allow us to compute arbitrary polynomials on encrypted data without ever having to decrypt it first. This could be used to allow a party with sensitive data to enlist the help of an external company to help process its data without divulging its secrets to this external company. Concretely, the military may want to set up some supply depots that are placed in an optimal position given the current locations of their armed forces. They could then enlist the help of an optimization company, and give them the encrypted locations of their forces. The optimization company could process this information to provide an optimal solution for the military, without ever knowing the sensitive military information, nor the results that they provided. This would allow organizations with highly classified information to outsource, instead of requiring people with specialized knowledge bases to have the proper security clearance.

The idea of a fully homomorphic encryption scheme was originally introduced by Rivest, Adleman, and Dertouzos in their 1978 paper entitled "On Data Banks and Privacy Homomorphisms" [19], where they initially referred to fully homomorphic encryption functions as privacy homomorphisms.

Their paper proposes a situation where a loan company enlists the services of a third party time-sharing company to store and process the accumulated loan data. However, due to the sensitivity of the loan company's data, everything must be encrypted in their data banks provided by the time-sharing company. If the loan company wanted to know how much the average loan was, or how many loans over a certain value had been granted, they would have to do some computations on their data. However, to do any computations on their data, the loan company would have to either decrypt the data on their end, and process the data themselves, or allow the third party company to have some decrypted data. If the loan company doesn't have the resources to perform these operations on their end, they would have to compute these statistics on the side of the data storage company.

If the loan company was using a fully homomorphic encryption scheme however, they could compute all of the information they need on the encrypted data, and then output the needed information in encrypted form. This would prevent the data storage company from seeing any secret information, but would still allow the loan company to make full use of the data storage company's resources.

Gentry's Breakthrough

Before 2009, no one was able to come up with a workable suggestion for a fully homomorphic encryption scheme. However, that year, Gentry published his PhD thesis that described the first fully homomorphic encryption scheme [8]. His scheme used similar ideas as that of the Goldreich, Goldwasser, and Halevi (GGH) cryptosystem [14], [5], except using ideal lattices. Every encrypted message contains some sort of error term, and when these encrypted messages are added or multiplied, the error term grows. After the error terms get too big, the encryption scheme is no longer able to correctly decrypt the messages. The idea of the scheme was to create a "somewhat" homomorphic encryption scheme that could handle a certain number of homomorphic operations, and then shrink the error terms.

To accomplish this, the somewhat homomorphic encryption scheme needed to be able to evaluate its own decryption circuit (plus another operation) so that a message that was encrypted twice could have its initial encryption that has high error value removed while still being encrypted with the second encryption. This would leave the ciphertext just with the error associated with the second encryption. This technique was called bootstrapping.

Unfortunately, the decryption circuit required too many operations to perform homomorphically. To solve this issue, Gentry proposed "squashing" the decryption circuit. To do this, some extra information is given to the ciphertexts about the private key, so the decryption circuit doesn't have to do as much homomorphically.

Since 2009, there have been a number of other fully homomorphic schemes created using the same techniques. The main cryptosystems we will be focusing on are the BGV encryption scheme which uses modular lattices and the Learning with Errors Problem to create a leveled fully homomorphic encryption scheme, and the DGHV encryption scheme which is a fully homomorphic cryptosystem based on integers.

Outline of Thesis

In the Preliminaries chapter, we first define the notation that we will be using throughout this thesis. We then provide an overview of lattices, their bases, and the ways that the orthogonality of these bases can be measured. We then look at some of the more well known lattice-based hard problems, as well as some other hard problems used in the cryptosystems covered. A discussion on lattice reduction techniques is also included.

Chapter 3 is an overview of what it means for a cryptosystem to be fully homomorphic. We provide a number of definitions of various properties needed for an encryption scheme to be fully homomorphic. A look at leveled fully homomorphic encryption is provided,

as well as the requirements and implementation of bootstrapping needed to create a fully homomorphic encryption scheme from a somewhat homomorphic encryption scheme.

In Chapter 4 we take a closer look at the lattice-based encryption scheme BGV. We describe the somewhat fully homomorphic scheme, and then extend this to a leveled fully homomorphic encryption scheme that does not incorporate bootstrapping. We then look into the correctness of the scheme, and determine properties that the error function needs to satisfy in order for decryption to work correctly. We then examine the proofs described in the BGV paper related to the modulus switching function, and correct the statements of these proofs. To conclude this chapter, we study an experimental lower bound for the running time needed to solve the underlying lattice problem. Using this, we compute parameters for each security level.

In Chapter 5 we move from a lattice-based cryptosystem to DGHV, which is an integer-based cryptosystem. Here we look at the original somewhat homomorphic cryptosystem and, as in the previous chapter, we extend this to a leveled fully homomorphic encryption scheme. However, we also study a bootstrappable variant of the scheme, and an optimized version of this variant. Then, considering the attacks used by Coron, Mandal, Naccache, and Tibouchi [5], we extend the parameters needed for high security levels.

Finally, in Chapter 6, we take a brief look at a few other encryption schemes, including the first homomorphic encryption scheme created by Craig Gentry.

New Contributions

In this thesis, we make corrections to Lemmas 1 and 4 from the paper introducing the BGV encryption scheme (4.3) [2]. These lemmas are used to show that the modulus switching technique keeps the noise of the encrypted data at a reasonable size. However, the inequalities were incorrectly stated. We found counter examples to the original proofs, and were able to correct the statements of the lemmas and provide proofs of the corrected statements.

Also, we use the analysis by Gentry, Halevi, and Smart [13] to determine parameters that are needed for different security levels in the BGV encryption scheme (4.4). However, we generalize their calculations to cases where the standard deviation on the error distribution is not 1. In these parameters, we also include the maximum size the error term can take.

We also extend the parameters for the optimized version of the DGHV scheme by Coron, Mandal, Naccache, and Tibouchi [5] to higher security levels (5.4.5). While doing

this, we provide a more direct method to compute these parameters for arbitrary security levels.

Chapter 2

Preliminaries

This chapter presents the notation and underlying mathematical concepts that will be used throughout this thesis. The presentation includes lattices and the hard problems that underlie the security of fully homomorphic encryption schemes.

2.1 Notation

Throughout this paper, we follow similar notation as defined in the paper by Brakerski, Gentry, and Vaikuntanathan [2].

Define R to be the ring $\mathbb{Z}[x]/(x^d+1)$. For any element $r \in R$, we will use r_j to represent the coefficient of x^j of r (i.e. $r = r_0 + r_1 x + r_2 x^2 + \cdots + r_{d-1} x^{d-1}$). We will denote vectors as lowercase letters in bold, where $\mathbf{v}[i]$ denotes the i'th component of vector $\mathbf{v}$. Thus, if $\mathbf{v} \in R^n$, then $\mathbf{v}[i]_j$ is the coefficient of x^j in the ring element $\mathbf{v}[i]$, which is the i'th component of $\mathbf{v}$. All vectors are column vectors unless otherwise specified. For a matrix A, let A_i be the i'th column vector of A.

For two vectors $\mathbf{u}, \mathbf{v} \in R^n$, define the dot product $\mathbf{u} \cdot \mathbf{v} = \sum_{k=1}^{n} \mathbf{u}[k] \cdot \mathbf{v}[k]$ where $\mathbf{u} \cdot \mathbf{v} \in R$. For $r \in R$, we define $\|r\|$ to be the Euclidean norm of the coefficient vector for r, that is $\|r\| = \sqrt{\sum_{j=0}^{d-1} r_j^2}$. We also define the l_1^R norm of a vector $\mathbf{v} \in R^n$ to be $\|\mathbf{v}\| = \sum_{i=1}^{n} \|\mathbf{v}[i]\|$, which is just the standard l_1 norm when $d = 1$.

For any $q \in \mathbb{Z}$ such that $q \geq 2$, define R_q to be R/qR. For any $r \in R$, we define $[r]_q$ to be r mod q such that the coefficients r_j of r are reduced to within the range $(-q/2, q/2]$. Also, for $x \in \mathbb{Z}$, define $\lceil x \rceil, \lfloor x \rfloor, \lfloor x \rceil$ to be x rounded up to the nearest integer, x rounded down to the nearest integer, and x rounded to the nearest integer respectively. Let $[x] = x - \lfloor x \rceil$ be the fractional part of x.

Define the function $\Xi : R^n \to R^{\binom{n+1}{2}}$ to be the tensor product of a vector in a ring R such that any repeated terms are excluded. For convention, we will always exclude the second like term in the standard tensor product. For example, $\Xi((1,2,3)^T) = (1 \cdot 1, 1 \cdot 2, 1 \cdot 3, 2 \cdot 2, 2 \cdot 3, 3 \cdot 3)^T = (1,2,3,4,6,9)^T$. Also, define $\otimes$ to be the standard tensor product binary operator.

We define $\gamma_R = \max\{\|a \cdot b\|/(\|a\|\|b\|) : a, b \in R\}$ to be the *expansion factor* γ_R of the ring R.

Lemma 2.1.1. *The expansion factor γ_R satisfies $0 \leq \gamma_R \leq \sqrt{d}$.*

Proof. Let a, $b \in R$ where $R = \mathbb{Z}[x]/(x^d + 1)$. Say $a = a_0 + a_1 x + \cdots + a_{d-1}x^{d-1}$ and $b = b_0 + b_1 x + \cdots + b_{d-1}x^{d-1}$. We first need to compute $a \cdot b$ over R.

$$\begin{aligned}
a \cdot b &= \sum_{i=0}^{d-1}\sum_{j=0}^{d-1} a_i b_j x^{i+j} \\
&= \sum_{k=0}^{d-1}\left(\sum_{j=0}^{k} a_{k-j}b_j\right)x^k + \sum_{k=d}^{2d-2}\left(\sum_{j=k-d+1}^{d-1} a_{k-j}b_j\right)x^k \\
&= \sum_{k=0}^{d-1}\left(\sum_{j=0}^{k} a_{k-j}b_j\right)x^k - \sum_{k=d}^{2d-2}\left(\sum_{j=k-d+1}^{d-1} a_{k-j}b_j\right)x^{k-d} \\
&= \sum_{k=0}^{d-1}\left(\sum_{j=0}^{k} a_{k-j}b_j\right)x^k - \sum_{k=0}^{d-2}\left(\sum_{j=k+1}^{d-1} a_{k+d-j}b_j\right)x^k \\
&= \sum_{k=0}^{d-1}\left(\sum_{j=0}^{k} a_{k-j}b_j - \sum_{j=k+1}^{d-1} a_{k+d-j}b_j\right)x^k.
\end{aligned}$$

Now, let $\mathbf{a}'$ be the coefficient vector of a in absolute values, that is $\mathbf{a}' = (|a_0|, |a_1|, \ldots, |a_{d-1}|)$, and likewise, $\mathbf{b}' = (|b_0|, |b_1|, \ldots, |b_{d-1}|)$. Also, let

$$a^*_{k,j} = \begin{cases} a_{k-j} & : \ 0 \leq j \leq k \\ a_{k+d-j} & : \ k+1 \leq j \leq d-1 \end{cases}$$

and define $\mathbf{a}^*_k = (|a^*_{k,0}|, |a^*_{k,1}|, \ldots, |a^*_{k,d-1}|)$. Notice that the map from a_j to $a^*_{k,j}$ is a permutation of the terms in the coefficient vector of a, so we get that $\|\mathbf{a}'\| = \|\mathbf{a}^*_k\|$ for all $k \in \{0, \ldots, d-1\}$. Then, we get

$$\begin{aligned}
\|a \cdot b\|^2 &= \sum_{k=0}^{d-1} \left(\sum_{j=0}^{k} a_{k-j} b_j - \sum_{j=k+1}^{d-1} a_{k+d-j} b_j \right)^2 \\
&\leq \sum_{k=0}^{d-1} \left(\sum_{j=0}^{k} |a_{k-j} b_j| + \sum_{j=k+1}^{d-1} |a_{k+d-j} b_j| \right)^2 \\
&= \sum_{k=0}^{d-1} \left(\sum_{j=0}^{d-1} |a^*_{k,j} b_j| \right)^2 \\
&= \sum_{k=0}^{d-1} (\mathbf{a}^*_k \cdot \mathbf{b}')^2 \\
&\leq \sum_{k=0}^{d-1} \left(\|\mathbf{a}^*_k\|^2 \|\mathbf{b}'\|^2 \right) \text{ (by Cauchy-Schwarz)} \\
&= \sum_{k=0}^{d-1} \left(\|\mathbf{a}'\|^2 \|\mathbf{b}'\|^2 \right) \\
&= d\|a\|^2 \|b\|^2.
\end{aligned}$$

Thus, $\sqrt{d} \geq \|a \cdot b\|/(\|a\|\|b\|)$ for all $a, b \in R$, so $\max\{\|a \cdot b\|/(\|a\|\|b\|) : a, b \in R\} = \gamma_R \leq \sqrt{d}$. □

For $\mathbf{c} \in R^n_q$, we define the linear function $L_{\mathbf{c}}(\mathbf{x})$ over the coefficients of $\mathbf{x}$ to be $L_{\mathbf{c}}(\mathbf{x}) = \mathbf{c} \cdot \mathbf{x}$. We also define $L_{\mathbf{c}_1,\mathbf{c}_2}(\Xi(\mathbf{x}))$ to be the linear function over the coefficients of $\Xi(\mathbf{x})$ to be $L_{\mathbf{c}_1,\mathbf{c}_2}(\Xi(\mathbf{x})) = L_{\mathbf{c}_1}(\mathbf{x}) \cdot L_{\mathbf{c}_2}(\mathbf{x})$. We treat the entries of $\mathbf{x}$ as variables, so $L_{\mathbf{c}_1}(\mathbf{x}) \cdot L_{\mathbf{c}_2}(\mathbf{x})$ is a linear expression in the elements of $\Xi(\mathbf{x})$, and this is therefore well defined.

For a set of hash functions $\mathcal{H}$ from X to Y, where both X and Y are finite sets, we say that $\mathcal{H}$ is *2-universal* if for all $x, x' \in X$ where $x \neq x'$, $\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(x')] = 1/|Y|$.

For two distributions D_1 and D_2 over a finite domain X, we say that the *statistical difference* between D_1 and D_2 is $\frac{1}{2} \sum_{x \in X} |D_1(x) - D_2(x)|$. We say that a distribution is ϵ-*uniform* if its statistical difference from the uniform distribution is at most ϵ.

We say that a function $f(x)$ is $\tilde{O}(x)$ if $f(x)$ is $O(x \cdot (\log(x))^k)$ for some integer k.

We say that an integer is α-*rough* if it has no prime factors smaller than α.

2.2 Lattices

Let $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ be a set of linearly independent vectors in $\mathbb{R}^n$. The *lattice* L generated by this set of vectors is the set of all integer combinations of this set of vectors. That is, $L = \{a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k \mid a_i \in \mathbb{Z} \text{ for all } 1 \leq i \leq k\}$.

We call any linearly independent set of vectors that generates L a *basis* for L, and there are an infinite number of such sets [7]. Any two bases for L have the same number of elements. The *dimension* of L (or the *rank* of L) is the number of vectors in a basis for L. Lattice $L \subset \mathbb{R}^n$ has full rank if its dimension is n.

Let $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ and $W = \{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$ be two bases for a lattice $L \subset \mathbb{R}^n$. Since we can generate L with integer combinations of V, we can write $\mathbf{w}_i = u_{i,1}\mathbf{v}_1 + \cdots + u_{i,k}\mathbf{v}_k$ for each $\mathbf{w}_i$, where each $u_{i,j} \in \mathbb{Z}$.

We can construct the matrix

$$U = \begin{pmatrix} u_{1,1} & \cdots & u_{1,k} \\ \vdots & \ddots & \vdots \\ u_{k,1} & \cdots & u_{k,k} \end{pmatrix} \text{ such that } U \cdot \begin{pmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_k \end{pmatrix} = \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_k \end{pmatrix}.$$

where the $\mathbf{v}_i$'s and $\mathbf{w}_i$'s are written as row vectors. Since W is also a basis for L, we can write each $\mathbf{v}_i$ as an integer combination of the vectors in W. Thus, the matrix U^{-1} is also an integer matrix. Since both U and U^{-1} are integer, we know that $\det(U), \det(U^{-1}) \in \mathbb{Z}$. But since $1 = \det(I) = \det(UU^{-1}) = \det(U)\det(U^{-1})$, this gives us that $\det(U) = \det(U^{-1}) = \pm 1$, so U is a unimodular matrix.

Let L be a lattice with basis $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$. Then the *fundamental domain* (also called the *fundamental parallelopiped*) for L for this basis is

$$\mathcal{F}(\mathbf{v}_1, \ldots, \mathbf{v}_k) = \{t_1\mathbf{v}_1 + \cdots + t_k\mathbf{v}_k \mid 0 \leq t_i < 1\}.$$

We call the k-dimensional volume of $\mathcal{F}$ the determinant of L, denoted by $\det(L)$. It is also called the covolume of L, as this is the volume of the quotient group $\mathbb{R}^n/L$. If $L \subset \mathbb{R}^n$ is of dimension n, then we can write the basis vectors as rows of a square matrix F. Concretely,

if $\mathbf{v}_i = (f_{i,1}, \ldots, f_{i,n})$, then

$$F = \begin{pmatrix} f_{1,1} & \cdots & f_{1,n} \\ \vdots & \ddots & \vdots \\ f_{n,1} & \cdots & f_{n,n} \end{pmatrix}.$$

Then the volume of the fundamental domain $\mathcal{F}$ is $\text{Vol}(\mathcal{F}) = |\det(F)|$. But since we can multiply F by a unimodular matrix to get a matrix representing another basis for L, the determinant of this new matrix is either $\det(F)$, or $-\det(F)$. Thus, the value of $\det(L)$ does not depend on the basis chosen.

If L is a lattice of rank k, for $0 \leq i \leq k$ we define its *i'th successive minimum* to be $\lambda_i(L) = \min\{\max\{\|\mathbf{x}_1\|, \ldots, \|\mathbf{x}_i\|\} \mid \mathbf{x}_1, \ldots, \mathbf{x}_i \in L \text{ are linearly independent}\}$. Then, if L has dimension n, for any $1 \leq r \leq n$ we have [7]:

$$\left(\prod_{i=1}^{r} \lambda_i(L)\right)^{\frac{1}{r}} \leq \sqrt{h_n}\det(L)^{\frac{1}{n}}. \tag{2.1}$$

The constant h_n is called *Hermite's constant* of dimension n, and the values of h_n are only known for $1 \leq n \leq 8$ and $n = 24$. However, when n is large, Hermite's constant satisfies $\frac{n}{2\pi e} \leq h_n \leq \frac{n}{\pi e}$ [14].

2.2.1 Orthogonality

We want to deal with bases whose elements are orthogonal as possible when trying to solve lattice based hard problems, as we will discuss later. Therefore, we need to have a way to determine the quality of a basis. Given a highly non-orthogonal lattice, we also want to be able to find a basis that is more orthogonal.

Let $L \subset \mathbb{R}^n$ be a lattice with dimension n. We define the *Hadamard Ratio* of the basis $B = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ to be

$$\mathcal{H}(B) = \left(\frac{\det(L)}{\prod_{i=1}^{n}\|\mathbf{v}_i\|}\right)^{1/n}.$$

The Hadamard Ratio satisfies $0 < \mathcal{H}(B) \leq 1$, where the basis B has elements that are more orthogonal to each other when $\mathcal{H}(B)$ is closer to 1. We call the reciprocal of $\mathcal{H}(B)$

the *orthogonality defect* of a basis. Notice that $\det(L)$ is fixed for all bases of a lattice L, so when the basis has shorter elements, the basis vectors are also pairwise more orthogonal.

Say we order our basis B so that $\|\mathbf{v}_1\| \leq \|\mathbf{v}_2\| \leq \cdots \leq \|\mathbf{v}_n\|$. Then we call $\|\mathbf{v}_1\|/\lambda_1(L)$ the *approximation factor*. However, given a lattice L, we do not know the value of $\lambda_1(L)$, and computing it is difficult, so we can replace $\lambda_1(L)$ with $\det(L)^{1/n}$ [15]. This is due to the relation between these values shown in (2.1). This value $\|\mathbf{v}_1\|/\det(L)^{1/n}$ is called the *Hermite factor*, and we call $(\|\mathbf{v}_1\|/\det(L)^{1/n})^{1/n}$ the *root-Hermite factor*.

2.2.2 Other Types of Lattices

Since a lattice $L \subset \mathbb{R}^n$ with basis $B = \{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ is defined as the set of all integer combinations of B, we can write L as $L = \{A\mathbf{x} \mid \mathbf{x} \in \mathbb{Z}^k\}$ where $A \in \mathbb{R}^{n \times m}$ is the matrix where column A_i is $\mathbf{v}_i$. We can define the *dual lattice* of L to be $L^\perp = \{\mathbf{x} \in \mathbb{R}^n \mid \forall \mathbf{v} \in B, \mathbf{v} \cdot \mathbf{x} \in \mathbb{Z}\}$. We say that L is an *integer lattice* if $L \subseteq \mathbb{Z}^n$.

A *modular lattice* with modulus q (also known as a *q-ary lattice*) is an integer lattice that has the form $L = \{A\mathbf{x} \bmod q \mid \mathbf{x} \in \mathbb{Z}^k\}$, where $A \in \mathbb{Z}_q^{n \times k}$ is the matrix whose columns are the basis vectors of L. These types of lattices also have a corresponding dual lattice of the form $L^\perp = \{\mathbf{x} \in \mathbb{Z}^n \mid \forall \mathbf{v} \in L, \mathbf{v} \cdot \mathbf{x} = 0 \bmod q\}$.

We can also create lattices based on ideals. If we are given a ring $R = \mathbb{Z}[x]/(f(x))$, for some polynomial $f(x)$ of degree d (in practice we will use $f(x) = x^d + 1$), then R is a polynomial ring where every element has degree less than d. We can think of R as $\mathbb{Z}^d$, where each element of R is represented by its coefficient vector. Let I be an ideal of R, which is a non-empty set of points in R that is closed under addition and multiplication by any element in R. Since a subset of $\mathbb{R}^n$ is a lattice if and only if it is a discrete additive subgroup [14], the elements of I can be used as the points in our lattice. Such a lattice is called an *ideal lattice* due to its dual nature of being both a lattice and an ideal. If we take I to be a principal ideal generated by $r \in R$, then the ideal lattice corresponding to I is generated by the vectors $\{r \cdot x^i \ : \ 0 \leq i < d\}$, where multiplication is done over R. Let $K = \mathbb{Q}[x]/(f(x))$ be a field containing I. Then the inverse of $I \subseteq R$ is given by $I^{-1} = \{w \in K \mid \forall v \in I, w \cdot v \in R\}$.

2.3 Underlying Hard Problems

This section defines a number of the underlying hard problems that are used in fully homomorphic encryption schemes and other lattice based encryption schemes. Since many

of the fully homomorphic encryption schemes are based on lattices, the problems in these schemes can be reduced to some of the more well studied lattice based problems. Although many of these problems are not referred to explicitly in any of the schemes talked about in this thesis, we present them give an overview of the relation between the problems we make use of and other common problems. We first define the Subset-Sum Problem, and then will define problems related to lattices. Most of the lattice problem definitions are taken from Laarhoven, van de Pol, and de Weger [15].

Subset-Sum Problem

If we are given a set of integers $S = \{x_1, \ldots, x_n\}$, the Subset-Sum Problem is to find a subset T of S whose elements sum to a given integer k, given that there exists at least one subset that satisfies this [14]. The Sparse Subset-Sum Problem (SSSP) is the same as the Subset-Sum Problem, except that a very small set T can be used to satisfy the sum.

2.3.1 Problems Related to Finding Short Vectors

This section reviews a few common hard problems that are used as the foundation of a number of lattice based cryptographic schemes. These problems deal with finding short vectors in a lattice.

First recall that the first successive minimum of a lattice L is $\lambda_1(L) = \min\{\|\mathbf{x}\| \mid \mathbf{x} \in L, \mathbf{x} \neq 0\}$, the smallest distance between two lattice points. Throughout, let L be a lattice of full rank in $\mathbb{Z}^n$.

Shortest Vector Problem (SVP)

The Shortest Vector Problem is to find a lattice point $\mathbf{y} \in L$ such that $\|\mathbf{y}\| = \lambda_1(L)$ given a basis in L.

The Shortest Vector Problem is known to be NP-Hard if we extend the class of polynomial-time algorithms to include non-derterministic, probabilistic algorithms that terminate, with high probability, in polynomial time with the correct result [14].

Approximate Shortest Vector Problem ($\mathbf{SVP}_\gamma$)

The Approximate Shortest Vector Problem is to find a lattice point $\mathbf{y} \in L$ such that $0 < \|\mathbf{y}\| \leq \gamma\lambda_1(L)$ given a basis in L and an approximation factor $\gamma \geq 1$.

For the previous two problems, the value of $\lambda_1(L)$ is not initially known. The next problem is similar to the Approximate Shortest Vector Problem, but uses a known value instead of the length of the shortest vector in the lattice.

Hermite Shortest Vector Problem ($\mathbf{HSVP}_\gamma$)

The Hermite Shortest Vector Problem is to find a lattice point $\mathbf{y} \in L$ such that $0 < \|\mathbf{y}\| \leq \gamma\text{vol}(L)^{1/n}$ given a basis in L and an approximation factor $\gamma > 0$.

Decision Shortest Vector Problem (DSVP)

The Decision Shortest Vector Problem is to determine whether a lattice point $\mathbf{y} \in L$ exists such that $0 < \|\mathbf{y}\| \leq r$ given a basis in L and a radius $r > 0$.

Approximate Shortest Length Problem ($\mathbf{SLP}_\gamma$)

The Approximate Shortest Length Problem is to find a λ such that $\lambda_1(L) \leq \lambda \leq \gamma\lambda_1(L)$ given a basis in L and an approximation factor $\gamma > 1$.

Unique Shortest Vector Problem ($\mathbf{USVP}_\gamma$)

The Unique Shortest Vector Problem is to find a lattice point $\mathbf{y} \in L$ such that any $\mathbf{v} \in L$ with $\|\mathbf{v}\| \leq \gamma\|\mathbf{y}\|$ is an integral multiple of $\mathbf{y}$, given a basis in L and a gap factor $\gamma \geq 1$.

Gap Shortest Vector Problem ($\mathbf{GapSVP}_\gamma$)

The Gap Shortest Vector Problem is if we are given a basis in L, an approximation factor $\gamma > 1$, and a radius $r > 0$, to return YES if $\lambda_1(L) \leq r$, NO if $\lambda_1(L) > \gamma r$, and either YES or NO otherwise.

2.3.2 Problems Related to Finding Closest Vectors

These problems deal with finding close vectors in a lattice to a given point. Let $d(\mathbf{t}, L)$ be the distance from $\mathbf{t} \in \mathbb{R}^n$ to the closest lattice point in L. Again, let L be a lattice of full rank in $\mathbb{Z}^n$.

Closest Vector Problem (CVP)

The Closest Vector Problem is to determine a lattice point $\mathbf{y} \in L$ such that $\|\mathbf{y} - \mathbf{t}\| = d(\mathbf{t}, L)$, given a basis in L and a target vector $\mathbf{t} \in \mathbb{R}^n$.

The Closest Vector Problem is known to be NP-hard. We can reduce the SVP to the CVP problem, so if we can solve the CVP, then we can also solve the SVP [15], [14].

Approximate Closest Vector Problem (CVP_γ)

The Approximate Closest Vector Problem is to determine a lattice point $\mathbf{y} \in L$ such that $\|\mathbf{y} - \mathbf{t}\| \leq \gamma d(\mathbf{t}, L)$, given a basis in L, a target vector $\mathbf{t} \in \mathbb{R}^n$, and an approximation factor $\gamma \geq 1$.

Decision Closest Vector Problem (DCVP)

The Decision Closest Vector Problem is to determine whether there exists a lattice point $\mathbf{y} \in L$ such that $\|\mathbf{y} - \mathbf{t}\| \leq r$, given a basis in L, a target vector $\mathbf{t} \in \mathbb{R}^n$, and a radius $r > 0$.

Bounded Distance Decoding Problem (BDD_α)

The Bounded Distance Decoding Problem is to determine a lattice point $\mathbf{y} \in L$ such that $\|\mathbf{y} - \mathbf{t}\| = d(\mathbf{t}, L)$, given a basis in L, a distance parameter $\alpha > 0$, and a target vector $\mathbf{t} \in \mathbb{R}^n$ such that $d(\mathbf{t}, L) < \alpha \lambda_1(L)$.

2.3.3 Problems Dealing with Sets of Short Vectors

These problems deal with finding sets of vectors in a lattice that are as small as possible. Recall that the i'th successive minimum $\lambda_i(L)$ is defined as $\lambda_i(L) = \min\{\max\{\|\mathbf{x}_1\|, \ldots, \|\mathbf{x}_i\|\} \mid \mathbf{x}_1, \ldots, \mathbf{x}_i \in L \text{ where } \mathbf{x}_1, \ldots, \mathbf{x}_i \text{ are linearly independent}\}$. Again, let L be a lattice of full rank in $\mathbb{Z}^n$.

Successive Minima Problem (SMP)

Given a basis of L, the Successive Minima Problem is to find a linearly independent set of lattice points $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\} \in L$ such that $\|\mathbf{y}_i\| = \lambda_i(L)$ for $i = 1, \ldots, n$.

Approximate Shortest Basis Problem ($\mathbf{SBP}_\gamma$)

The Approximate Shortest Basis Problem is to find a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ of L such that $\max_i\|\mathbf{b}_i\| \leq \gamma \cdot \min\{\max_i\|\mathbf{a}_i\| \mid \{\mathbf{a}_i, \ldots, \mathbf{a}_n\} \text{ is a basis for } L\}$, given a basis for L and an approximation factor $\gamma \geq 1$.

Shortest Independent Vector Problem ($\mathbf{SIVP}_\gamma$)

Given a basis of L and an approximation factor $\gamma > 1$, the Shortest Independent Vector Problem is to find a linearly independent set of lattice points $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\} \in L$ such that $\max_i\|\mathbf{y}_i\| \leq \gamma\lambda_n(L)$.

2.3.4 Learning with Errors (LWE) and Modular Lattice Problems

Before we define the Learning with Errors Problem, we will first define another modular lattice problem.

Small Integer Solutions Problem ($\mathbf{SIS}_{n,m,q,v}$)

If we are given a modulus q, a matrix $A \in \mathbb{Z}_q^{n\times m}$ and a $v \in \mathbb{R}$ such that $1 \leq v < q$, the Small Integer Solutions Problem is to find a nonzero $\mathbf{y} \in \mathbb{Z}^m$ such that $\|\mathbf{y}\| \leq v$ and $A\mathbf{y} = 0 \pmod q$.

We will define the Learning with Errors Problem in two different formats below. For the next problems, let $n \geq 1, m \geq 1,$ and $q \geq 2$ be integers, and let $f(x) = x^d + 1$, where d is a power of 2. Let $R = \mathbb{Z}[x]/(f(x))$, and let χ be some distribution over R. We will always choose χ to be a discrete Gaussian distribution with standard deviation $\sigma = \alpha q/\sqrt{2\pi}$ for some parameter α, although these problems can be posed with any general distribution χ.

The Search General Learning with Errors (search-GLWE$_{n,m,q,\alpha}$) Problem

The Search General Learning with Errors (search-GLWE) Problem is to determine $\mathbf{s} \in R_q^n$ given the set consisting of the pairs $(\mathbf{a}_i, b_i) \in R_q^n \times R_q$ where $\mathbf{a}_i$ is sampled uniformly from R_q^n, e_i is sampled from χ, and $b_i = \mathbf{a}_i \cdot \mathbf{s} + e_i$ for $i = 1, \ldots, m$. The search-GLWE Assumption is that the search-GLWE Problem is computationally infeasible.

We can also define the Search Learning with Errors (search-LWE) Problem and Assumption to be the search-GLWE Problem and Assumption where $d = 1$, and the Search Ring Learning with Errors (search-RLWE) Problem and Assumption to be the search-GLWE Problem and Assumption where $n = 1$.

The Search Learning with Errors (search-LWE$_{n,m,q,\alpha}$) Problem

The Search Learning with Errors (search-LWE) Problem is to determine $\mathbf{s} \in \mathbb{Z}_q^n$ given the set consisting of the pairs $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ where $\mathbf{a}_i$ is sampled uniformly from $\mathbb{Z}_q^n$, e_i is sampled from χ, and $b_i = \mathbf{a}_i \cdot \mathbf{s} + e_i$ for $i = 1, \ldots, m$.

The Search Ring Learning with Errors (search-RLWE$_{m,q,\alpha}$) Problem

The Search Ring Learning with Errors (search-RLWE) Problem is to determine $s \in R_q$ given the set consisting of the pairs $(a_i, b_i) \in R_q \times R_q$ where a_i is sampled uniformly from R_q, e_i is sampled from χ, and $b_i = a_i \cdot s + e_i$ for $i = 1, \ldots, m$.

The Decision General Learning with Errors (decision-GLWE$_{n,m,q,\alpha}$) Problem

The Decision General Learning with Errors (decision-GLWE) Problem is to distinguish the following two sets. The first set consists of the pairs $(\mathbf{a}_i, b_i)$ sampled uniformly from $R_q^n \times R_q$ for $i = 1, \ldots, m$. Before constructing the second set, we must first draw $\mathbf{s}$ uniformly from R_q^n. Then the second set consists of the pairs $(\mathbf{a}_i, b_i) \in R_q^n \times R_q$ where $\mathbf{a}_i$ is sampled uniformly from R_q^n, e_i is sampled from χ, and $b_i = \mathbf{a}_i \cdot \mathbf{s} + e_i$ for $i = 1, \ldots, m$. The decision-GLWE Assumption is that the decision-GLWE Problem is computationally infeasible.

We can also define the Decision Learning with Errors (decision-LWE) Problem and Assumption to be the decision-GLWE Problem and Assumption where $d = 1$, and the Decision Ring Learning with Errors (decision-RLWE) Problem and Assumption to be the decision-GLWE Problem and Assumption where $n = 1$.

The Decision Learning with Errors (decision-LWE$_{n,m,q,\alpha}$) Problem

The Decision Learning with Errors (decision-LWE) Problem is to distinguish the following two sets. The first set consists of the pairs $(\mathbf{a}_i, b_i)$ sampled uniformly from $\mathbb{Z}_q^n \times \mathbb{Z}_q$ for $i = 1, \ldots, m$. Before constructing the second set, we must first draw $\mathbf{s}$ uniformly from $\mathbb{Z}_q^n$. Then the second set consists of the pairs $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ where $\mathbf{a}_i$ is sampled uniformly from $\mathbb{Z}_q^n$, e_i is sampled from χ, and $b_i = \mathbf{a}_i \cdot \mathbf{s} + e_i$ for $i = 1, \ldots, m$.

The Decision Ring Learning with Errors (decision-RLWE$_{m,q,\alpha}$) Problem

The Decision Ring Learning with Errors (decision-RLWE) Problem is to distinguish the following two sets. The first set consists of the pairs (a_i, b_i) sampled uniformly from $R_q \times R_q$ for $i = 1, \ldots, m$. Before constructing the second set, we must first draw s uniformly from R_q. Then the second set consists of the pairs $(a_i, b_i) \in R_q \times R_q$ where a_i is sampled uniformly from R_q, e_i is sampled from χ, and $b_i = \mathbf{a}_i \cdot \mathbf{s} + e_i$ for $i = 1, \ldots, m$.

Comments about LWE

The search-LWE Problem is at least as hard as the decision-LWE Problem [20], and in fact, the problems are equivalent after a polynomial time reduction when q is prime and polynomial in size to n [18], [17]. It also holds that the search-RLWE and decision-RLWE are equivalent via a probabilistic polynomial time reduction when q is prime and polynomial in size to n [17]. Throughout this thesis, we will be dealing with either the LWE Problem or the RLWE Problem (and not the general case), as the cases where neither the ring dimension d nor the vector length n equals 1 have not been explored [2].

2.3.5 Solving Lattice Based Hard Problems

If someone is able to break the SIS$_{n,m,q,1.5\sqrt{2\pi}/\alpha}$ Problem, then they can break decision-LWE$_{n,m,q,\alpha}$ [20]. Thus, when looking for a way to solve the LWE Problem, we can focus on solving the SIS Problem. However, each of SBP$_\gamma$, SIVP$_\gamma$, HSVP$_\gamma$, USVP$_\gamma$, GapSVP$_\gamma$, SIS$_{n,m,q,v}$, BDD$_{1/\gamma}$, and Search-LWE$_{n,m,q,\alpha}$ can be reduced to SVP$_\gamma$ [15]. Since we can reduce SVP$_\gamma$ to CVP$_\gamma$, if we can solve CVP$_\gamma$, we can solve all of these hard lattice problems.

If we have a lattice $L \subset \mathbb{R}^n$ of full rank, with a basis $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$, then Babai's Closest Vertex Algorithm solves CVP if the basis vectors are sufficiently orthogonal [14]. Babai's algorithm is stated next.

Babai's Closest Vertex Algorithm

Let $\mathbf{w} \in \mathbb{R}^n$ *be the target vector that we are trying to find the nearest lattice point to. Since we have* n *basis vectors, we can write* $\mathbf{w} = t_1\mathbf{v}_1 + \cdots + t_n\mathbf{v}_n$ *where* $t_i \in \mathbb{R}$ *for all* $i \in \{1, \ldots, n\}$*. Let* $a_i = \lfloor t_i \rceil$ *for all* $i \in \{1, \ldots, n\}$*. Then return the lattice point* $\mathbf{v} = a_1\mathbf{v}_1 + \cdots + a_n\mathbf{v}_n$*.*

If the basis vectors are somewhat orthogonal to each other, this algorithm solves CVP_γ for some γ, depending on how orthogonal the basis is. However, if the basis is very nonorthogonal, then the outputted lattice point is typically far away from the actual closest lattice point [14].

To solve lattice based problems, we therefore need to be able to determine a basis that is orthogonal enough given any basis for the lattice we are dealing with. We call such a basis *reduced*. A common algorithm used to get a reduced basis is the LLL algorithm by Lenstra, Lenstra and Lovász. The LLL algorithm is a polynomial time algorithm that returns a basis (said to be LLL-reduced) with a Hermite factor less than $(4/3)^{(n-1)/4}$ and an approximation factor less than $(4/3)^{(n-1)/2}$, where n is the lattice dimension [7]. If we apply the LLL algorithm to some basis, and then use Babai's algorithm, this solves CVP_γ to within a factor of C^n for some C [14].

Another algorithm to find a fairly orthogonal basis is a variant on the LLL method, called the *deep insertion method* (DEEP). This algorithm may improve the Hermite factor and approximation factor (and does so in practice quite quickly for most lattices [14]), but may in the worst case have a superexponential complexity [7]. This algorithm uses a block size parameter, which we can increase to improve the basis, but doing this also increases the runtime.

A third variant of the LLL algorithm is the *block Korkin-Zolotarev* (BKZ) algorithm. This algorithm uses the idea of a KZ-reduced basis. A KZ-reduced basis is typically better than an LLL-reduced basis, and in fact has its shortest vector as the solution to the SVP [14]. As such, any method to solve for a KZ-reduced basis runs in exponential time with respect to the lattice dimension.

The distinguishing feature of the BKZ algorithm is that it takes a block of basis vectors that generate the lattice and replaces them with a KZ-reduced lattice that is a basis for the same sublattice[14]. As in DEEP, increasing the block size increases the runtime, but returns a better reduced basis. Taking the block size to be the size of the lattice dimension solves the SVP exactly, but is exponential in the dimension. If BKZ terminates, then it

returns a basis with Hermite factor less than $\sqrt{h_\beta}^{1+(n-1)/(\beta-1)}$ and an approximation factor less than $h_\beta^{(n-1)/(\beta-1)} \leq \left(\frac{\beta}{\pi e}\right)^{(n-1)/(\beta-1)}$ where β is the blocksize [7].

2.3.6 The Approximate GCD Problem

We define the Approximate GCD Problem, and some of its variations. These problems are used for the fully homomorphic encryption schemes over the integers discussed in Chapter 5.

The Approximate GCD Problem

Let $D_{\gamma,\rho}(p) = \{$ choose $q \in \mathbb{Z}$ from $[0, 2^\gamma/p)$, and $r \in \mathbb{Z}$ from $(-2^\rho, 2^\rho)$, and output $x = pq + r\}$ be a probability distribution for some variables $\gamma \in \mathbb{Z}, \rho \in \mathbb{Z}$, and $p \in \mathbb{Z}$ where p is an odd number that has bit length $\eta \in \mathbb{Z}$. The (ρ, η, γ) Approximate GCD Problem is to determine p given a polynomial number of samples from $D_{\gamma,\rho}(p)$. The Approximate GCD assumption, is that the Approximate GCD Problem is computationally infeasible [23].

The Decisional Approximate GCD Problem

Let $D_{\gamma,\rho}(p)$ be the same distribution as defined in the Approximate GCD Problem. Say we are given polynomially many samples from $D_{\gamma,\rho}(p)$. Then, if we are given a random integer $j \in [0, \eta]$ and an integer $z = x + b \cdot \lfloor 2^j \cdot p/2^{\eta+1} \rceil$, where x is sampled from $D_{\gamma,\rho}(p)$, the (ρ, η, γ)-Decisional Approximate GCD Problem is to determine the value of b, where $b \in \{0, 1\}$. The Decisional Approximate GCD assumption, is that the Decisional Approximate GCD Problem is computationally infeasible [6].

The Error-free Approximate GCD Problem

Let $D'_\rho(p, q_0) = \{$ choose $q \in \mathbb{Z}$ from $[0, q_0)$, and $r \in \mathbb{Z}$ from $(-2^\rho, 2^\rho)$, and output $x = pq + r\}$ be a probability distribution for some variables $\rho \in \mathbb{Z}$, and $q_0 \in \mathbb{Z}$. The (ρ, η, γ)-Error-free Approximate GCD Problem is: For a random prime integer p whose bit-length is η, if we are given $x_0 = q_0 \cdot p$ where q_0 is a random square-free 2^λ-rough integer in $[0, 2^\gamma/p)$, and polynomially many samples from $D'_\rho(p, q_0)$, we need to determine p. The Error-free Approximate GCD assumption, is that the Error-free Approximate GCD Problem is computationally infeasible [5].

Chapter 3

Fully Homomorphic Encryption

This chapter gives an overview of what it means for an encryption scheme to be fully homomorphic.

We will use the definitions related to homomorphic encryption from van Dijk, Gentry, Halevi, and Vaikuntanathan [23], and originally stated by Gentry [9], [8]. For all these definitions, we will deal with boolean circuits that have gates that compute addition and multiplication mod 2. More explicitly, this means that a circuit C takes in a number of inputs $(m_1, m_2, ..., m_t) \in \mathbb{Z}_2^t$, and then performs a number of addition and multiplication operations on these inputs, and outputs the resulting number in $\mathbb{Z}_2$. We will denote the security parameter by λ.

3.1 Basic Properties

For a homomorphic public key cryptosystem $\mathcal{E}$, there are four primary functions. As in other public key cryptosystems, there is a key generation function, an encryption function and a decryption function. However, in homomorphic encryption systems, there is another function *Evaluate* that takes a public key, a circuit, and a set of ciphertexts $\mathbf{c} = (c_1, c_2, \ldots, c_t)$, where t is the number of inputs for the circuit, and outputs the circuit applied to the set of ciphertexts.

Definition 3.1.1. (Correct Homomorphic Decryption). We say that an encryption scheme $\mathcal{E}$ = (KeyGen, Encrypt, Decrypt, Evaluate) is *correct*, for some given circuit C with t inputs, if for any pair of secret and public keys (sk, pk) that are output by KeyGen(λ), and

any t plaintext bits $m_1, \ldots, m_t$ and ciphertexts $\mathbf{c} = (c_1, \ldots, c_t)$ such that $c_i = \text{Encrypt}(pk, m_i)$, we get that Decrypt($sk$, Evaluate($pk$, C, $\mathbf{c}$)) $= C(m_1, \ldots, m_t)$.

In other words, a correct homomorphic encryption scheme for a circuit C allows us to perform the addition and multiplication operations of C on a set of encrypted data, and not have to decrypt first in order to get a legitimate ciphertext. This ability would allow someone to perform the operation described by C on a set of data without knowing what the data is.

Definition 3.1.2. (Homomorphic Encryption). We say that an encryption scheme $\mathcal{E}$ = (KeyGen, Encrypt, Decrypt, Evaluate) is *homomorphic* for a class of circuits $\mathcal{C}$ if it is correct for all circuits $C \in \mathcal{C}$. We say that $\mathcal{E}$ is *fully homomorphic* if it is correct for all boolean circuits.

We can create a simple encryption scheme that is fully homomorphic very easily based on any encryption scheme. If the ciphertext that is output by the Evaluate function contains all of the original ciphertexts, and a complete description of the circuit that is being evaluated, we can make the Decrypt function simply decrypt the original ciphertexts first and then pass these plaintext bits through the circuit. However, we want our encryption scheme to not reveal any information about the circuit that is being evaluated, so this solution becomes impractical.

We say that an encryption scheme maintains *circuit-privacy* if no information about the circuit is revealed by the Evaluate function except for the output value of the circuit given a set of input ciphertexts. This property must hold even if the secret key is known. To achieve this, it is sufficient for a fully homomorphic scheme to have the compactness property, that is, the size of the ciphertext output by Evaluate does not depend on the input circuit.

Definition 3.1.3. (Compact Homomorphic Encryption). We say that a homomorphic encryption scheme $\mathcal{E}$ = (KeyGen, Encrypt, Decrypt, Evaluate) is *compact* if there is a fixed polynomial bound $b(\lambda)$ such that for any secret key and private key pair (sk, pk) output by KeyGen(λ), any circuit C and any set of input ciphertexts $\mathbf{c} = (c_1, c_2, \ldots, c_t)$ generated using the public key pk, the bit length of Evaluate$(pk, C, \mathbf{c})$ is less than or equal to $b(\lambda)$.

The difficulty with creating a homomorphic encryption scheme that is correct is that when we encrypt data, we have to scramble the original data in some way. To do this we must apply some noise to the original data. Our decryption scheme can handle the

noise that is introduced when we encrypt the data, but as we add and multiply ciphertexts together, the noise terms are also added and multiplied. As such, the resulting ciphertexts have larger noise than the ciphertexts created by directly encrypting a given plaintext. Eventually, the noise will become too much for the decryption function to handle, and this can cause the decryption function to decrypt ciphertexts incorrectly.

3.2 Leveled Homomorphic Decryption

Even if we are unable to create a fully homomorphic scheme, we might be able to create something called a leveled fully homomorphic encryption scheme.

Definition 3.2.1. (Leveled Fully Homomorphic Encryption). We say that a set of encryption schemes $\{\mathcal{E}^{(d)} : d \in \mathbb{Z}\}$ is a set of *leveled fully homomorphic encryption schemes* if (i) they all use the same Decrypt function; (ii) $\mathcal{E}^{(d)}$ is homomorphic for all circuits of depth d or less; (iii) the computational complexity of the functions in $\mathcal{E}^{(d)}$ is polynomial in λ, and d; and (iv) Evaluate has polynomial computational complexity with respect to the evaluated circuit C.

A leveled fully homomorphic encryption scheme allows us to perform a certain number of operations on input ciphertexts without making the noise in the resulting ciphertext too large so that decryption fails. As long as we can create leveled fully homomorphic encryption schemes that are homomorphic for circuits of arbitrary depth, we can apply any function to encrypted data and still get a meaningful result. Thus, if we know what operations we want applied to data beforehand, we can encrypt the data with a leveled fully homomorphic encryption scheme that can handle large enough circuits, and then pass the data off to be operated on in encrypted form. The BGV encryption scheme [2] is such a scheme, and will be discussed further in Chapter 4.

3.3 Bootstrapping

Bootstrapping is a process that can be applied to a ciphertext to reduce the noise in the ciphertext. Since every addition and multiplication increases the noise in the ciphertext, we need a way to shrink the noise back to a size that is more manageable. We know that if we decrypt the ciphertext, and then re-encrypt it, the noise introduced is back to the original size, and as such, we can do this whenever the noise starts to get too large. However, we

want the entity that is evaluating the circuit to have this ability to shrink the noise. To do this, we need to be able to evaluate the decryption circuit homomorphically. To get anything useful though, we need to have an encryption scheme that can evaluate a little deeper than its own decryption scheme so that we can evaluate addition and multiplication operations after we decrypt the ciphertexts homomorphically.

Definition 3.3.1. (Augmented Decryption Circuits). Let $\mathcal{E}$ be an encryption scheme $\mathcal{E}$ = (KeyGen, Encrypt, Decrypt, Evaluate), where the running time of Decrypt depends only on the security parameter λ. For a given λ, let $D_{\mathcal{E}}(\lambda)$ be the set of *augmented decryption circuits*. This set consists of two circuits that take as input two secret keys and two corresponding ciphertexts. The first circuit decrypts the two ciphertexts, and then outputs the sum of the resulting plaintexts mod 2. The second circuit decrypts the two ciphertexts, and then outputs the product of the resulting plaintexts mod 2.

Definition 3.3.2. (Bootstrappable Encryption). Given a homomorphic encryption scheme $\mathcal{E}$ = (KeyGen, Encrypt, Decrypt, Evaluate) and a security parameter λ, let $C_{\mathcal{E}}(\lambda)$ be the set of circuits for which $\mathcal{E}$ is correct. $\mathcal{E}$ is said to be *bootstrappable* if $D_{\mathcal{E}}(\lambda) \subseteq C_{\mathcal{E}}(\lambda)$ for all λ.

Definition 3.3.2 says that if we can decrypt the ciphertexts homomorphically, and can then apply an addition or multiplication operation on the plaintexts, then the scheme is bootstrappable. If we have a bootstrappable scheme, we can compute any circuit homomorphically. All we would need to do is replace any addition gates in the circuit by the augmented addition circuit, and similarly replace all multiplication gates by the augmented multiplication circuit. Then, after every addition or multiplication, we would have a refreshed ciphertext and the noise would be shrunk back to its original level.

For instance, say we want to take two encrypted plaintexts m_1 and m_2 and apply some binary operation $\circledast$ to them (where $\circledast$ is either addition or multiplication). Let $c_1 = \text{Encrypt}(pk_1, m_1)$ with corresponding secret key sk_1 and $c_2 = \text{Encrypt}(pk_2, m_2)$ with corresponding secret key sk_2. Let $\overline{sk_1} = \text{Encrypt}(pk_3, sk_1)$, $\overline{c_1} = \text{Encrypt}(pk_3, c_1)$, $\overline{sk_2} = \text{Encrypt}(pk_3, sk_2)$, and $\overline{c_2} = \text{Encrypt}(pk_3, c_2)$ with corresponding secret key sk_3. Also, let $D_{\circledast}$ be the augmented decryption circuit that decrypts two ciphertexts and performs the operation $\circledast$ on them, and say the encryption scheme is correct for this circuit. Then if we

set c_3 to be Evaluate$(pk_3, D_\circledast, (\overline{sk_1}, \overline{c_1}, \overline{sk_2}, \overline{c_2}))$ we get

$$\begin{aligned} c_3 &= \text{Evaluate}(pk_3, D_\circledast, (\overline{sk_1}, \overline{c_1}, \overline{sk_2}, \overline{c_2})) \\ &= D_\circledast(\text{Encrypt}(pk_3, sk_1), \text{Encrypt}(pk_3, c_1), \text{Encrypt}(pk_3, sk_2), \text{Encrypt}(pk_3, sk_2)) \\ &= \text{Encrypt}(pk_3, D_\circledast(sk_1, c_1, sk_2, c_2)) \\ &= \text{Encrypt}(pk_3, \text{Decrypt}(sk_1, c_1) \circledast \text{Decrypt}(sk_2, c_2)) \\ &= \text{Encrypt}(pk_3, m_1 \circledast m_2). \end{aligned}$$

This gives us that $\text{Decrypt}(sk_3, c_3) = m_1 \circledast m_2$, which is what we wanted. We get that $D_\circledast(\text{Encrypt}(pk_3, sk_1)$,$\text{Encrypt}(pk_3, c_1)$,$\text{Encrypt}(pk_3, sk_2)$,$\text{Encrypt}(pk_3, sk_2)) = \text{Encrypt}(pk_3, D_\circledast(sk_1, c_1, sk_2, c_2))$ because the Evaluate function is able to homomorphically evaluate the circuit $D_\circledast$. Also, in this process we can see that the original ciphertexts were decrypted homomorphically, which removes the original noise applied to the plaintexts so that the noise doesn't become too high.

To perform these operations, we have to give the party evaluating the circuits some information, namely the $\overline{sk}$'s. Since each sk is encrypted with the same encryption function that is being used in the rest of the scheme, we can assume that this does not give away any information about the secret key, otherwise we would have the problem of a lack of semantic security with the encryption scheme even encrypting ordinary messages, as information would be able to be gathered about any encrypted message. However, using the same public key to encrypt multiple times in a row may not guarantee semantic security.

One problem that is encountered with trying to create a bootstrappable encryption scheme is that sometimes the decryption circuit is too large to evaluate it homomorphically. In this case, we can squash the decryption circuit. This means that we can preprocess the ciphertext, and partially decrypt it before trying to evaluate anything homomorphically. By adding some extra information into the ciphertext, we can decrease the number of steps needed for decryption, and hence get a decryption circuit small enough to evaluate homomorphically.

Also, if we have a bootstrappable scheme, we can create a leveled fully homomorphic encryption scheme as mentioned in the following theorem. However, the requirement of bootstrappability is not necessary to get a leveled fully homomorphic encryption scheme. The BGV Encryption scheme [2] for instance is a scheme that is leveled fully homomorphic, but does not require the scheme to be bootstrappable to get this result. This isn't to say however that this scheme is fully homomorphic, just that it doesn't require the bootstrappable property to be leveled fully homomorphic.

Theorem 3.3.1 ([23]). *Given an encryption scheme $\mathcal{E}$ that is bootstrappable, and a parameter d dependent on the security parameter, there is an efficient and explicit transformation that outputs a leveled fully homomorphic encryption scheme $\mathcal{E}^{(d)}$ that is compact and has the same Decrypt function as $\mathcal{E}$. Also, if $\mathcal{E}$ is semantically secure, then so is $\mathcal{E}^{(d)}$. It is also true that for any attack that has advantage ϵ against $\mathcal{E}^{(d)}$, this attack can be changed into an attack on $\mathcal{E}$ with similar complexity, that has advantage at least $\epsilon/(ld)$, where l is the length of the secret key in $\mathcal{E}$.*

Chapter 4

Lattice Based Fully Homomorphic Encryption

This chapter focuses on the fully homomorphic encryption scheme by Brakerski, Gentry, and Vaikuntanathan. It details the encryption scheme, provides concrete parameter choices to make basic decryption work, corrects some errors in the assumptions about the noise, and provides concrete security parameters based on experimental data from Lindner and Peikert.

4.1 The BGV Encryption Scheme

The BGV (Brakerski, Gentry, Vaikuntanathan) encryption scheme is a lattice based encryption scheme that relies on the assumption that the Learning with Errors Problem is computationally infeasible. This is a leveled fully homomorphic scheme, which means that it can evaluate all circuits homomorphically up to a predefined depth. It also doesn't require bootstrappability to achieve this property.

4.1.1 The Basic Encryption Scheme

The basic BGV encryption scheme describes the functions used in the encryption scheme that do not deal with any of the homomorphic properties. It outlines how to generate secret and public keys, and how to encrypt and decrypt.

Setup($\lambda \in \mathbb{Z}, \mu \in \mathbb{Z}, b \in \{0,1\}$):

The following encryption scheme is due to Brakerski, Gentry, and Vaikuntanathan in their paper "Fully Homomorphic Encryption without Bootstrapping" [2]. We use similar notation as that paper, with influences from Gentry, Halevi, and Smart[11].

To set up the basic encryption scheme, we first need to determine whether we are working with the Learning with Errors (LWE) based scheme, or the Ring Learning with Errors (RLWE) based scheme. Let $n+1$ be the dimension of the secret key. In the first case when $b = 0$, we set the $d = 1$, and as such define R to be $R = \mathbb{Z}[x]/(x^d + 1) = \mathbb{Z}$. In the second case when $b = 1$, we set $n = 1$. The cases where neither d nor n are 1 may not have been studied by anyone [2] and as such there are no statements about the hardness of these problems, so we will not cover these cases.

We choose our values for $n \in \mathbb{Z}$ and $d \in \mathbb{Z}$ depending on whether we are using the LWE or RLWE instance as above and so that the scheme achieves a 2^λ security level against current attacks, and then set $N = \lceil (2n+1)\log_2(q) \rceil$; here q is an odd integer with bit length μ. We also set χ to be some probability distribution over R_q. We typically use a discrete Gaussian-like distribution over R_q centred about 0 (e.g. each coefficient of r sampled from χ is sampled from a Guassian distribution over $\mathbb{Z}_q$ centred at 0). That is, we will assign a probability proportional to $e^{-\pi\|r\|/\sigma^2}$ to each $r \in R_q$, where σ^2 is the variance of the distribution χ.

For each of the following functions, we will use these parameters, which we will denote as $params = (q, d, n, N, \chi)$.

SecretKeyGen($q \in \mathbb{Z}, d \in \mathbb{Z}, n \in \mathbb{Z}, \chi$ *a distibution over* R_q):

Sample $\mathbf{s}' \in R_q^n$ from χ^n, and then set the secret key to be $sk = \mathbf{s} = (1, \mathbf{s}'[1], \ldots, \mathbf{s}'[n])^T \in R_q^{n+1}$.

PublicKeyGen($q \in \mathbb{Z}, d \in \mathbb{Z}, n \in \mathbb{Z}, N \in \mathbb{Z}, \chi$ *a distibution over* $R_q, sk = (1, \mathbf{s}') \in R_q^{n+1}$):

Generate A' uniformly from $R_q^{N \times n}$, and generate $\mathbf{e} \in R_q^N$ from χ^N, and let $\mathbf{b} = A'\mathbf{s}' + 2\mathbf{e} \in R_q^n$. Let the public key be $pk = A = \left[\ \mathbf{b}\ \middle|\ -A'\ \right] \in R_q^{N \times (n+1)}$.

Encrypt($q \in \mathbb{Z}, d \in \mathbb{Z}, n \in \mathbb{Z}, N \in \mathbb{Z}, pk = A \in R_q^{N\times(n+1)}, m \in R_2$)**:**

To encrypt a message m, set $\mathbf{m} = (m, 0, \ldots, 0)^T \in R_q^{n+1}$, sample $\mathbf{r}$ from R_2^N, and output the ciphertext $\mathbf{c} = \mathbf{m} + A^T\mathbf{r} \in R_q^{n+1}$.

Decrypt($sk = \mathbf{s} \in R_q^{n+1}, \mathbf{c} \in R_q^{n+1}$)**:**

Output the original message $m = [\mathbf{c} \cdot \mathbf{s}]_2 \in R_2$.

Correctness of the decryption procedure is established in Section 4.2.

4.1.2 Some Other Necessary Functions

The following functions will be used in the fully homomorphic version of the BGV encryption scheme.

BitDecomp($\mathbf{x} \in R_q^n, q \in \mathbb{Z}$)**:**

This function changes $\mathbf{x}$ into a binary representation of itself. Let q be an odd integer.

Write $\mathbf{x}$ as $\sum_{k=0}^{\lfloor \log_2 q \rfloor} 2^k \mathbf{u}_k$ where $\mathbf{u}_k \in R_2^n$, and output $\mathbf{u} = (\mathbf{u}_0^T, \mathbf{u}_1^T, \ldots, \mathbf{u}_{\lfloor \log_2 q \rfloor}^T)^T \in R_2^{n\lceil \log_2 q \rceil}$.

Powersof2($\mathbf{x} \in R_q^n, q \in \mathbb{Z}$)**:**

This function outputs a vector that contains a number of the inputted vectors multiplied by various powers of 2, and in some sense is the reverse of the BitDecomp function. In fact, $\text{BitDecomp}(\mathbf{c}, q) \cdot \text{Powersof2}(\mathbf{s}, q) = \mathbf{c} \cdot \mathbf{s} \pmod q$. Let q be an odd integer.

Output $\mathbf{u} = (\mathbf{x}^T, 2\mathbf{x}^T, \ldots, 2^{\lfloor \log_2 q \rfloor}\mathbf{x}^T)^T \in R_q^{n\lceil \log_2 q \rceil}$.

SwitchKeyGen($\mathbf{s}_1 \in R_q^{n_1}, \mathbf{s}_2 = (1, \mathbf{s}_2') \in R_q^{n_2}, q \in \mathbb{Z}, d \in \mathbb{Z}, n_1 \in \mathbb{Z}, n_2 \in \mathbb{Z}, \chi$ *a distibution over* R_q)**:**

This function creates a matrix that is used to change a ciphertext encrypted under $\mathbf{s}_1$ to be encrypted under $\mathbf{s}_2$.

Let $N = n_1\lceil \log_2(q) \rceil$, $\mathbf{u} = \text{Powersof2}(\mathbf{s}_1) \in R_q^N$, and set $A = \text{PublicKeyGen}(q, d, n_2, N, \chi, \mathbf{s}_2)$ $\in R_q^{N \times n_2}$. Output the key switching matrix $\tau_{\mathbf{s}_1,\mathbf{s}_2} = \left[\begin{array}{c|c|c|c} \mathbf{u} + A_1 & A_2 & \cdots & A_{n_2+1} \end{array} \right] \in R_q^{N \times n_2}$.

SwitchKey($\tau_{\mathbf{s}_1,\mathbf{s}_2} \in R_q^{n_1\lceil \log_2(q) \rceil \times n_2}, \mathbf{c}_1 \in R_q^{n_1}, q \in \mathbb{Z}$):

This function uses the key switching matrix $\tau_{\mathbf{s}_1,\mathbf{s}_2}$ to change the vector $\mathbf{c}_1$ from being encrypted under $\mathbf{s}_1$ to being encrypted under $\mathbf{s}_2$.

Since $\text{BitDecomp}(\mathbf{c}_1, q)$ is in $R_2^{n_1\lceil \log_2(q) \rceil}$, we can interpret this vector as a vector in $R_q^{n_1\lceil \log_2(q) \rceil}$ where all of the entries are polynomials with 0 or 1 coefficients. Thus, we can multiply this vector by $\tau_{\mathbf{s}_1,\mathbf{s}_2}$. Output the new ciphertext $\mathbf{c}_2 = (\text{BitDecomp}(\mathbf{c}_1, q)^T \cdot \tau_{\mathbf{s}_1,\mathbf{s}_2})^T \in R_q^{n_2}$ that is encrypted under $\mathbf{s}_2$.

Scale($\mathbf{c} \in R^n, q \in \mathbb{Z}, p \in \mathbb{Z}, r \in \mathbb{Z}$):

This function changes the moduli that a ciphertext $\mathbf{c}$ is being represented with from q to p.

For an R-vector $\mathbf{c}$, and integers $q > p > r$, we output $\mathbf{c}'$ such that $\mathbf{c}'$ is the closest R-vector to $(p/q)\mathbf{c}$ satisfying $\mathbf{c}' \equiv \mathbf{c} \pmod{r}$.

4.1.3 The Leveled Fully Homomorphic Encryption Scheme without Bootstrapping

This section describes the functions in the leveled fully homomorphic encryption scheme. As in the standard scheme, it details how to create the public and secret keys. However, since a different key is required for each level, the FHEKeyGen function computes a secret and public key for however many levels are specified. Encrypt and Decrypt are described here as before. There are also the functions FHEAdd and FHEMult which describe how to add and multiply encrypted messages, which is the added benefit of having a fully homomorphic encryption scheme.

FHESetup($\lambda \in \mathbb{Z}, L \in \mathbb{Z}, b \in \{0,1\}$):

To set up the basic encryption scheme, we first need to determine whether we are working with the Learning with Errors based scheme, or the Ring Learning with Errors based scheme. In the first case when $b = 0$, we set the $d = 1$, and as such define R to be $R = \mathbb{Z}[x]/(x^d + 1) = \mathbb{Z}$. In the second case when $b = 1$, we set $n = 1$.

For the fully homomorphic version of this encryption scheme, we will construct a number of levels L that we can encrypt with. For each level, this provides us with the ability to perform one addition or multiplication.

Let $\mu \in \mathbb{Z}$ be the length of the smallest moduli to achieve the necessary security level. For $j \in \{0, \ldots, L\}$, run $\text{Setup}(\lambda, (j+1)\mu, b)$, to get the parameters $params_j = (q_j, d_j, n_j, N_j, \chi_j)$. This creates a ladder of decreasing moduli from q_L that is $(L+1)\mu$ bits long to q_0 that is μ bits long. However, for all j, we can let $d_j = d$, and $\chi_j = \chi$, as these parameters do not depend on the circuit level.

FHEKeyGen:

For j from L to 0 do the following.

Set $\mathbf{s}_j = \text{SecretKeyGen}(q_j, d, n_j, N_j, \chi) \in R_{q_j}^{n_j+1}$, and $A_j = \text{PublicKeyGen}(q_j, d, n_j, N_j, \chi, \mathbf{s}_j) \in R_{q_j}^{N_j \times (n_j+1)}$. Let $\mathbf{s}'_j = \Xi(\mathbf{s}_j) \in R_{q_j}^{\binom{n_j+2}{2}}$, and let $\mathbf{s}''_j = \text{BitDecomp}(\mathbf{s}'_j, q_j) \in R_2^{\binom{n_j+2}{2}\lceil \log_2(q_j) \rceil}$. When $j \neq L$, let the key switching matrix be $\tau_{\mathbf{s}''_{j+1}, \mathbf{s}_j} = \text{SwitchKeyGen}(\mathbf{s}''_{j+1}, \mathbf{s}_j, q_j) \in R_{q_j}^{\binom{n_{j+1}+2}{2}\lceil \log_2(q_{j+1}) \rceil \lceil \log_2(q_j) \rceil \times (n_j+1)}$.

We set the secret key sk to be the set of all $\mathbf{s}_j$, and the public key pk to be the set of all A_j and the set of all $\tau_{\mathbf{s}''_{j+1}, \mathbf{s}_j}$.

FHEEncrypt($pk, m \in R_2$):

For a message m, output the ciphertext $\mathbf{c} = \text{Encrypt}(q_L, d, n_L, N_L, \chi, A_L, m) \in R_{q_L}^{n_L+1}$.

FHEDecrypt($sk, \mathbf{c} \in R_{q_j}^{n_j+1}$):

If $\mathbf{c}$ is encrypted under $\mathbf{s}_j$ for some $\mathbf{s}_j \in R_{q_j}^{n_j+1}$ in the secret key sk, output the message $m = \text{Decrypt}(\mathbf{s}_j, \mathbf{c}) \in R_2$.

FHERefresh($\mathbf{c} \in R_{q_j}^{\binom{n_j+2}{2}}, \tau_{\mathbf{s}_j'', \mathbf{s}_{j-1}} \in R_{q_j}^{\binom{n_j+2}{2}\lceil \log_2(q_j)\rceil \lceil \log_2(q_{j-1})\rceil \times (n_{j-1}+1)}, q_j \in \mathbb{Z}, q_{j-1} \in \mathbb{Z}$):

This function takes in a ciphertext $\mathbf{c} \in R_{q_j}^{\binom{n_j+2}{2}}$ encrypted under $\mathbf{s}_j' \in R_{q_j}^{\binom{n_j+2}{2}}$.

Let $\mathbf{w}_1 = \text{Powersof2}(\mathbf{c}, q_j) \in R_{q_j}^{\binom{n_j+2}{2}\lceil \log_2(q_j)\rceil}$, and let $\mathbf{w}_2 = \text{Scale}(\mathbf{w}_1, q_j, q_{j-1}, 2) \in R_{q_j}^{\binom{n_j+2}{2}\lceil \log_2(q_j)\rceil}$. Then output $\mathbf{w}_3 = \text{SwitchKey}(\tau_{\mathbf{s}_j'', \mathbf{s}_{j-1}}, \mathbf{w}_2, q_{j-1}) \in R_{q_{j-1}}^{n_{j-1}+1}$, which is a ciphertext encrypted under $\mathbf{s}_{j-1} \in R_{q_{j-1}}^{n_{j-1}+1}$ with modulus q_{j-1}.

FHEAdd($pk, \mathbf{c}_1 \in R_{q_j}^{n_j+1}, \mathbf{c}_2 \in R_{q_j}^{n_j+1}$):

This function requires that $\mathbf{c}_1$ and $\mathbf{c}_2$ are encrypted under the same secret key $\mathbf{s}_j \in R_{q_j}^{n_j+1}$. If they are not, we can apply the FHERefresh step to the ciphertext encrypted with the larger modulus. Say $\mathbf{c}_1 \in R_{q_i}^{n_i+1}$ is encrypted under $\mathbf{s}_i \in R_{q_i}^{n_i+1}$, and $\mathbf{c}_2 \in R_{q_k}^{n_k+1}$ is encrypted under $\mathbf{s}_k \in R_{q_k}^{n_k+1}$, where $i > k$, then we can append zeros to the end of $\mathbf{c}_1$ to get $\mathbf{c}_1' \in R_{q_i}^{\binom{n_i+2}{2}}$, so that $\mathbf{c}_1'$ is the same length as $\mathbf{s}_i' \in R_{q_i}^{\binom{n_i+2}{2}}$, and as such $\mathbf{c}_1 \cdot \mathbf{s}_i = \mathbf{c}_1' \cdot \mathbf{s}_i'$. We then use FHERefresh on $\mathbf{c}_1'$ (i.e. FHERefresh($\mathbf{c}_1'$, $\tau_{\mathbf{s}_i'', \mathbf{s}_{i-1}}$, q_i, q_{i-1})) to get a ciphertext encrypted under $\mathbf{s}_{i-1} \in R_{q_{i-1}}^{n_{i-1}+1}$, and we can continue this process until $\mathbf{c}_1$ and $\mathbf{c}_2$ are encrypted under the same $\mathbf{s}_j$.

Let $\mathbf{c}_3 = \mathbf{c}_1 + \mathbf{c}_2 \pmod{q_j}$, and then append zeros to the end of $\mathbf{c}_3$ until it has the same length as $\mathbf{s}_j' \in R_{q_j}^{\binom{n_j+2}{2}}$ to get $\mathbf{c}_3' \in R_{q_j}^{\binom{n_j+2}{2}}$. We can interpret $\mathbf{c}_3'$ as a ciphertext under $\mathbf{s}_j'$. Then output $\mathbf{c}_4 = \text{FHERefresh}(\mathbf{c}_3', \tau_{\mathbf{s}_j'', \mathbf{s}_{j-1}}, q_j, q_{j-1}) \in R_{q_{j-1}}^{n_{j-1}+1}$.

FHEMult($pk, \mathbf{c}_1 \in R_{q_j}^{n_j+!}, \mathbf{c}_2 \in R_{q_j}^{n_j+1}$):

This function requires that $\mathbf{c}_1$ and $\mathbf{c}_2$ are encrypted under the same secret key $\mathbf{s}_j \in R_{q_j}^{n_j+1}$. If they are not, we can apply the FHERefresh step to the ciphertext encrypted with the larger modulus. Say $\mathbf{c}_1 \in R_{q_i}^{n_i+1}$ is encrypted under $\mathbf{s}_i \in R_{q_i}^{n_i+1}$, and $\mathbf{c}_2 \in R_{q_k}^{n_k+1}$ is encrypted under $\mathbf{s}_k \in R_{q_k}^{n_k+1}$, where $i > k$, then we can append zeros to the end of $\mathbf{c}_1$ to get $\mathbf{c}_1' \in R_{q_i}^{\binom{n_i+2}{2}}$, so that $\mathbf{c}_1'$ is the same length as $\mathbf{s}_i' \in R_{q_i}^{\binom{n_i+2}{2}}$, and as such $\mathbf{c}_1 \cdot \mathbf{s}_i = \mathbf{c}_1' \cdot \mathbf{s}_i'$. We then use FHERefresh on $\mathbf{c}_1'$ (i.e. FHERefresh($\mathbf{c}_1'$, $\tau_{\mathbf{s}_i'', \mathbf{s}_{i-1}}$, q_i, q_{i-1})) to get a ciphertext encrypted under $\mathbf{s}_{i-1} \in R_{q_{i-1}}^{n_{i-1}+1}$, and we can continue this process until $\mathbf{c}_1$ and $\mathbf{c}_2$ are encrypted under the same $\mathbf{s}_j$.

Let $\mathbf{c}_3 \in R_{q_j}^{\binom{n_j+2}{2}}$ be the coefficient vector of $L_{\mathbf{c}_1,\mathbf{c}_2}(\Xi(\mathbf{x}))$, such that the entries of $\mathbf{c}_3$ are ordered in the same way as the entries of $\mathbf{s}'_j \in R_{q_j}^{\binom{n_j+2}{2}}$. We can interpret $\mathbf{c}_3$ as a ciphertext under $\mathbf{s}'_j$. Then output $\mathbf{c}_4 = \text{FHERefresh}(\mathbf{c}_3, \tau_{\mathbf{s}''_j,\mathbf{s}_{j-1}}, q_j, q_{j-1}) \in R_{q_{j-1}}^{n_{j-1}+1}$.

4.2 Correctness of the Encryption Scheme

This section gives a proof that encryption and decryption work properly. It also provides a bound on the error terms that are required for proper decryption.

To check that the encryption scheme works, we need to check that $m = [\langle \mathbf{c}, \mathbf{s} \rangle]_2$. We have:

$$\begin{aligned}
\mathbf{c} \cdot \mathbf{s} &= (\mathbf{m} + A^T\mathbf{r}) \cdot \mathbf{s} \\
&= \mathbf{m}^T\mathbf{s} + \mathbf{r}^T A\mathbf{s} \\
&= \begin{pmatrix} m & 0 & \cdots & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \mathbf{s}'[1] \\ \vdots \\ \mathbf{s}'[n] \end{pmatrix} + \mathbf{r}^T \begin{bmatrix} \mathbf{b} \mid -A' \end{bmatrix} \begin{pmatrix} 1 \\ \mathbf{s}'[1] \\ \vdots \\ \mathbf{s}'[n] \end{pmatrix} \\
&= m + \mathbf{r}^T(\mathbf{b} - A'\mathbf{s}') \\
&= m + \mathbf{r}^T(2\mathbf{e}).
\end{aligned}$$

Since all of our operations are being computed over $\mathbb{Z}_q$, where we take values in the range $(-q/2, q/2]$, to ensure that $m + 2\mathbf{r}^T\mathbf{e}$ will have the same parity as m, we can impose the restriction that $\|m + 2\mathbf{r}^T\mathbf{e}\| < q/2$. Let B_χ be the bound on the magnitude of elements sampled from χ, i.e. $B_\chi = \sup\{\|\text{a}\| : \text{a is sampled from } \chi\}$. We have:

$$\begin{aligned}
\|\mathbf{c} \cdot \mathbf{s}\| &= \|m + 2\mathbf{r}^T\mathbf{e}\| \\
&\leq m + 2\|\mathbf{r} \cdot \mathbf{e}\| \\
&= m + 2\|\sum_{i=1}^{N} \mathbf{r}[i]\mathbf{e}[i]\| \\
&\leq m + 2\sum_{i=1}^{N} \|\mathbf{r}[i]\mathbf{e}[i]\|
\end{aligned}$$

$$
\begin{aligned}
&\leq m + 2\gamma_R \sum_{i=1}^{N} \|\mathbf{r}[i]\| \|\mathbf{e}[i]\| \text{ (by Cauchy-Schwarz)} \\
&= m + 2\gamma_R \sum_{i=1}^{N} \sqrt{\sum_{j=0}^{d-1} (\mathbf{r}[i]_j)^2} \|\mathbf{e}[i]\| \\
&\leq m + 2\gamma_R \sum_{i=1}^{N} \sqrt{\sum_{j=0}^{d-1} (1)^2} \|\mathbf{e}[i]\| \text{ (since } \mathbf{r} \in R_2^N) \\
&\leq m + 2\gamma_R \sqrt{d} \sum_{i=1}^{N} B_\chi \\
&\leq m + 2\gamma_R \sqrt{d} N B_\chi \\
&\leq m + 2\sqrt{d}\sqrt{d} N B_\chi.
\end{aligned}
$$

Since we want $\|\mathbf{c}\cdot\mathbf{s}\|$ to be bounded by $q/2$, we achieve this result when $m+2\gamma_R\sqrt{d}NB_\chi < q/2$. Since $N = \lceil (2n+1)\log_2(q) \rceil$, we get that $B_\chi < (q/2-1)/(2d\lceil (2n+1)\log_2(q)\rceil)$.

Notice that since $\|\mathbf{e}[i]\| \leq B_\chi$ for all $i \in \{1, \ldots, N\}$, if $d = 1$ as with the standard-LWE version of this cryptosystem, then we need

$$
\mathbf{e} \in \left\{ -\left\lfloor \frac{(q/2-1)}{2\lceil (2n+1)\log_2(q)\rceil} \right\rfloor, \ldots, \left\lfloor \frac{(q/2-1)}{2\lceil (2n+1)\log_2(q)\rceil} \right\rfloor \right\}^N
$$

in order to have decryption work properly.

However, in the general-LWE instance, where we do not have the restriction that $d = 1$, we see that $\|\mathbf{e}[i]\| \leq B_\chi$ for all $i \in \{1, \ldots, N\}$, so $\sqrt{\sum_{j=0}^{d-1} |\mathbf{e}[i]_j|^2} \leq B_\chi$. This inequality holds if $d|\mathbf{e}[i]_j|^2 \leq B_\chi^2$ for all $j \in \{0, \ldots, d-1\}$, so if we impose the restriction that $|\mathbf{e}[i]_j| \leq \frac{B_\chi}{\sqrt{d}}$, then certainly $\|\mathbf{e}[i]\| \leq B_\chi$. Thus, for decryption to work in the general-LWE instance, we need

$$
\mathbf{e} \in \{a_0 + a_1 x + \cdots + a_{d-1} x^{d-1} : a_i \in S, \forall i \in \{0, \ldots, d-1\}\}^N.
$$

where

$$
S = \left\{ -\left\lfloor \frac{(q/2-1)}{2d^{3/2}\lceil (2n+1)\log_2(q)\rceil} \right\rfloor, \ldots, \left\lfloor \frac{(q/2-1)}{2d^{3/2}\lceil (2n+1)\log_2(q)\rceil} \right\rfloor \right\}.
$$

4.3 Modulus Switching and Noise Management

In the paper describing the BGV encryption scheme [2], the authors used a couple of lemmas for modulus switching. The following proofs correct the lemmas used, and adjust the statements of those lemmas. These lemmas are used to show that after performing a modulus switching operation, the noise $|[\mathbf{c} \cdot \mathbf{s}]_q|$ in the ciphertext $\mathbf{c}$ under the secret key $\mathbf{s}$ is less than $q/2$, which preserves the ability for decryption to work properly.

The following lemma is a strengthening of Lemma 1 from the paper describing the BGV encryption scheme. This lemma deals with the instances when the ring that we are using is just the integers. It removes the restriction that $|[\mathbf{c} \cdot \mathbf{s}]_q|$ must be strictly less than $q/2 - (q/p) \cdot \|\mathbf{s}\|_1$. This lets a greater range of noise to be allowed, while still preserving the correctness of decryption.

Lemma 4.3.1. *Let p and q be odd moduli, and let $\boldsymbol{c}$ and $\boldsymbol{s}$ be length-n vectors with integer entries. Let $\boldsymbol{c}'$ be the integer vector closest to $(p/q) \cdot \boldsymbol{c}$ such that $\boldsymbol{c}' \equiv \boldsymbol{c} \bmod 2$. Then for any $\boldsymbol{s}$ such that $|[\boldsymbol{c} \cdot \boldsymbol{s}]_q| \leq q/2 - (q/p) \cdot \|\boldsymbol{s}\|_1$, we get that*

$$[\boldsymbol{c}' \cdot \boldsymbol{s}]_p \equiv [\boldsymbol{c} \cdot \boldsymbol{s}]_q \bmod 2 \text{ and } |[\boldsymbol{c}' \cdot \boldsymbol{s}]_p| < \frac{p}{q} \cdot |[\boldsymbol{c} \cdot \boldsymbol{s}]_q| + \|\boldsymbol{s}\|_1.$$

Proof. We know that $[\mathbf{c} \cdot \mathbf{s}]_q = \mathbf{c} \cdot \mathbf{s} - kq$ for some $k \in \mathbb{Z}$ by definition. Let $e = \mathbf{c}' \cdot \mathbf{s} - kp$, for the same value of k. Notice that

$$\begin{aligned} e &\equiv \mathbf{c}' \cdot \mathbf{s} - kp \bmod 2 \\ &\equiv \mathbf{c} \cdot \mathbf{s} - kq \bmod 2 \text{ (since } \mathbf{c}' \equiv \mathbf{c} \bmod 2 \text{ and } p \equiv q \bmod 2 \text{)} \\ &\equiv [\mathbf{c} \cdot \mathbf{s}]_q \bmod 2. \end{aligned}$$

Thus, to show that $[\mathbf{c}' \cdot \mathbf{s}]_p \equiv [\mathbf{c} \cdot \mathbf{s}]_q \bmod 2$, we only need to show that $e = [\mathbf{c}' \cdot \mathbf{s}]_p$. But since $e = \mathbf{c}' \cdot \mathbf{s} - kp$, and $[\mathbf{c}' \cdot \mathbf{s}]_p = \mathbf{c}' \cdot \mathbf{s} - mp$ for some $m \in \mathbb{Z}$ such that $|[\mathbf{c}' \cdot \mathbf{s}]_p| < p/2$, we just need to show that $|e| < p/2$.

First notice that $|\mathbf{c}'[j] - (p/q)\mathbf{c}[j]| < 1$ for all $j \in \{1, \ldots, n\}$. This is because $\mathbf{c}'$ is the closest integer vector to $(p/q)\mathbf{c}$ such that $\mathbf{c}' \equiv \mathbf{c} \bmod 2$, so $|\mathbf{c}'[j] - (p/q)\mathbf{c}[j]| \leq 1$. However, $|\mathbf{c}'[j] - (p/q)\mathbf{c}[j]| \neq 1$, because otherwise $(p/q)\mathbf{c}[j] \in \mathbb{Z}$. Since p and q are both odd, this would mean that $(p/q)\mathbf{c}[j]$ must have the same parity as $\mathbf{c}[j]$, so $(p/q)\mathbf{c}[j] \equiv \mathbf{c}[j] \bmod 2$. Thus, $\mathbf{c}'[j] = (p/q)\mathbf{c}[j]$, and $|\mathbf{c}'[j] - (p/q)\mathbf{c}[j]| = 0$. This also shows that $\mathbf{c}'$ is well defined.

Finally,

$$\begin{aligned}
|e| &= |\mathbf{c}' \cdot \mathbf{s} - kp| \\
&= \left| \mathbf{c}' \cdot \mathbf{s} - kp + \frac{p}{q}[\mathbf{c} \cdot \mathbf{s}]_q - \frac{p}{q}[\mathbf{c} \cdot \mathbf{s}]_q \right| \\
&= \left| \frac{p}{q}[\mathbf{c} \cdot \mathbf{s}]_q + \mathbf{c}' \cdot \mathbf{s} - \frac{p}{q}(kq + [\mathbf{c} \cdot \mathbf{s}]_q) \right| \\
&= \left| \frac{p}{q}[\mathbf{c} \cdot \mathbf{s}]_q + \mathbf{c}' \cdot \mathbf{s} - \frac{p}{q}\mathbf{c} \cdot \mathbf{s} \right| \\
&= \left| \frac{p}{q}[\mathbf{c} \cdot \mathbf{s}]_q + (\mathbf{c}' - \frac{p}{q}\mathbf{c}) \cdot \mathbf{s} \right| \\
&\leq \frac{p}{q} |[\mathbf{c} \cdot \mathbf{s}]_q| + \left| (\mathbf{c}' - \frac{p}{q}\mathbf{c}) \cdot \mathbf{s} \right| \\
&= \frac{p}{q} |[\mathbf{c} \cdot \mathbf{s}]_q| + \left| \sum_{j=1}^{n} (\mathbf{c}'[j] - \frac{p}{q}\mathbf{c}[j])\mathbf{s}[j] \right| \\
&\leq \frac{p}{q} |[\mathbf{c} \cdot \mathbf{s}]_q| + \sum_{j=1}^{n} \left| (\mathbf{c}'[j] - \frac{p}{q}\mathbf{c}[j])\mathbf{s}[j] \right| \\
&< \frac{p}{q} |[\mathbf{c} \cdot \mathbf{s}]_q| + \sum_{j=1}^{n} |(1)||\mathbf{s}[j]| \\
&= \frac{p}{q} |[\mathbf{c} \cdot \mathbf{s}]_q| + \|\mathbf{s}\|_1 \text{ (giving the second desired result)} \\
&\leq \frac{p}{q} \left(\frac{q}{2} - \frac{q}{p} \cdot \|\mathbf{s}\|_1 \right) + \|\mathbf{s}\|_1 \text{ (by assumption)} \\
&= \frac{p}{2}.
\end{aligned}$$

□

The next lemma is a strengthening of Lemma 4 from the paper describing the BGV encryption scheme in the case where we are working mod 2. This lemma expands upon Lemma 4.3.1 to the case where the ring we are using is a polynomial ring over the integers. It removes the restriction that $\|[\mathbf{c} \cdot \mathbf{s}]_q\|$ must be strictly less than $q/2 - (q/p) \cdot \sqrt{d} \cdot \gamma_R \cdot \|\mathbf{s}\|_{1,R}$. This lets a greater range of noise to be allowed, while still preserving the correctness of decryption.

Lemma 4.3.2. *Let d be the degree of the ring R, and let $q > p > 2$ be positive integers such that $q \equiv p \equiv 1 \bmod 2$. Let $\boldsymbol{c} \in R^n$ and let $\boldsymbol{c'} = Scale(\boldsymbol{c}, q, p, 2)$, the closest vector to $(p/q)\boldsymbol{c}$ with entries in R such that $\boldsymbol{c'} \equiv \boldsymbol{c} \bmod 2$. Then for all $\boldsymbol{s} \in R^n$ such that $\|[\boldsymbol{c} \cdot \boldsymbol{s}]_q\| \leq q/2 - (q/p) \cdot \sqrt{d} \cdot \gamma_R \cdot \|\boldsymbol{s}\|_{1,R}$, we get that*

$$[\boldsymbol{c'} \cdot \boldsymbol{s}]_p \equiv [\boldsymbol{c} \cdot \boldsymbol{s}]_q \bmod 2 \text{ and } \|[\boldsymbol{c'} \cdot \boldsymbol{s}]_p\| < \frac{p}{q} \cdot \|[\boldsymbol{c} \cdot \boldsymbol{s}]_q\| + \sqrt{d} \cdot \gamma_R \cdot \|\boldsymbol{s}\|_{1,R}.$$

Proof. We know that $[\mathbf{c} \cdot \mathbf{s}]_q = \mathbf{c} \cdot \mathbf{s} - kq$ for some $k \in R$ by definition. Let $e = \mathbf{c'} \cdot \mathbf{s} - kp$, for the same value of k. Notice that

$$\begin{aligned} e &\equiv \mathbf{c'} \cdot \mathbf{s} - kp \bmod 2 \\ &\equiv \mathbf{c} \cdot \mathbf{s} - kq \bmod 2 \text{ (since } \mathbf{c'} \equiv \mathbf{c} \bmod 2 \text{ and } p \equiv q \bmod 2) \\ &\equiv [\mathbf{c} \cdot \mathbf{s}]_q \bmod 2. \end{aligned}$$

Thus, to show that $[\mathbf{c'} \cdot \mathbf{s}]_p \equiv [\mathbf{c} \cdot \mathbf{s}]_q \bmod 2$, we just need to show that $e = [\mathbf{c'} \cdot \mathbf{s}]_p$. But since $e = \mathbf{c'} \cdot \mathbf{s} - kp$, and $[\mathbf{c'} \cdot \mathbf{s}]_p = \mathbf{c'} \cdot \mathbf{s} - mp$ for some $m \in R$ such that each coefficient of $[\mathbf{c'} \cdot \mathbf{s}]_p$ is less than $p/2$, we just need to show that $\|e\| < p/2$. This is because if $\|e\| < p/2$, then all of the coefficients of e must be less than $p/2$, since the Euclidean norm of a vector is always greater than the largest coefficient of that vector.

First notice that $|\mathbf{c'}[j]_k - (p/q)\mathbf{c}[j]_k| < 1$ for all $j \in \{1, \ldots, n\}$, and all $k \in \{0, \ldots, d-1\}$. This is because $\mathbf{c'}$ is the closest integer vector to $(p/q)\mathbf{c}$ such that $\mathbf{c'} \equiv \mathbf{c} \bmod 2$, so $|\mathbf{c'}[j]_k - (p/q)\mathbf{c}[j]_k| \leq (2/2) = 1$. However, $|\mathbf{c'}[j]_k - (p/q)\mathbf{c}[j]_k| \neq 1$, because $\mathbf{c'}[j]_k$, $\mathbf{c}[j]_k$, p, and q are all integers, so this would mean that $(p/q)\mathbf{c}[j]_k \in \mathbb{Z}$. Since p and q are both odd, this would mean that $(p/q)\mathbf{c}[j]_k$ must have the same parity as $\mathbf{c}[j]_k$, so $(p/q)\mathbf{c}[j]_k \equiv \mathbf{c}[j]_k \bmod 2$. Thus, $\mathbf{c'}[j]_k = (p/q)\mathbf{c}[j]_k$, and $|\mathbf{c'}[j]_k - (p/q)\mathbf{c}[j]_k| = 0$. This also shows that $\mathbf{c'}$ is well defined. Finally,

$$\begin{aligned} \|e\| &= \|\mathbf{c'} \cdot \mathbf{s} - kp\| \\ &= \left\| \mathbf{c'} \cdot \mathbf{s} - kp + \frac{p}{q}[\mathbf{c} \cdot \mathbf{s}]_q - \frac{p}{q}[\mathbf{c} \cdot \mathbf{s}]_q \right\| \end{aligned}$$

$$
\begin{aligned}
&= \left\| \frac{p}{q}[\mathbf{c}\cdot\mathbf{s}]_q + \mathbf{c}'\cdot\mathbf{s} - \frac{p}{q}(kq + [\mathbf{c}\cdot\mathbf{s}]_q) \right\| \\
&= \left\| \frac{p}{q}[\mathbf{c}\cdot\mathbf{s}]_q + \mathbf{c}'\cdot\mathbf{s} - \frac{p}{q}\mathbf{c}\cdot\mathbf{s} \right\| \\
&= \left\| \frac{p}{q}[\mathbf{c}\cdot\mathbf{s}]_q + (\mathbf{c}' - \frac{p}{q}\mathbf{c})\cdot\mathbf{s} \right\| \\
&\leq \frac{p}{q}\left\|[\mathbf{c}\cdot\mathbf{s}]_q\right\| + \left\| (\mathbf{c}' - \frac{p}{q}\mathbf{c})\cdot\mathbf{s} \right\| \\
&= \frac{p}{q}\left\|[\mathbf{c}\cdot\mathbf{s}]_q\right\| + \left\| \sum_{j=1}^{n} (\mathbf{c}'[j] - \frac{p}{q}\mathbf{c}[j])\mathbf{s}[j] \right\| \\
&\leq \frac{p}{q}\left\|[\mathbf{c}\cdot\mathbf{s}]_q\right\| + \sum_{j=1}^{n} \left\| (\mathbf{c}'[j] - \frac{p}{q}\mathbf{c}[j])\mathbf{s}[j] \right\| \\
&\leq \frac{p}{q}\left\|[\mathbf{c}\cdot\mathbf{s}]_q\right\| + \gamma_R \sum_{j=1}^{n} \left\| (\mathbf{c}'[j] - \frac{p}{q}\mathbf{c}[j]) \right\| \|\mathbf{s}[j]\| \\
&= \frac{p}{q}\left\|[\mathbf{c}\cdot\mathbf{s}]_q\right\| + \gamma_R \sum_{j=1}^{n} \sqrt{\sum_{k=0}^{d-1} (\mathbf{c}'[j]_k - \frac{p}{q}\mathbf{c}[j]_k)^2}\, \|\mathbf{s}[j]\| \\
&< \frac{p}{q}\left\|[\mathbf{c}\cdot\mathbf{s}]_q\right\| + \gamma_R \sum_{j=1}^{n} \sqrt{\sum_{k=0}^{d-1} (1)^2}\, \|\mathbf{s}[j]\| \\
&= \frac{p}{q}\left\|[\mathbf{c}\cdot\mathbf{s}]_q\right\| + \gamma_R \cdot \sqrt{d} \cdot \sum_{j=1}^{n} \|\mathbf{s}[j]\| \\
&= \frac{p}{q}\left\|[\mathbf{c}\cdot\mathbf{s}]_q\right\| + \gamma_R \cdot \sqrt{d} \cdot \|\mathbf{s}\|_{1,R} \text{ (giving the second desired result sortof)} \\
&= \frac{p}{q}\left(\left\|[\mathbf{c}\cdot\mathbf{s}]_q\right\| + \frac{q}{p} \cdot \gamma_R \cdot \sqrt{d} \cdot \|\mathbf{s}\|_{1,R} \right) \\
&\leq \frac{p}{q}\left(\frac{q}{2}\right) \text{ (by assumption)} \\
&= \frac{p}{2}.
\end{aligned}
$$

□

The next lemma is a restatement of Lemma 4 from the paper describing the BGV encryption scheme in the case where we are working mod r, where $r \neq 2$. The original proof purported to show that

$$\|[\mathbf{c}' \cdot \mathbf{s}]_p\| < \frac{p}{q} \cdot \|[\mathbf{c} \cdot \mathbf{s}]_q\| + \frac{r}{2} \cdot \sqrt{d} \cdot \gamma_R \cdot \|\mathbf{s}\|_{1,R}$$

given the same set of assumptions. However, this is not true, as will be shown after the proof of the corrected statement.

Since this statement is not true, it cannot be guaranteed that the noise $\|[\mathbf{c}' \cdot \mathbf{s}]_p\|$ satisfies

$$\|[\mathbf{c}' \cdot \mathbf{s}]_p\| < B,$$

where $B = \frac{p}{q} \cdot \|[\mathbf{c} \cdot \mathbf{s}]_q\| + \frac{r}{2} \cdot \sqrt{d} \cdot \gamma_R \cdot \|\mathbf{s}\|_{1,R}$. Since decryption only works when the noise is strictly less than $p/2$, and adding and multiplying ciphertexts increases the noise of the resulting ciphertexts, fewer operations can be done to ensure proper decryption. Since the basic encryption scheme works with a decreasing ladder of moduli congruent to 1 mod 2, this lemma does not affect this scheme. However, the batching optimization that is mentioned by Brakerski et al. [2] is dependent on this lemma.

Lemma 4.3.3. *Let d be the degree of the ring R, and let $q > p > r$ be positive integers such that $q \equiv p \equiv 1$ mod r. Let $\boldsymbol{c} \in R^n$ and let $\boldsymbol{c}' = Scale(\boldsymbol{c}, q, p, r)$, a closest vector to $(p/q)\boldsymbol{c}$ with entries in R such that $\boldsymbol{c}' \equiv \boldsymbol{c}$ mod r. Then for all $\boldsymbol{s} \in R^n$ such that $\|[\boldsymbol{c} \cdot \boldsymbol{s}]_q\| < q/2 - (q/p) \cdot (r/2) \cdot \sqrt{d} \cdot \gamma_R \cdot \|\boldsymbol{s}\|_{1,R}$, we get that*

$$[\boldsymbol{c}' \cdot \boldsymbol{s}]_p \equiv [\boldsymbol{c} \cdot \boldsymbol{s}]_q \text{ mod } r \text{ and } \|[\boldsymbol{c}' \cdot \boldsymbol{s}]_p\| \leq \frac{p}{q} \cdot \|[\boldsymbol{c} \cdot \boldsymbol{s}]_q\| + \frac{r}{2} \cdot \sqrt{d} \cdot \gamma_R \cdot \|\boldsymbol{s}\|_{1,R}.$$

Proof. We know that $[\mathbf{c} \cdot \mathbf{s}]_q = \mathbf{c} \cdot \mathbf{s} - kq$ for some $k \in R$ by definition. Let $e = \mathbf{c}' \cdot \mathbf{s} - kp$, for the same value of k. Notice that

$$\begin{aligned} e &\equiv \mathbf{c}' \cdot \mathbf{s} - kp \bmod r \\ &\equiv \mathbf{c} \cdot \mathbf{s} - kq \bmod r \text{ (since } \mathbf{c}' \equiv \mathbf{c} \bmod r \text{ and } p \equiv q \bmod r) \\ &\equiv [\mathbf{c} \cdot \mathbf{s}]_q \bmod r. \end{aligned}$$

Thus, to show that $[\mathbf{c}' \cdot \mathbf{s}]_p \equiv [\mathbf{c} \cdot \mathbf{s}]_q \bmod r$, we just need to show that $e = [\mathbf{c}' \cdot \mathbf{s}]_p$. But since $e = \mathbf{c}' \cdot \mathbf{s} - kp$, and $[\mathbf{c}' \cdot \mathbf{s}]_p = \mathbf{c}' \cdot \mathbf{s} - mp$ for some $m \in R$ such that each coefficient of $[\mathbf{c}' \cdot \mathbf{s}]_p$ is less than $p/2$, we just need to show that $\|e\| < p/2$. This is because if $\|e\| < p/2$, then all of the coefficients of e must be less than $p/2$, since the Euclidean norm of a vector is always greater than the largest coefficient of that vector.

First notice that $|\mathbf{c}'[j]_k - (p/q)\mathbf{c}[j]_k| \leq (r/2)$ for all $j \in \{1, \ldots, n\}$, and all $k \in \{0, \ldots, d-1\}$. This is because $\mathbf{c}'$ is the closest integer vector to $(p/q)\mathbf{c}$ such that $\mathbf{c}' \equiv \mathbf{c} \bmod r$, so $|\mathbf{c}'[j]_k - (p/q)\mathbf{c}[j]_k| \leq (r/2)$.

As shown in the proof of Lemma 4.3.2,

$$\|e\| \leq \frac{p}{q} \|[\mathbf{c} \cdot \mathbf{s}]_q\| + \gamma_R \sum_{j=1}^{n} \sqrt{\sum_{k=0}^{d-1} (\mathbf{c}'[j]_k - \frac{p}{q}\mathbf{c}[j]_k)^2} \, \|\mathbf{s}[j]\| \, .$$

Thus, we get the following inequalities.

$$\begin{aligned}
\|e\| &\leq \frac{p}{q} \|[\mathbf{c} \cdot \mathbf{s}]_q\| + \gamma_R \sum_{j=1}^{n} \sqrt{\sum_{k=0}^{d-1} (\mathbf{c}'[j]_k - \frac{p}{q}\mathbf{c}[j]_k)^2} \, \|\mathbf{s}[j]\| \\
&\leq \frac{p}{q} \|[\mathbf{c} \cdot \mathbf{s}]_q\| + \gamma_R \sum_{j=1}^{n} \sqrt{\sum_{k=0}^{d-1} \left(\frac{r}{2}\right)^2} \, \|\mathbf{s}[j]\| \\
&= \frac{p}{q} \|[\mathbf{c} \cdot \mathbf{s}]_q\| + \gamma_R \cdot \sqrt{d} \cdot \frac{r}{2} \sum_{j=1}^{n} \|\mathbf{s}[j]\| \\
&= \frac{p}{q} \|[\mathbf{c} \cdot \mathbf{s}]_q\| + \gamma_R \cdot \sqrt{d} \cdot \frac{r}{2} \|\mathbf{s}\|_{1,R} \text{ (giving the second desired result)} \\
&= \frac{p}{q} \left(\|[\mathbf{c} \cdot \mathbf{s}]_q\| + \frac{q}{p} \cdot \gamma_R \cdot \sqrt{d} \cdot \frac{r}{2} \|\mathbf{s}\|_{1,R} \right) \\
&< \frac{p}{q} \left(\frac{q}{2} \right) \text{ (by assumption)} \\
&= \frac{p}{2}.
\end{aligned}$$

□

The above proof shows that the reworded lemma is true; however, it remains to be shown that the original statement is in fact false. The original statement of this lemma said that $\|[\mathbf{c}' \cdot \mathbf{s}]_p\| < (p/q) \cdot \|[\mathbf{c} \cdot \mathbf{s}]_q\| + (r/2) \cdot \sqrt{d} \cdot \gamma_R \cdot \|\mathbf{s}\|_{1,R}$. However, in some cases, the adjusted inequality $\|[\mathbf{c}' \cdot \mathbf{s}]_p\| \leq (p/q) \cdot \|[\mathbf{c} \cdot \mathbf{s}]_q\| + (r/2) \cdot \sqrt{d} \cdot \gamma_R \cdot \|\mathbf{s}\|_{1,R}$ holds with equality. For instance, if we let $d = 1$, $q = 10$, $p = 7$, $r = 3$, $n = 1$, $\mathbf{c} = 15$, and $\mathbf{s} = 2$ we get that

$\gamma_R = 1$ and

$$\|[\mathbf{c} \cdot \mathbf{s}]_q\| = \|[30]_{10}\| = 0 < \frac{5}{7} = \frac{10}{2} - \frac{10}{7} \cdot \frac{3}{2} \cdot \sqrt{1} \cdot 1 \cdot \|2\|_{1,R} = \frac{q}{2} - \frac{q}{p} \cdot \frac{r}{2} \cdot \sqrt{d} \cdot \gamma_R \cdot \|\mathbf{s}\|_{1,R}$$

which satisfies the assumption in the lemma.

Since $\mathbf{c}'$ is the closest integer to $(p/q)\mathbf{c} = 10.5$ such that $\mathbf{c}' \equiv \mathbf{c} \pmod{r}$, either $\mathbf{c}' = 12$ or $\mathbf{c}' = 9$ satisfies this. This also means that the function $\text{Scale}(\mathbf{c}, q, p, r)$ is not well defined over all values of r. Despite this, if we pick $\mathbf{c}' = 12$, then we get that

$$\|[\mathbf{c}' \cdot \mathbf{s}]_p\| = \|[24]_7\| = 3 = \frac{7}{10}(0) + \frac{3}{2} \cdot \sqrt{1} \cdot 1 \cdot 2 = \frac{p}{q}\|[\mathbf{c} \cdot \mathbf{s}]_q\| + \frac{r}{2} \cdot \sqrt{d} \cdot \gamma_R \cdot \|\mathbf{s}\|_{1,R}.$$

The equation $\|[\mathbf{c}' \cdot \mathbf{s}]_p\| = \frac{p}{q}\|[\mathbf{c} \cdot \mathbf{s}]_q\| + \frac{r}{2} \cdot \sqrt{d} \cdot \gamma_R \cdot \|\mathbf{s}\|_{1,R}$ also holds if we pick $\mathbf{c}' = 9$. Since this statement is true with equality, the original statement of the lemma is false.

4.4 Security Parameters for the Fully Homomorphic Encryption Scheme

This section provides a table that gives concrete parameters that can be used to satisfy a number of different security levels for the BGV encryption scheme. It is based on an analysis of the runtime of the BKZ lattice reduction algorithm, as well as information on the optimal lattice dimension needed to get a reduced basis in this algorithm, and hence break the BGV encryption scheme.

The BGV encryption scheme makes use of the Learning with Errors assumption when generating the public key. Recall that the public key A is a $R_q^{N \times (n+1)}$ matrix of the form $pk = A = \left[\begin{array}{c|c} \mathbf{b} & -A' \end{array} \right]$ where $\mathbf{b} = A'\mathbf{s}' + 2\mathbf{e} \in R_q^n$, A' is generated uniformly from $R_q^{N \times n}$, and $\mathbf{e} \in R_q^N$ is generated from χ^N.

We can see that trying to determine the unknown part $\mathbf{s}'$ of the secret key is equivalent to the Search Learning with Errors Problem. We are trying to determine $\mathbf{s}' \in R_q^n$ given the set of pairs $(((A')^T)_i, \mathbf{b}[i]) \in R_q^n \times R_q$, where $((A')^T)_i$ is sampled uniformly from R_q^n, e_i is sampled from χ, and $\mathbf{b}[i] = ((A')^T)_i \cdot \mathbf{s}' + e_i$ for $i = 1, \ldots, N$.

This, however, is just an instance of the Closest Vector Problem. We can think of the columns of A' as a set of vectors in a lattice. Then $A'\mathbf{s}' = \sum_{j=1}^{n} A'_j \cdot \mathbf{s}'[j]$ is just an integer

combination of the columns of A', and is therefore a point in the lattice generated by the columns of A'. We can write this lattice as

$$\Lambda(A') = \{\mathbf{z} \in R_q^N : \exists\ \mathbf{s}' \in R_q^n \text{ such that } \mathbf{z} = A'\mathbf{s}'\}.$$

Thus, if we can find which lattice point is closest to $\mathbf{b}$, and hence solve the Closest Vector Problem, we can determine the value of $A'\mathbf{s}'$. Since we are given the value of A' in the public key, we can then use simple row reduction to determine the entries in $\mathbf{s}'$.

However, in order to solve the Closest Vector Problem using the best method known, we need to reduce the basis of the lattice, and then use this reduced basis to determine the closest vector point in the lattice. If the reduced basis is short enough, we can determine the closest vector.

Lindner and Peikert [16] use the BKZ algorithm to compute a reduced basis. The BKZ algorithm includes a blocksize parameter k that ranges from 2 to the dimension of the lattice that is to be reduced. By increasing this parameter, the quality of the reduced basis increases, but the time the algorithm takes to compute a basis becomes much greater. In fact, when $k \geq 30$, this algorithm becomes practically infeasible.

Gama and Nguyen [7] observed that the most relevant parameter that determines the quality of a reduced basis, as well as the time taken to run the basis reduction algorithm is the *Hermite factor*. For an m-dimensional lattice Λ with basis $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_m\}$, where the basis vectors are ordered in increasing length, the Hermite factor of Λ is

$$\delta^m = \frac{\|\mathbf{b}_1\|}{\det(\Lambda)^{1/m}}.$$

We will say that δ, the mth root of δ^m, is the *root-Hermite factor*.

Since we are trying to determine $\mathbf{s}'$ given $\mathbf{b} = A'\mathbf{s}' + 2\mathbf{e} \in R_q^n$, where $A' \in R_q^{N \times n}$, and $N \geq n$, the extra rows of A' give us redundant information to solve the problem. Since we can ignore some of the rows of A', we can just use the amount of information that gives us the most effective lattice attack, and thus the best ability to attack the scheme. It turns out that for any desired root-Hermite factor δ, the best subdimension is

$$N_{OPT} = \sqrt{n \log_2(q) / \log_2(\delta)}$$

for attacking the LWE Problem. If the subdimension N happens to be less than the optimum value, then we can just use the original subdimension in our attack. This doesn't affect the analysis of the security levels though as we are finding a lower bound on the time needed to solve the LWE Problem.

Lindner and Peikert used this optimal value of N in experiments to compute a lower bound on the computational time needed to solve the LWE Problem over random q-ary lattices. They also used the result by Gama and Nguyen [7] that on random lattices of large dimensions (above 200), the run-time of the BKZ algorithm required to get a particular δ is primarily dependent on δ alone, as the dimension of the lattice and its determinant only contribute second order terms. In these experiments, they used a single 2.3 GHz AMD Opteron machine, and used the single-precision floating-point BKZ implementation from the Number Theory Library (NTL) 5.5.2 for C++ [16]. Any attack on an LWE instance that could be used in practice would have to have a higher precision, which would increase the running time of the algorithm, so this analysis is once again an underestimate of the actual time needed to solve the LWE Problem.

Let $T_{BKZ}(\delta)$ be the time required to run the BKZ algorithm for a given δ. Using a least-squares regression, Lindner and Peikert found that $\log_2(T_{BKZ}(\delta)) = 1.806/\log_2(\delta) - 91$. They then adjusted this equation to deal with improvements to the basis reduction techniques to get that $\log_2(T_{BKZ}(\delta)) = 1.8/\log_2(\delta) - 110$ [16].

However, we can generalize this equation to take into account any computer being used, by expressing this equation as the number of operations required by the computer. Since the aforementioned experiments used a 2.3 GHz processor, we get that the number of operations 2^λ is $T_{BKZ}(\delta)/(2.3 \times 10^9)$. Thus, our security parameter $\lambda = T_{BKZ}(\delta) - 31.1$, and so

$$\lambda(\delta) = 1.8/\log_2(\delta) - 78.9.$$

Using the analysis of Lindner and Peikert [16], Gentry, Halevi, and Smart [13] were able to derive security parameters for any given security level. To do this, they had to make a number of assumptions.

The first assumption is that the security level is dependent on the ratio q/σ, where σ^2 is the variance of the distribution χ from which error vectors are sampled, and not on q and σ independently. The second assumption for their analysis is that the optimal runtime/advantage ratio happens when we maximize the advantage. This means that we have to assume that the best option for an attack is to spend a large amount of time finding a high quality reduced basis for the lattice. The third assumption is that to get an advantage of close to $1/2$, we need to reduce the basis enough so that the smallest vector in the basis (i.e. $\mathbf{b}_1$) has size less than q/σ.

Gentry, Halevi, and Smart considered the case where $\sigma = 1$, but we will generalize this for any σ. To do this, we need to make use of the second and third assumptions.

Say we want a cryptosystem that has a certain security parameter λ. Then, we will need to find a lattice that requires a basis with root-Hermite factor δ to solve the LWE

Problem, where δ is determined by $\lambda = 1.8/\log_2(\delta) - 78.9$. Since $\delta^N = \frac{\|\mathbf{b}_1\|}{\det(\Lambda)^{1/N}}$, we need to find a reduced lattice basis such that $\|\mathbf{b}_1\|$ must satisfy $\|\mathbf{b}_1\| = \delta^N \det(\Lambda)^{1/N}$.

Also, since we are working over q-ary lattices of rank n, we get that $\det(\Lambda) = q^n$ with high probability. Therefore, we need to find a reduced basis with $\|\mathbf{b}_1\| = \delta^N q^{n/N}$. But since we need to reduce the basis so that the shortest vector is of size less than q/σ, we need that $q/\sigma \leq \delta^N q^{n/N}$.

However, an LWE attacker will be able to choose the dimension of the matrix we are dealing with. As such, the attacker will pick the optimal dimension for an attack, which is $N = \sqrt{n\log_2(q)/\log_2(\delta)}$, so our choice of parameters needs to take that into consideration. Thus, we get

$$\begin{aligned}
\log_2(q/\sigma) &\leq \log_2(\delta^N q^{n/N}) \\
&= N\log_2(\delta) + \frac{n}{N}\log_2(q) \\
&= \sqrt{\frac{n\log_2(q)}{\log_2(\delta)}}\log_2(\delta) + \frac{n}{\sqrt{\frac{n\log_2(q)}{\log_2(\delta)}}}\log_2(q) \\
&= \sqrt{n\log_2(q)\cdot\log_2(\delta)} + \sqrt{n\log_2(q)\cdot\log_2(\delta)}. \\
&= 2\sqrt{n\log_2(q)\cdot\log_2(\delta)},
\end{aligned}$$

which can be rewritten as

$$n \geq \frac{\log_2(q\sigma)^2}{4\log_2(q)\log_2(\delta)}.$$

But we know that $\lambda = 1.8/\log_2(\delta) - 78.9$ so this inequality becomes

$$n \geq \frac{\log_2(q/\sigma)^2}{7.2\log_2(q)} \cdot (\lambda + 78.9).$$

This gives us the following table, where n is the smallest possible lattice rank, and B_χ is the bound on the magnitude of any error vector, which we calculated before in the standard-LWE case to be $B_\chi = (q/2-1)/(2\lceil(2n+1)\log_2(q)\rceil)$.

Table 4.1 provides minimum necessary parameters to satisfy each security level λ of the BGV encryption scheme. These values are based on the runtime of the BKZ algorithm to provide a basis that is reduced enough to solve the LWE Problem. These values consider the case where $\sigma^2 = 1$ for simplicity. If we increase the variance, this makes the error term more noisy, so we can use a smaller dimension n to achieve the same amount of security. This in turn increases the bound B_χ on the elements of the error vector, as the error vector has less elements due to the smaller dimension, so we can allow larger elements in the error vector without making the magnitude of the error vector too large. The problem with doing this is that even though B_χ increases, we will have less probability of choosing an error vector bounded by B_χ, because the variance of our error distribution increases. This makes it less likely that the error vector we choose will be small enough for decryption to work properly.

λ	$\log_2(q)$	n	B_χ	$\log_2(q)$	n	B_χ	$\log_2(q)$	n	B_χ
42	15	252	1	20	336	19	30	504	8868
52	16	291	1	20	364	17	30	546	8186
62	16	314	1	20	392	16	30	588	7602
72	16	336	1	20	420	15	30	629	7107
80	16	354	1	20	442	14	30	663	6742
112	16	425	1	20	531	12	30	796	5616
128	16	460	1	20	575	11	30	863	5181
192	17	640	1	20	753	8	30	1129	3960
256	17	791	1	20	931	7	30	1396	3203

Table 4.1: Some security parameters when χ has variance $\sigma^2 = 1$.

Chapter 5

Fully Homomorphic Encryption over the Integers

This chapter discusses another type of fully homomorphic encryption scheme based on integers rather than lattices. It overviews the initial scheme, an optimized version of the scheme, as well as a modified scheme that makes use of the techniques to remove bootstrapping to create another leveled fully homomorphic scheme.

The DGHV scheme discussed in this chapter provides an example of a truly fully homomorphic encryption scheme, rather than the leveled variety discussed in the previous chapter. It makes use of the bootstrappability definitions as defined in Section 3.3, and includes a technique called squashing that allows the scheme to be bootstrappable.

5.1 The Leftover Hash Lemma

In both the DGHV encryption scheme [23], and the optimized version proposed by Coron, Mandal, Naccache, and Tibouchi [5], the Leftover Hash Lemma is needed to ensure that the encryption scheme can be reduced to an underlying computationally hard problem. Due to the necessity of using this lemma, there are certain restrictions to choosing the parameters.

Lemma 5.1.1. (Simplified Leftover Hash Lemma) *Let $\mathcal{H}$ be a family of hash functions from X to Y that is 2-universal. Let $h \in \mathcal{H}$ and $x \in X$ be chosen uniformly and randomly. Then $(h, h(x))$ is $\frac{1}{2}\sqrt{|Y|/|X|}$-uniform over $\mathcal{H} \times Y$.*

5.2 DGHV Encryption Scheme

The following encryption scheme is a somewhat homomorphic encryption scheme by van Dijk, Gentry, Halevi, and Vaikuntanathan [23].

5.2.1 Parameters

Let λ be the security parameter, ξ be the bit-length of the integers in the public key, and let n be the number of integers in the public key. Let η be the bit-length of the secret key (where the secret key is the approximate-gcd of the public key integers), and let ρ be the bit-length of the noise (that is the maximum bit-length of the distance between the public key elements and their closest multiple of the secret key). We also need a parameter ρ' that is the bit-length of another noise variable.

Given a security level λ, we require that $\rho = \omega(\log_2(\lambda))$ to protect against brute force attacks on the noise. The bit-length of the secret key η needs to satisfy $\eta \geq \rho \cdot \Theta(\lambda(\log_2(\lambda))^2)$ so that the squashed decryption circuit can be evaluated homomorphically. The bit-length of the public key integers ξ needs to be $\omega(\eta^2 \log_2(\lambda))$ to prevent lattice attacks on the underlying Approximate-GCD Problem. The number of integers in the public key n must be larger than $\xi + \omega(\log_2(\lambda))$ so that we can use the leftover hash lemma in the reduction to the Approximate-GCD Problem. Also, we need $\rho' = \rho + \omega(\log_2(\lambda))$.

The paper by van Dijk, Gentry, Halevi, and Vaikuntanathan [23] suggests the parameters $\rho = \lambda$, $\rho' = 2\lambda$, $\eta = \tilde{O}(\lambda^2)$, $\xi = \tilde{O}(\lambda^5)$ and $n = \xi + \lambda$. This results in a scheme with public key of size $\tilde{O}(\lambda^{10})$.

Let $D_{\xi,\rho}(p) = \{$choose $q \in \mathbb{Z}$ uniformly from $[0, 2^\xi/p)$, and $r \in \mathbb{Z}$ uniformly from $(-2^\rho, 2^\rho)$, and output $x = pq + r\}$ be a probability distibution.

5.2.2 The Somewhat Homomorphic Encryption Scheme

KeyGen($\lambda \in \mathbb{Z}$):

Select an odd integer p with bit-length η to be the secret key sk.

For $n \in \mathbb{Z}$, sample x_i for $i = 0, \ldots, n$ from $D_{\xi,\rho}(p)$. Relabel the indices of the x_i so that x_0 is largest. If x_0 is even, or if $[x_0]_p$ is odd, then reselect the x_i's until this is not the case. Set the public key to be $pk = (x_0, x_1, \ldots, x_n) \in \mathbb{Z}^{n+1}$.

Encrypt$(pk \in \mathbb{Z}^{n+1}, m \in \{0,1\})$**:**

Select a random set $S \subseteq \{1, \ldots, n\}$ and a random integer $r \in (-2^{\rho'}, 2^{\rho'})$, and output the ciphertext $c = \left[m + 2r + 2\sum_{i \in S} x_i \right]_{x_0} \in \mathbb{Z}$.

Evaluate$(pk \in \mathbb{Z}^{n+1}, C, (c_1, \ldots, c_t) \in \mathbb{Z}^t)$**:**

If we are given a binary circuit C with t inputs, and t ciphertexts $c_1, \ldots, c_t$, apply the integer addition and multiplication gates of C on the ciphertexts, and output the resulting integer.

Decrypt$(sk \in \mathbb{Z}, c \in \mathbb{Z})$**:**

We have $sk = p$. Output the message $m = [c]_p \bmod 2$. Since $c \bmod p = c - p \cdot \lfloor c/p \rceil$, and p is odd, we can equivalently compute $m = [c]_2 \oplus [\lfloor c/p \rceil]_2$.

5.2.3 Security and Correctness of Circuit Evaluation

With use of the Leftover Hash Lemma, we can reduce the security of this decryption scheme to the Approximate GCD assumption. The following theorem is due to van Dijk, Gentry, Halevi, and Vaikuntanathan.

Theorem 5.2.1 ([23])**.** *Fix the parameters $(\rho, \rho', \eta, \xi, n)$ as defined in the previous encryption scheme, which are all polynomial in λ. Then, any attack $\mathcal{A}$ with advantage ϵ on the encryption scheme can be made into an algorithm $\mathcal{B}$ for solving the (ρ, η, n)-Approximate GCD Problem with probability of success at least $\epsilon/2$. The running time of $\mathcal{B}$ is polynomial in λ, $1/\epsilon$, and the running time of $\mathcal{A}$.*

In order to bootstrap this scheme, we need to have some sufficient condition on the input circuits for correct decryption. We get the following lemma.

Lemma 5.2.2 ([23])**.** *Let C be a boolean circuit with t inputs, and let C^* be the corresponding integer circuit where the boolean gates are replaced by gates that perform operations over the integers. If $f(x_1, \ldots, x_t)$ is the polynomial evaluated by C^*, such that f has degree d, then C is correct for our encryption scheme if $\|f\|_1 \cdot (2^{\rho'+2})^d \leq 2^{\eta-4}$, where $\|f\|_1$ is the l_1 norm of the coefficient vector of f.*

Thus, the DGHV encryption scheme can evaluate a polynomial f homomorphically if

$$d \leq \frac{\eta - 4 - \log_2(\|f\|)}{\rho' + 2}.$$

We will refer to any polynomial f that satisfies this inequality as a *permitted polynomial*, and we will denote the set of all such polynomials by $\mathcal{P}_{\mathcal{E}}$, where $\mathcal{E}$ is the encryption scheme. We will denote the set of circuits that evaluate such polynomials by $C(\mathcal{P}_{\mathcal{E}})$.

5.3 Bootstrapping the Scheme

In order for the DGHV to be fully homomorphic, we require that the scheme be correct for the set of augmented circuits $\mathcal{D}_{\mathcal{E}}$. However, if we represent the Decrypt function by some circuit, then the decryption circuit is too deep to evaluate homomorphically. Thus, we need to make use of a technique called *squashing*, which was proposed by Gentry [9], to *squash* the decryption circuit.

Squashing the decryption circuit means that we shrink the depth of the decryption circuit so that it can be evaluated homomorphically by the encryption scheme. To accomplish this, we need to modify the encryption scheme so that it provides some extra information in the public key about the corresponding secret key. Then, we can use this information to modify the ciphertext before evaluating it in the decryption circuit so that the decryption circuit doesn't have to do as much work, thereby making the decryption circuit smaller, or "squashed".

The following scheme is a modification of the original DGHV encryption scheme that adds extra information to the public key to make decryption easier. We also require that the added information provided in the public key does not give an attacker any advantage in breaking the cryptosystem. In this case, the added assumption is that the Sparse Subset Sum Problem (SSSP) is computationally infeasible.

In the original scheme, we were able to decrypt the ciphertext by computing $m = [c]_2 \oplus [\lfloor c/p \rceil]_2$. However, computing the value of $[\lfloor c/p \rceil]_2$ takes too many computations to evaluate this homomorphically, so this is the part of the decryption scheme that we want to squash. The issue arises because of the multiplication operations involved, so we want to replace it with additions.

To do this, we can compute c/p ahead of time. But in order for the secret key p to remain hidden, we can find some rational numbers y_j such that $\left[\sum_{j \in S} y_j\right]_2 = \frac{1}{p}$, for some

set S. In actuality, there is another term here, but it is so small that it becomes negligible. Thus, $[c/p]_2 = \left[\sum_{j \in S} c \cdot y_j\right]_2$. However, if we publish a large number of y_j terms, an attacker would not know which ones were used in the sum, which relies on the assumption that the SSSP is computationally hard.

5.3.1 The Modified Scheme

In the following section, we require three additional parameters κ, θ, and Θ. We will use $\kappa = \xi\eta/\rho'$, $\theta = \lambda$, and $\Theta = \omega(\kappa \cdot \log_2 \lambda)$.

BootKeyGen($\lambda \in \mathbb{Z}$):

Run the KeyGen(λ) function from the somewhat homomorphic scheme to get a secret key $sk^* = p$ and public key $pk^* = (x_0, x_1, \ldots, x_n)$. Let $x_p^* = \lfloor 2^\kappa/p \rceil$, and let $\mathbf{s} = (s_1, \ldots, s_\Theta)$ be a random vector with $s_i \in \mathbb{Z}_2,\ \forall i \in \{1, \ldots, \Theta\}$ such that the Hamming weight of $\mathbf{s}$ is θ. Let $S = \{i : s_i = 1\}$.

Select integers $u_i \in \mathbb{Z} \cap [0, 2^{\kappa+1})$ uniformly at random for $i \in \{1, \ldots, \Theta\}$ such that $\sum_{j \in S} u_j = x_p^* \pmod{2^{\kappa+1}}$. Let $\mathbf{y} = (y_1, \ldots, y_\Theta)$ where $y_i = u_i/2^\kappa$. So each y_i is a random number less than 2 with κ bits of precision after the binary point. We also get that $\left[\sum_{j \in S} y_j\right]_2 = \frac{1}{p} - \Delta_p$ for some $|\Delta_p| < 2^{-\kappa}$.

Output the secret key $sk = \mathbf{s}$ and the public key $pk = (pk^*, \mathbf{y})$.

BootEncrypt($pk, m \in \{0, 1\}$):

We can write $pk = (pk^*, \mathbf{y})$. Run Encrypt(pk^*, m) to get a ciphertext $c^* \in \mathbb{Z}$ as before. Let $\mathbf{z} = (z_1, \ldots, z_\Theta)$ where $z_i = [c^* \cdot y_i]_2$, keeping $\lceil \log_2 \theta \rceil + 3$ bits after the binary point for each z_i.

Output $c = (c^*, \mathbf{z})$ as the ciphertext.

BootEvaluate$(pk, C, (c_1, \ldots, c_t) \in \mathbb{Z}^t)$**:**

We can write $pk = (pk^*, \mathbf{y})$. Run Evaluate$(pk, C, (c_1, \ldots, c_t))$ to get a ciphertext c^* as before. Let $\mathbf{z} = (z_1, \ldots, z_\Theta)$ where $z_i = [c^* \cdot y_i]_2$, keeping $\lceil \log_2 \theta \rceil + 3$ bits after the binary point for each z_i.

Output $c = (c^*, \mathbf{z})$ as the ciphertext.

BootDecrypt$(sk \in \mathbb{Z}_2^\Theta, c \in \mathbb{Z} \times \mathbb{R}^\Theta)$**:**

We can write $c = (c^*, \mathbf{z})$, where $c^* \in \mathbb{Z}$ and $\mathbf{z} \in \mathbb{R}^\Theta$.

Output the message $m = \left[c^* - \left\lfloor \sum_{i=1}^{\Theta} s_i z_i \right\rceil \right]_2$.

5.3.2 Bootstrappability

Since we originally would compute m in the Decrypt function by $m = [c^*]_2 \oplus [\lfloor c^*/p \rceil]_2$, we get that the m computed by BootDecrypt is the same if and only if $[\lfloor c^*/p \rceil]_2 = \left[\left\lfloor \sum_{i=1}^{\Theta} s_i z_i \right\rceil \right]_2$; this result was proven by van Dijk et al. [23]. They also showed that the modified scheme is actually correct for $C(\mathcal{P}_\mathcal{E})$. We also get the following theorem.

Theorem 5.3.1. *If $\mathcal{E}$ is the modified encryption scheme above, and $\mathcal{D}_\mathcal{E}$ is the set of augmented squashed decryption circuits, then $\mathcal{D}_\mathcal{E} \subset C(\mathcal{P}_\mathcal{E})$.*

Thus, the modified encryption scheme is bootstrappable, which gives us a fully homomorphic encryption scheme.

5.4 An Optimization of the Bootstrappable Scheme

This section gives a variant of the DGHV encryption scheme, proposed by Coron, Mandal, Naccache, and Tibouchi [5]. The variant optimizes the original scheme by decreasing the size of the public key from $\tilde{O}(\lambda^{10})$ to $\tilde{O}(\lambda^7)$. This is accomplished by replacing the x_i's in the public key with $x'_{i,j} = x_{i,0}x_{j,1} \bmod x_0$ for $1 \leq i, j \leq \beta$ where β is a new parameter. Thus, only $2\beta + 1$ integers need to be stored in the public key to get $n = \beta^2$ integers used

for encryption. The public key is thereby reduced from $n\xi$ bits to about $2\sqrt{n}\xi$ bits. The key size is also reduced by taking linear combinations of the $x'_{i,j}$ with coefficients in $[0, 2^\alpha)$ instead of bits.

Similarly to the original scheme, we get the following constraints on the parameters. We require that $\rho = \omega(\log_2 \lambda)$, $\eta \geq (2\rho + \alpha) \cdot \Theta(\lambda(\log_2 \lambda)^2)$, $\xi = \omega(\eta^2 \log_2 \lambda)$, $\alpha \cdot \beta^2 \geq \xi + \omega(\log_2 \lambda)$, and $\rho' = 2\rho + \alpha + \omega(\log_2 \lambda)$. Concretely, we can set $\rho = \lambda$, $\eta = \tilde{O}(\lambda^2)$, $\xi = \tilde{O}(\lambda^5)$, $\alpha = \lambda$, $\beta = \tilde{O}(\lambda^2)$, and $\rho' = 4\lambda$.

5.4.1 The Somewhat Homomorphic Encryption Scheme

OptKeyGen($\lambda \in \mathbb{Z}$):

Generate a random prime $p \in [2^{\eta-1}, 2^\eta)$ (i.e. with bit-length η) to be the secret key sk.

Let q_0 be a random square-free 2^λ-rough integer in $[0, 2^\xi/p)$. Let $x_0 = q_0 \cdot p$. For $i \in \{1, \ldots, \beta\}$ and $b \in \{0, 1\}$, let $q_{i,b}$ be random integers in $[0, q_0)$ and let $r_{i,b}$ be random integers in $(-2^\rho, 2^\rho)$. Let $x_{i,b} = p \cdot q_{i,b} + r_{i,b} \in \mathbb{Z}$ for all $i \in \{1, \ldots, \beta\}, b \in \{0, 1\}$.

Let the secret key be $sk = p \in \mathbb{Z}$ and the public key be $pk = (x_0, x_{1,0}, x_{1,1}, \ldots, x_{\beta,0}, x_{\beta,1}) \in \mathbb{Z}^{2\beta+1}$.

OptEncrypt($pk \in \mathbb{Z}^{2\beta+1}, m \in \{0, 1\}$):

Select a random vector $\mathbf{b} = (b_{i,j})_{1 \leq i,j \leq \beta}$ of size $n = \beta^2$ with components in $[0, 2^\alpha)$, and select a random integer $r \in (-2^{\rho'}, 2^{\rho'})$, and output the ciphertext

$$c = \left[m + 2r + 2 \sum_{1 \leq i,j \leq \beta} b_{i,j} \cdot x_{i,0} \cdot x_{j,1} \right]_{x_0} \in \mathbb{Z}_{x_0}.$$

OptEvaluate($pk \in \mathbb{Z}^{2\beta+1}, C, (c_1, \ldots, c_t) \in \mathbb{Z}^t$):

If we are given a binary circuit C with t inputs, and t ciphertexts $c_1, \ldots, c_t$, apply the integer addition and multiplication gates of C on the ciphertexts, and output the resulting integer modulo x_0.

OptDecrypt($sk \in \mathbb{Z}, c \in \mathbb{Z}$)**:**

We have $sk = p$. Output the message $m = [c]_p \bmod 2$.

5.4.2 Security and Correctness of Circuit Evaluation

With use of the Leftover Hash Lemma, we can reduce the security of this decryption scheme to the Error-Free GCD assumption. The following theorem is due to Coron, Mandal, Naccache, and Tibouchi [5].

Theorem 5.4.1. *Fix the parameters $(\rho, \rho', \eta, \xi, n = \beta^2)$ as defined in the previous encryption scheme, which are all polynomial in λ. Then, any attack $\mathcal{A}$ with advantage ϵ on the encryption scheme can be made into an algorithm $\mathcal{B}$ for solving the (ρ, η, n)-Error-Free GCD Problem with probability of success at least $\epsilon/2$. The running time of $\mathcal{B}$ is polynomial in λ, $1/\epsilon$, and the running time of $\mathcal{A}$*[23].

In order to bootstrap this scheme, we need to have some sufficient condition on the input circuits for correct decryption. As in the original scheme, we make use of the idea of permitted polynomials. For this particular scheme, the permitted polynomials are the polynomials that satisfy

$$d \leq \frac{\eta - 3 - n - \log_2(\|f\|)}{\rho' + 2 + 2\log_2 \beta},$$

and these polynomials can be correctly evaluated homomorphically by the encryption scheme.

We will denote the set of such polynomials by $\mathcal{P}_{\mathcal{E}}$, where $\mathcal{E}$ is this encryption scheme. We will denote the set of circuits that evaluate such polynomials by $C(\mathcal{P}_{\mathcal{E}})$.

5.4.3 Bootstrapping the Scheme

As in the original DGHV scheme, it is necessary to squash the decryption circuit in order to homomorphically evaluate the augmented decryption circuits. The following scheme is a modification of the optimized DGHV encryption scheme that adds extra information to the public key and squashes the decryption function to make decryption easier. This incorporates the same ideas as the bootstrappable version of the original DGHV scheme by adding the y_i values to the public key. However, this is optimized by only storing a seed to a pseudo-random number generator rather than all of the y_i values, and regenerating them as needed.

OptBootKeyGen($\lambda \in \mathbb{Z}$):

Run the OptKeyGen(λ) function from the somewhat homomorphic scheme to get a secret key $sk^* = p \in \mathbb{Z}$ and public key $pk^* = (x_0, x_{1,0}, x_{1,1}, \ldots, x_{\beta,0}, x_{\beta,1}) \in \mathbb{Z}^{2\beta+1}$. Let $x_p^* = \lfloor 2^\kappa / p \rceil$, and let $\mathbf{s} = (s_1, \ldots, s_\Theta)$ be a random vector with $s_1 = 1$, and $s_i \in \mathbb{Z}_2$, $\forall i \in \{1, \ldots, \Theta\}$ such that the Hamming weight of $\mathbf{s}$ is θ. Let $S = \{i : s_i = 1\}$.

Let f be a pseudo-random number generator, and let $\mathbf{se}$ be a random seed for f. Then use $f(\mathbf{se})$ to generate integers $u_i \in \mathbb{Z} \cap [0, 2^{\kappa+1})$, for $i \in \{2, \ldots, \Theta\}$. Then set u_1 such that $\sum_{j \in S} u_j = x_p^* \pmod{2^{\kappa+1}}$. Let $\mathbf{y} = (y_1, \ldots, y_\Theta)$ where $y_i = u_i / 2^\kappa$. So each y_i is a random number less than 2 with κ bits of precision after the binary point. We also get that $\left[\sum_{j \in S} y_j\right]_2 = \frac{1}{p} - \Delta_p$ for some $|\Delta_p| < 2^{-\kappa}$.

Output the secret key $sk = \mathbf{s}$ and the public key $pk = (pk^*, \mathbf{se}, y_1)$.

OptBootEncrypt($pk, m \in \{0, 1\}$):

We can write $pk = (pk^*, \mathbf{se}, y_1)$. Run Encrypt$(pk^*, m)$ to get a ciphertext c^* as before. Let $\mathbf{z} = (z_1, \ldots, z_\Theta)$ where $z_i = [c^* \cdot y_i]_2$ keeping $\lceil \log_2(\theta + 1) \rceil$ bits after the binary point for each z_i.

Output $c = (c^*, \mathbf{z})$ as the ciphertext.

OptBootEvaluate($pk, C, (c_1, \ldots, c_t) \in \mathbb{Z}^t$):

We can write $pk = (pk^*, \mathbf{se}, y_1)$. Run Evaluate$(pk, C, (c_1, \ldots, c_t))$ to get a ciphertext c^* as before. Let $\mathbf{z} = (z_1, \ldots, z_\Theta)$ where $z_i = [c^* \cdot y_i]_2$ keeping $\lceil \log_2(\theta + 1) \rceil$ bits after the binary point for each z_i.

Output $c = (c^*, \mathbf{z})$ as the ciphertext.

OptBootDecrypt($sk \in \mathbb{Z}_2^\Theta, c \in \mathbb{Z} \times \mathbb{R}^\Theta$):

We can write $c = (c^*, \mathbf{z})$, where $c^* \in \mathbb{Z}$ and $\mathbf{z} \in \mathbb{R}^\Theta$.

Output the message $m = \left[c^* - \left\lfloor \sum_{i=1}^{\Theta} s_i z_i \right\rceil\right]_2$.

5.4.4 Bootstrappability

Similarly to van Dijk et al., Coron et al. showed that the modified scheme is correct for $C(\mathcal{P}_{\mathcal{E}})$, as well as the following theorem [5].

Theorem 5.4.2. *If $\mathcal{E}$ is the optimized encryption scheme above, and $\mathcal{D}_{\mathcal{E}}$ is the set of augmented squashed decryption circuits, then $\mathcal{D}_{\mathcal{E}} \subset C(\mathcal{P}_{\mathcal{E}})$.*

Thus, the modified optimized encryption scheme is bootstrappable, which gives us a fully homomorphic encryption scheme.

5.4.5 Security Parameters of the Optimized Scheme

Coron, Mandal, Naccache, and Tibouchi [5] gave some explicit parameter for a few small values of the security parameter. Their results are given in Table 5.1.

λ	ρ	η	ξ	β	Θ	pk size (MB)
42	16	1088	$1.6 \cdot 10^5$	12	144	0.95
52	24	1632	$8.6 \cdot 10^5$	23	533	9.6
62	32	2176	$4.2 \cdot 10^6$	44	1972	89
72	39	2652	$1.9 \cdot 10^7$	88	7897	802

Table 5.1: Parameters for different security levels provided by Coron, Mandal, Naccache, and Tibouchi[5].

Coron et al. [5] provide a number of restrictions on the parameters so that security levels are met.

A brute force attack on the noise can be mounted, taking time $2^{\rho} \cdot \tilde{O}(\xi)$, where the $\tilde{O}(\xi)$ term is the time needed to multiply two ξ-bit numbers together. For a security level λ, we require that $2^{\lambda} \leq 2^{\rho} \cdot \tilde{O}(\xi)$. We are given the following table of experimental data for the time taken to compute a single modular multiplication of two ξ-bit numbers.

ξ	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
time (s)	0.5	1.2	2.8	6.7	15.4
$\log_2$(clock cycles)	30.0	31.3	32.5	33.7	34.9

Table 5.2: Time to multiply two ξ-bit integers in Sage [5].

Since the multiplication operation takes time in $\tilde{O}(\xi)$, we can estimate a lower bound with a linear function. Say that when ξ doubles, then the number of clock cycles also doubles. However, with the given data, we see that an increase by 1 in the $\log_2(\xi)$ term corresponds to an increase of about 1.2 in the $\log_2$(clock cycles) parameter, so our function that only adds 1 to $\log_2$(clock cycles) should be a good lower bound. Concretely, say that $\xi = 2^9 \cdot$(number of clock cycles) for $\xi \geq 2^{21}$. Thus, we need the equation $\lambda \leq \rho + \log_2(\xi) + 9$ to be satisfied.

To deal with an Approximate-GCD attack on the public key, we need the parameters to satisfy

$$b \cdot \left(\frac{\eta - \sqrt{\eta^2 - 4\log_2(c)\xi}}{2\log_2(c)} \right)^{a-1} \cdot \xi \geq 2^{\lambda} \tag{5.1}$$

where $a,\ b,\ c$ are determined by the LLL and BKZ algorithms used in the attack. Specifically, $(a, b, c) = (5.0, 0.06, 1.021)$ for LLL, and $(a, b, c) = (5.2, 0.36, 1.013)$ for BKZ, where c is the average computed root-Hermite factor. The parameters of the encryption scheme have to satisfy inequality (5.1) for both sets of parameters a, b, c. However, since the LLL parameters will make the left hand side of (5.1) smaller than the BKZ parameters, the LLL parameters are the ones we should focus on.

When trying to determine what values of ξ and η work, it was suggested that if we are given η, we can then fix ξ so that (5.1) is satisfied [5]. However, this is impossible to do in some cases. We need to have $\Delta = \eta^2 - 4\log_2(c)\xi \geq 0$ so that the left hand side of (5.1) remains a real number, but we may need to increase ξ too much in order for this inequality to be satisfied, making Δ negative. Since we want the expression inside the brackets to be as large as possible, we want to minimize Δ, while keeping Δ greater than 0. So setting $\xi = \lfloor \eta^2/(4\log_2(c)) \rfloor$ accomplishes that goal. We can then increase η until this inequality is satisfied, which will simultaneously increase ξ.

To prevent a lattice attack on the Sparse Subset-Sum Problem, we can take [5]

$$\Theta = \frac{\eta - \sqrt{\eta^2 - 4\log_2(c)\xi}}{2\log_2(c)}. \tag{5.2}$$

Also, for recryption to work, we need that $d \cdot \rho < \eta$, where d is the degree of the decryption polynomial. For all of the security levels, we can take $d = 15$, but then with degree-2 compression of the secret key bits, this doubles the decryption polynomial degree to $d = 30$. To allow for an additional multiplication, the augmented decryption circuit has degree $d = 60$ [5]. So we take $\rho = \eta/(d + 8)$ to allow for some room as in the original paper [5]. Furthermore, to counter knapsack sum attacks in the encryption, we can set $\beta = \sqrt{\Theta}$ [5].

Finally, the public key has size roughly equal to $2(\beta + \sqrt{\Theta} + 1)\xi$ bits, and this fully determines our variables.

Thus, to determine our variables for a given security level λ, we first set $\xi = \lfloor \eta^2/(4 \log_2(c)) \rfloor$, and then find the smallest η satisfying (5.1) for both LLL and BKZ parameters. We then set Θ as in (5.2), and let $\beta = \sqrt{\Theta}$, $\rho = \eta/(d+8)$, giving a public key of size $2(\beta+\sqrt{\Theta}+1)\xi$ bits.

Our discussion yields the following parameters for the various security levels.

λ	ρ	η	ξ	β	Θ	pk size (MiB)
42	0	23	4410	20	379	$4.3 \cdot 10^{-2}$
52	1	71	$4.2 \cdot 10^4$	35	1181	$7.1 \cdot 10^{-1}$
62	3	223	$4.1 \cdot 10^5$	61	3717	12
72	10	705	$4.1 \cdot 10^6$	108	11751	215
80	26	1775	$2.6 \cdot 10^7$	172	29595	2167
112	1059	$7.2 \cdot 10^4$	$4.3 \cdot 10^{10}$	1096	$1.2 \cdot 10^6$	$3.6 \cdot 10^7$
128	6765	$4.6 \cdot 10^5$	$1.8 \cdot 10^{12}$	2770	$7.7 \cdot 10^6$	$3.7 \cdot 10^9$
192	$1.1 \cdot 10^7$	$7.4 \cdot 10^8$	$4.6 \cdot 10^{18}$	$1.1 \cdot 10^5$	$1.2 \cdot 10^{10}$	$3.9 \cdot 10^{17}$
256	$1.9 \cdot 10^{10}$	$1.3 \cdot 10^{12}$	$1.4 \cdot 10^{25}$	$4.7 \cdot 10^6$	$2.2 \cdot 10^{13}$	$5.0 \cdot 10^{25}$

Table 5.3: Parameters for different security levels ignoring a brute force attack on the noise.

Notice that these parameter choices don't incorporate the inequality $\lambda \leq \rho + \log_2(\xi) + 9$ to counter the brute force attack on the noise. This inequality is satisfied for $\lambda \geq 112$, as the corresponding ρ values are all larger than λ, but is not satisfied for any of the other values. To deal with this, we just need to increase ρ until the inequality holds, and change the other values accordingly. Notice that as ρ increases, so does η as we can set $\eta = (d + 8)\rho$, which in turn increases the value of $\xi = \lfloor \eta^2/(4 \log_2(c)) \rfloor$. Since (5.1) holds

for our original η and ξ values, increasing them will not cause any problems. Changing the values of β, Θ, and the public key size will also not cause problems because these are determined directly by the other variables.

We thus obtain the following parameters for each security level. Notice that each of these new ξ parameters are larger than 2^{21}, so we are allowed to use our approximation of the runtime to multiply integers of bit-length ξ.

λ	ρ	η	ξ	β	Θ	pk size (MiB)
42	11	748	$4.7 \cdot 10^6$	111	12468	249
52	20	1360	$1.5 \cdot 10^7$	150	22676	1113
62	29	1972	$3.2 \cdot 10^7$	181	32883	2817
72	38	2584	$5.6 \cdot 10^7$	207	43088	5529
80	45	3060	$7.8 \cdot 10^7$	225	51025	8430
112	1059	$7.2 \cdot 10^4$	$4.3 \cdot 10^{10}$	1096	$1.2 \cdot 10^6$	$3.6 \cdot 10^7$
128	6765	$4.6 \cdot 10^5$	$1.8 \cdot 10^{12}$	2770	$7.7 \cdot 10^6$	$3.7 \cdot 10^9$
192	$1.1 \cdot 10^7$	$7.4 \cdot 10^8$	$4.6 \cdot 10^{18}$	$1.1 \cdot 10^5$	$1.2 \cdot 10^{10}$	$3.9 \cdot 10^{17}$
256	$1.9 \cdot 10^{10}$	$1.3 \cdot 10^{12}$	$1.4 \cdot 10^{25}$	$4.7 \cdot 10^6$	$2.2 \cdot 10^{13}$	$5.0 \cdot 10^{25}$

Table 5.4: Parameters for different security levels when considering a brute force attack on the noise.

We can see from Table 5.4 that the public key size is extremely large for the security levels that are computationally infeasible, so the encryption scheme is impractical at these security levels. However, these parameters minimize the value of ρ, so we can increase ρ (and then increase η along with it), so we may be able to decrease the value of ξ. However, as mentioned before, we can't increase η by too much as this actually decreases the security level, so decreasing ξ as well would just make it worse.

5.5 The DGHV Encryption Scheme without Bootstrapping

The following encryption scheme is due to Coron, Naccache, and Tibouchi (CNT) [6], and is a hybrid of the DGHV encryption scheme over the integers and the BGV encryption scheme. It provides a leveled fully homomorphic encryption scheme that uses integers, but

uses the machinery discussed in the BGV encryption scheme to eliminate the need to use bootstrapping for any circuit of some predefined depth.

The CNT cryptosystem is semantically secure under the Decisional Approximate GCD assumption and under the subset sum assumption [6].

5.5.1 Some Important Functions

As in the BGV cryptosystem, the CNT cryptosystem needs to be able to homomorphically evaluate circuits of ciphertexts encrypted under different public keys, depending on what level the ciphertexts were last encrypted on. In order to deal with this, the scheme utilizes a key switching matrix (although is this case it is a vector). This section provides the functions needed to implement that.

BitDecomp($\mathbf{x} \in [0, 2^{k+1})^{\Theta}, k \in \mathbb{Z}$):

This function changes $\mathbf{x}$ into a binary representation of itself.

Write $\mathbf{x}$ as $\sum_{j=0}^{k} 2^j \mathbf{u}_j$ where $\mathbf{u}_j \in \mathbb{Z}_2^{\Theta}$, and output $\mathbf{u} = (\mathbf{u}_0^T, \mathbf{u}_1^T, \ldots, \mathbf{u}_k^T)^T \in \mathbb{Z}_2^{(k+1)\Theta}$.

Powersof2($\mathbf{x} \in [0, 2^{k+1})^{\Theta}, k \in \mathbb{Z}$):

This function outputs a vector that contains a number of the inputted vectors multiplied by various powers of 2, and in some sense is the reverse of the BitDecomp function. In fact, $\text{BitDecomp}(\mathbf{c}, q) \cdot \text{Powersof2}(\mathbf{s}, q) = \mathbf{c} \cdot \mathbf{s}$.

Output $\mathbf{u} = (\mathbf{x}^T, 2\mathbf{x}^T, \ldots, 2^k \mathbf{x}^T)^T \in \mathbb{Z}^{k\Theta}$.

SwitchKeyGen($pk \in \mathbb{Z}^{n+1}, sk \in \mathbb{Z}, pk' \in \mathbb{Z}^{n'+1}, sk' \in \mathbb{Z}$):

Let $p = sk$, and $p' = sk'$ be two DGHV secret-keys of bit-length η and η' respectively. Let $\kappa = 2\xi + \eta$, where ξ is the bit-length of the x_i's in the public key $pk = (x_0, \ldots, x_n)$.

For some $\Theta \in \mathbb{Z}$ where $\Theta > 0$, generate a vector $\mathbf{y}$ of Θ random numbers modulo $2^{\eta'+1}$ such that each number in $\mathbf{y}$ has κ bit of precision after the binary point, and a vector

$\mathbf{s}$ of Θ random bits that satisfies $2^{\eta'}/p = \mathbf{s} \cdot \mathbf{y} + \epsilon \bmod 2^{\eta'+1}$ where $|\epsilon| \leq 2^{-\kappa}$. Compute $\mathbf{s}' = \text{Powersof2}(\mathbf{s}, \eta')$.

If $pk' = (x_0, \ldots, x_{n'})$, then we can write $x'_0 = q'_0 \cdot p' + r'$ where $0 \leq r' < p'$. Then, sample $\mathbf{q}$ from $(\mathbb{Z} \cap [0, q'_0))^{(\eta'+1)\Theta}$ and sample $\mathbf{r}$ from $(\mathbb{Z} \cap (-2^{\rho'}, 2^{\rho'}))^{(\eta'+1)\Theta}$. Then let

$$\sigma = p' \cdot \mathbf{q} + \mathbf{r} + \left\lfloor \mathbf{s}' \cdot \frac{p'}{2^{\eta'+1}} \right\rceil.$$

Then output $\tau_{pk,pk'} = (\mathbf{y}^T, \sigma^T)^T \in \mathbb{R}^\Theta \times \mathbb{Z}^{(\eta'+1)\Theta}$.

SwitchKey($\tau_{pk,pk'} \in \mathbb{R}^\Theta \times \mathbb{Z}^{(\eta'+1)\Theta}, c \in \mathbb{Z}_{x_0}$):

We can write $\tau_{pk,pk'}$ as $(\mathbf{y}^T, \sigma^T)^T$ where $\mathbf{y}$ is a vector of Θ numbers and $\sigma \in \mathbb{Z}^{(\eta'+1)\Theta}$. Let $\mathbf{c} = (c_1, \ldots, c_\Theta)$ where $c_i = \lfloor c \cdot y_i \rceil \bmod 2^{\eta'+1}, \forall i \in \{1, \ldots, \Theta\}$. Let $\mathbf{c}' = \text{BitDecomp}(\mathbf{c}, \eta')$. Output $c'' = 2\sigma \cdot \mathbf{c}' + [c]_2$.

5.5.2 The Leveled Fully Homomorphic Scheme

In this variant of the DGHV encryption scheme, we define initially the depth of the circuits that we want to evaluate. The scheme allows us to compute arbitrarily large circuits, without needing the ability to be bootstrappable.

FHESetup($\lambda \in \mathbb{Z}, L \in \mathbb{Z}$):

The key generation function takes as its input the security parameter λ and the number of levels L that we wish to encrypt on (i.e. the number of addition or multiplication operations we are able to perform on encrypted data).

Let $\mu \in \mathbb{Z}$ be the length of the smallest secret key needed for the encryption scheme to be secure. Then, for $j \in \{1, \ldots, L\}$, define η_j to be $\eta_j = (j+1)\mu$. We take ρ, ξ, and n defined as in the basic DGHV encryption scheme.

FHEKeyGen($\lambda \in \mathbb{Z}, L \in \mathbb{Z}$):

For j from L to 1 do the following.

Run KeyGen(λ) using the bit-length of the secret key parameter η_j. Let $pk_j \in \mathbb{Z}^{n+1}$ be the public key generated, and $sk_j = p_j \in \mathbb{Z}$ be the secret key generated. When $j \neq L$, let the key switching vector be $\tau_{pk_{j+1},pk_j} = \text{SwitchKeyGen}(pk_{j+1}, sk_{j+1}, pk_j, sk_j) \in \mathbb{R}^{\Theta} \times \mathbb{Z}^{(\eta_j+1)\Theta}$.

We then set the public key pk to be pk_L along with the set of all τ_{pk_{j+1},pk_j}, and the secret key sk to be $sk = (p_1, \ldots, p_L) \in \mathbb{Z}$.

FHEEncrypt($pk, m \in \{0, 1\}$):

For a message m, output the ciphertext $c = \text{Encrypt}(pk_L, m) \in \mathbb{Z}$.

FHEDecrypt(sk, c):

For a ciphertext c, if c is encrypted under some pk_j in pk, output the message $m = [c]_{p_j} \bmod 2$, where $p_j = sk_j$.

FHERefresh($\tau_{pk_{j+1},pk_j} \in \mathbb{R}^{\Theta} \times \mathbb{Z}^{(\eta_j+1)\Theta}, c \in \mathbb{Z}$):

This function takes as its input a ciphertext c that is encrypted under pk_{j+1} in the public key, and the key switching vector τ_{pk_{j+1},pk_j}. Output $c' = \text{SwitchKey}(\tau_{pk_{j+1},pk_j}, c)$, the new ciphertext encrypted under pk_j.

FHEAdd($pk, c_1 \in \mathbb{Z}, c_2 \in \mathbb{Z}$):

This function requires that c_1 and c_2 are encrypted under the same pk_j in the public key. If they are not, we can apply the FHERefresh step to the ciphertext encrypted with the pk_j further up the ladder of decreasing moduli. Say c_1 is encrypted with pk_i, and c_2 is encrypted with pk_k, where $i > k$. We then use FHERefresh on c_1 (i.e. FHERefresh($\tau_{pk_i,pk_{i-1}}$, c_1)) to get a ciphertext encrypted with pk_{i-1}, and we can continue this process until c_1 and c_2 are encrypted with the same pk_j.

Let $c_3 = c_1 + c_2$. If $j \neq 1$, output $c_4 = \text{FHERefresh}(\tau_{pk_j,pk_{j-1}}, c_3)$, otherwise output c_3.

FHEMult$(pk, c_1 \in \mathbb{Z}, c_2 \in \mathbb{Z})$**:**

This function requires that c_1 and c_2 are encrypted under the same pk_j in the public key. If they are not, we can apply the FHERefresh step to the ciphertext encrypted with the pk_j further up the ladder of decreasing moduli. Say c_1 is encrypted with pk_i, and c_2 is encrypted with pk_k, where $i > k$. We then use FHERefresh on c_1 (i.e. FHERefresh$(\tau_{pk_i,pk_{i-1}}, c_1)$) to get a ciphertext encrypted with pk_{i-1}, and we can continue this process until c_1 and c_2 are encrypted with the same pk_j.

Let $c_3 = c_1 \cdot c_2$. If $j \neq 1$, output $c_4 = \text{FHERefresh}(\tau_{pk_j,pk_{j-1}}, c_3)$, otherwise output c_3.

Chapter 6

Other Cryptosystems and Results

This chapter overviews a few other homomorphic encryption schemes. Gentry's original scheme is mentioned, as well as others based on more recent work.

6.1 Gentry's Scheme

The first fully homomorphic encryption scheme was created by Gentry, and uses ideal lattices. Gentry's scheme is similar to the GGH encryption scheme, except it is implemented over ideal lattices to make use of the multiplication ability. In this encryption scheme, we work over some ring $R = \mathbb{Z}[x]/(x^n + 1)$, and take two ideals of R, J and I. For the public key of this scheme, we use a bad (highly nonorthogonal basis) for J. The second ideal I needs to have a small basis, such as the principal ideal $I = (2)$.

In Gentry's scheme, we use ciphertexts that are close to a lattice point in J, where the message is in the distance of the ciphertext to a specific lattice point. Specifically, if the plaintext space is $\{0, 1\}$, then we can embed this into R/I (when $I = (2)$ we can do this by encoding 0 as the vector 0^n, and 1 as the vector 10^{n-1}), and set the error term to be $e = i \cdot r + m$ where $m \in R/I$ is the encoded bit, $r \in R$ is a random small vector, and $i \in I$ is a small vector in the basis of I (for $I = (2)$, $i = 2$). Multiplication is defined over R, and not some form of vector multiplication. Then we get that the ciphertext is $c = e + j$ for some $j \in J$.

To decrypt a ciphertext, we use a short vector $w \in J^{-1}$ as the private key, and compute $[w \cdot c]$, the fractional part of $w \cdot c$. Since $c = e + j$, $w \cdot c = w \cdot e + w \cdot j$, and since $w \cdot j$ is an integer vector (as $w \in J^{-1}$), we get that $[w \cdot c] = [w \cdot e]$. As long as w and e are

small enough, $[w \cdot e]$ will have magnitude less than $1/2$, which would mean $[w \cdot e] = w \cdot e$. So this gives us that $w \cdot c = w \cdot e$, which allows us to compute e by multiplying by the inverse of w. Then $m = e \bmod I$. However, in the actual scheme, w is adjusted so that $m = [w \cdot c] \bmod I$.

As mentioned in the description of the DGHV scheme, Gentry's scheme was the first to suggest the process of squashing the decryption circuit by adding information about the secret key to the ciphertexts so the decryption circuit does not have to do as much work. Similarly to the DGHV scheme, Gentry's scheme uses a large set of ring elements $S = \{x_i \mid i = 1, \ldots, m\}$, where a hidden sparse subset T adds up to w, and the new secret key becomes the characteristic vector $(\sigma_1, \ldots, \sigma_m)$ of the sparse subset T.

Then, by computing the products $y_i = x_i \cdot c$ for each element $x_1 \in S$, all one needs to do is compute $c - \left[\sum_{j=1}^{m} \sigma_j \cdot y_j\right] \bmod I$, which is equivalent to $m = c - [w \cdot c] \bmod I$ [10].

6.1.1 Optimizations of Gentry's Scheme

Since Gentry's initial creation of a fully homomorphic encryption scheme, others have attempted to implement Gentry's scheme and optimize it. The first attempt to do so was by Smart and Vercauteren in 2010 [21]. In the Smart-Vercauteren variant, principal-ideal lattices of prime determinant are used. The benefit of these types of lattices is that they can be determined using just two integers, so this decreases the key sizes, as entire bases do not need to be listed.

Smart and Vercauteren work over the ring $R = \mathbb{Z}[x]/(f(x))$ where $f(x) = x^{2^n} + 1$, and use the principal ideal $J = (v)$, the ideal generated by $v \in R$, such that the determinant of this ideal p is prime. They also use $I = (2)$. They also need to know a root r of $f(x) \bmod p$. They are then able to output a ciphertext c of the message m by $c = [2 \cdot u(r) + m]_p$, where $u(x)$ is a small random polynomial, and decrypt by computing $m = c - \lfloor cw/p \rceil \bmod 2$, where the secret key w is a single integer determined in such a way to make this work.

The following table shows some of the parameters for the Smart-Vercauteren encryption scheme. The two sets of $\lambda, s_2,$ and d arise from other parameter choices in the implementation of the scheme. The value of s_2 is chosen so that $\sqrt{\left(\frac{\log_2 p}{s_2}\right)} > 2^\lambda$, and d is the depth of circuits that can be evaluated given the other parameters. and $\hat{d}$ is the required circuit depth of circuits that can be evaluated homomorphically to allow for bootstrapping. As always, we use λ to represent the security level of the scheme.

n	$\log_2 p$	λ	s_2	d	λ	s_2	d
8	4096	25	5	0.3	36	8	0.0
9	11585	31	6	0.8	40	7	0.3
10	32768	41	7	1.2	48	8	0.8
11	92681	54	8	1.7	61	9	1.2
12	262144	73	10	2.1	80	11	1.6
13	741455	100	12	2.5	107	13	2.1

Table 6.1: Smart and Vercauteren's encryption scheme parameters [21].

The following table gives the required depth $\hat{d}$ of circuits needed to perform bootstrapping, for given values of s_2.

s_2	6	7	8	9	10	11	12	13
$\hat{d}$	7	7	7	8	8	8	8	8

Table 6.2: Required circuit depth for bootstrapping [21].

Unfortunately, none of the d-values in Table 6.1 match the $\hat{d}$ values needed for fully homomorphic encryption. However, Smart and Vercauteren showed that for $n \geq 27$, it is possible to obtain a fully homomorphic encryption scheme. However, considering the public key contains p, and p is 90.5 KB long when $n = 13$, increasing n to 27 will make the key sizes much too large in practice. Furthermore, they were unable to create keys when $n = 12$ because the public key creation would take too much time.

6.2 Other Results

6.2.1 Ideal Lattices in the BGV Encryption Scheme

In the BGV encryption scheme, an optimization was made to increase the size of the message space when R_2 was replaced with R_p. However, if we want to have a plaintext space that is exponential in p, we want to use ideal lattices. Since the noise is proportional to p after the modulus switching step [2], the original method doesn't work.

Instead of having a ladder of decreasing moduli in the leveled fully homomorphic scheme, we replace this by a ladder of principal ideals that are congruent to 1 mod I,

where I is a ideal lattice with a short basis, and whose norm is a large prime. However, having a ladder of ideals instead of integers requires a generalization of the modulus switching technique, and a bound on the noise after modulus switching. The proof given that deals with a bound on the error after modulus switching has the same issues as the other lemmas that were fixed earlier in this paper [2].

6.2.2 Encryption Schemes with Constant Degree Decryption Functions Cannot Be Homomorphic

In the paper "When Homomorphism Becomes a Liability" [1], Brakerski was able to show that encryption schemes with certain properties cannot be homomorphic. Namely, any encryption scheme that uses a decryption function that can be represented as a polynomial of fixed degree with respect to the elements in the secret key and ciphertext cannot be fully homomorphic, if the probability of an incorrect decryption is less than $1/2 - 1/\text{poly}(n)$. Thus, if the decryption function of an encryption scheme is too simple, then the scheme can not be fully homomorphic without first changing the decryption function.

This conclusion comes from the result that if we have a decryption function of the form $\mathbf{s} \cdot \mathbf{c}$, where $\mathbf{s} \in \mathbb{F}^n$ is the secret key and $\mathbf{c}$ is the ciphertext, and where $\mathbb{F}$ is some field, then the encryption scheme cannot compute the majority function of $\text{O}(\log n/\epsilon^2)$ elements homomorphically if the decryption scheme incorrectly decrypts with probability less than $(1/2 - \epsilon)$.

Since an encryption scheme that has a decryption function of the form $\mathbf{s} \cdot \mathbf{c}$ is unable to evaluate the majority function, it is not able to evaluate all circuits homomorphically, and as such cannot be fully homomorphic. Also, we can represent any polynomial function of fixed degree in the form $\mathbf{s} \cdot \mathbf{c}$, as any polynomial in the elements of $\mathbf{s}$ and $\mathbf{c}$ can be written as a sum of monomials, where each monomial is the product of terms in $\mathbf{s}$ and terms in $\mathbf{c}$. If we take all of the $\mathbf{s}$ terms as a single element for each monomial, and likewise for $\mathbf{c}$, this gives us an inner product that has length equal to the number of monomials.

6.2.3 Homomorphic Evaluation of AES

Recently, there have been implementations of fully homomorphic encryption schemes. Smart and Vercauteren were able to use Gentry's original scheme [8][10], and their specialization of the scheme [21], to adapt the parameter choices to allow the use of SIMD (Single-Instruction Multiple-Data) operations in homomorphic encryption schemes [22].

This allows each ciphertext to represent a number of independent plaintexts, and as such, decrease the amount of space needed to store data.

The Smart-Vercauteren modifications give a scheme that is able to perform homomorphic operations on large finite fields of characteristic 2, and can perform SIMD operations. Smart and Vercauteren discuss how these SIMD operations can be used to do parallel computations in the bootstrapping operations, as well as how these operations can be used to evaluate the AES circuit homomorphically.

Using this work, as well as some prior optimizations [12], Gentry, Halevi and Smart were able to implement a fully homomorphic encryption scheme that is able to evaluate an AES-128 circuit [13]. The encryption scheme that they used is a variant of the Ring Learning with Errors BGV encryption scheme discussed in Section 4.1.3.

When evaluating the AES circuit, Gentry et al. used the NTL C++ Library, and ran their implementation on a machine with a 2.0 GHz processor with 18MB of cache, and that contained 256 GB of RAM. Even with this large amount of memory, they needed to choose an implementation that minimized the amount of memory used, as memory was still the limiting factor. The machine was able to compute a single block of AES encryption in about eight days, which is two orders of magnitude faster than if they had used an optimized version of Gentry's scheme. However, since each ciphertext can hold 1512 plaintext slots of $\mathbb{F}_{2^8}$, 94 AES blocks can be processed in this time period in parallel, giving an average of about two hours per AES block [13].

Chapter 7

Conclusion

Since Craig Gentry's breakthrough three years ago, there has been significant progress in the area of fully homomorphic encryption. There have been a few different types of encryption schemes proposed since then, from Gentry's original scheme using ideal lattices, to the DGHV fully homomorphic encryption scheme over the integers, to the BGV encryption scheme using the Learning With Errors Problem that is an example of a leveled fully homomorphic encryption scheme. There has also been a lot of work on optimizations to preexisting work, in both computation time as well as in decreasing memory requirements.

Currently, the amount of memory required for any of the fully homomorphic or leveled fully homomorphic encryption schemes is too large to be useful in practice; however, the optimizations that have been proposed have allowed for the creation of an implementation of fully homomorphic encryption for evaluating an AES-128 circuit. This implementation may require a huge amount of memory and take days to evaluate, but at least exists when only a few years ago this was considered unlikely.

As new results are found to decrease the key size and decrease the running time, new attacks are being found on the lattice problems. This helps give a better bound on what the security level of these fully homomorphic encryption schemes actually is for different sets of parameters. However, new and improved results in this area also shrink the security levels of previous encryption schemes, requiring larger keys to achieve the same level of security.

Ever since Rivest et al. proposed the idea of fully homomorphic encryption back in 1978 and explained its potential applications, the importance of fully homomorphic encryption has been widely appreciated. In fact, a fully homomorphic public key encryption scheme

has been referred to as a "holy grail" of cryptography by some [21]. As such, it is important to continue efforts in this area to try and find a truly practical encryption scheme.

In the CRYPTO 2011 talk given by Mehdi Tibouchi [5], it is mentioned that attacks by Chen and Nguyen [3] and by Cohn and Heninger [4] can be mounted on the DGHV encryption scheme. These new attacks require that larger parameters be chosen to achieve the same level of security. Calculating the new parameters to take into account these two attacks remains to be done.

References

[1] Zvika Brakerski. When homomorphism becomes a liability. Cryptology ePrint Archive Report 2012/225. http://eprint.iacr.org/2012/225/.

[2] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:111, 2011.

[3] Yuanmi Chen and Phong Q. Nguyen. Faster algorithms for approximate common divisors: Breaking fully-homomorphic encryption challenges over the integers. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 502–519. Springer-Verlag Berlin/Heidelberg, 2012.

[4] Henry Cohn and Nadia Heninger. Approximate common divisors via lattices. *CoRR*, abs/1108.2714, 2011.

[5] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *Advances in Cryptology – CRYPTO'11*, pages 487–504. Springer-Verlag Berlin/Heidelberg, 2011.

[6] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Advances in Cryptology – EUROCRYPT'12*, pages 446–464. Springer-Verlag Berlin/Heidelberg, 2012.

[7] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51. Springer-Verlag Berlin/Heidelberg, 2008.

[8] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[9] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[10] Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *Advances in Cryptology EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer-Verlag Berlin/Heidelberg, 2011.

[11] Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P. Smart. Ring switching in BGV-style homomorphic encryption. In *SCN*, volume 7485 of *Lecture Notes in Computer Science*, pages 19–37. Springer-Verlag Berlin/Heidelberg, 2012.

[12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Advances in Cryptology – EUROCRYPT'12*, pages 465–482. Springer-Verlag Berlin/Heidelberg, 2012.

[13] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer-Verlag Berlin/Heidelberg, 2012.

[14] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *An Introduction to Mathematical Cryptography*, chapter 6 (Lattices and Cryptography). Springer, 2008.

[15] Thijs Laarhoven, Joop van de Pol, and Benne de Weger. Solving hard lattice problems and the security of lattice-based cryptosystems. Cryptology ePrint Archive, Report 2012/533, 2012. http://eprint.iacr.org/2012/533/.

[16] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In *Topics in Cryptology: CT-RSA 2011*, CT-RSA'11, pages 319–339. Springer-Verlag Berlin/Heidelberg, 2011.

[17] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer-Verlag Berlin/Heidelberg, 2010.

[18] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *Proceedings of the 41st annual ACM Symposium on Theory of Computing*, STOC '09, pages 333–342, New York, NY, USA, 2009. ACM.

[19] Ron Rivest, Len Adleman, and Michael Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–180, 1978.

[20] Markus Rückert and Michael Schneider. Estimating the security of lattice-based cryptosystems. Cryptology ePrint Archive Report 2010/137. http://eprint.iacr.org/2010/137/.

[21] N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Practice and Theory in Public Key Cryptography – PKC'10*, pages 420–443. Springer-Verlag Berlin/Heidelberg, 2010.

[22] N.P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. Cryptology ePrint Archive, Report 2011/133, 2011. http://eprint.iacr.org/2011/133/.

[23] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer-Verlag Berlin/Heidelberg, 2010.