# Parameter Constraints on Homomorphic Encryption Over the Integers

Melanie Pabstel

Thesis submitted to the Faculty of Graduate and Postdoctoral Studies in partial fulfillment of the requirements for the degree of
Master of Science in Mathematics[1]

Department of Mathematics and Statistics
Faculty of Science
University of Ottawa



[1]The M.Sc. program is a joint program with Carleton University, administered by the Ottawa-Carleton Institute of Mathematics and Statistics

# Abstract

The research paper *Fully Homomorphic Encryption over the Integers* by van Dijk, Gentry, Halevi, and Vaikuntanathan [31] explores the construction of an encryption scheme over the integers that is fully homomorphic, using modular arithmetic. The plaintext messages in this encryption are single bits and the ciphertexts are large integers. The homomorphic property means that the algebraic operations on the plaintexts can be carried out analogously on the ciphertexts. We analyze in detail the parameter constraints required to make the scheme functional and secure, prove auxiliary results about noise accumulation, and generate a toy example to concretely illustrate parts of the scheme.

# Résumé

L'article de recherche intitulé *Fully Homomorphic Encryption over the Integers* par van Dijk, Gentry, Halevi et Vaikuntanathan [31] démontre la construction d'un schéma de chiffrement totalement homomorphe dont la méthode de chiffrement utilise l'arithmétique modulaire. Les messages sont des bits qui, lorsque chiffrés, deviennent des nombres entiers très grands. La propriété homomorphe du schéma indique que toute fonction pouvant être évaluée sur les messages a une fonction correspondante qui peut être évaluée sur les messages chiffrés. Nous analysons en détails les contraintes nécessaires sur les paramètres du schéma afin que celui-ci soit fonctionnel et sécuritaire, nous offrons des preuves de résultats auxiliaires concernant l'accumulation d'erreur sur les messages chiffrés, et nous générons un exemple de petite taille afin d'illustrer concrètement certaines parties du schéma.

# Dedication

To my family, who have been encouraging and supporting me every step of the way.

# Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Monica Nevins, for her continued, unwavering support throughout my entire degree. She was always available to answer my questions and guide me through the writing process, she knew what questions to ask to explore new avenues of research, and her clear explanations helped me understand a very computer-science heavy topic in a mathematical way. Her endless enthusiasm is contagious and she always knew just what to say to renew my motivation to accomplish my thesis. Thank you so very much.

I would also like to thank Jason Levy for co-supervising me in the first eight months of my Masters study, introducing me to the topic of homomorphic encryption and providing valuable insight and tools to be able to create code to help me understand how the scheme works.

Thank you to Philippe, my better half, for his support, patience, and for helping me push to achieve my goals. I couldn't have done it without you. Thank you also to my family and friends for their continuous encouragement, helping me keep my head up when times got tough.

Finally, I would like to thank the University of Ottawa for their financial support of this research, and the faculty and staff in the Department of Mathematics and Statistics who have always been available and welcoming.

# Contents

**List of Figures** viii

**List of Tables** ix

**1 Introduction** 1

**2 The DGHV Scheme** 5
- 2.1 Overview of the Scheme . . . 5
  - 2.1.1 Key generation . . . 5
  - 2.1.2 Decryption of a ciphertext . . . 6
  - 2.1.3 Encryption of a plaintext . . . 6
- 2.2 Toy Numerical Example . . . 9
- 2.3 Parameters and Algorithms . . . 11
- 2.4 Constraints on Parameters . . . 13
  - 2.4.1 Functional constraints . . . 14
  - 2.4.2 Security: the subset-sum problem? . . . 16
  - 2.4.3 Security: the approximate gcd problem . . . 18
  - 2.4.4 Security: the secondary noise parameter . . . 20
  - 2.4.5 Summary . . . 23

**3 Homomorphic Aspects of the DGHV Scheme** 26
- 3.1 Overview of Homomorphic Encryption . . . 26
- 3.2 A First Attempt at SHE with the DGHV Scheme . . . 28
  - 3.2.1 Numerical example continued . . . 30
- 3.3 Optimization – A Successful Attempt at SHE . . . 31
  - 3.3.1 Optimization algorithm . . . 31
  - 3.3.2 Numerical example of optimization . . . 33
  - 3.3.3 Depth-one homomorphic DGHV . . . 34
  - 3.3.4 Proof of the optimization noise accumulation theorem . . . 37
- 3.4 Depth-$d$ Homomorphic Encryption . . . 46

**4 A Fully Homomorphic DGHV Scheme** **51**
4.1 From Somewhat Homomorphic to Fully Homomorphic . . . . . 52
4.2 Bootstrappability Implementation Problems . . . . . 54
4.3 Squashing the Private Key . . . . . 55
4.3.1 Parameter constraints of the squashed key: functionality . . 57
4.3.2 Parameter constraints of the squashed key: security . . . . . 59
4.4 The Squashed Decryption Circuit . . . . . 60
4.4.1 Explicit computation of the decryption function . . . . . 61
4.4.2 Encryption of the decryption circuit . . . . . 62
4.5 Example Adder: the Ripple-Carry Adder . . . . . 63
4.5.1 Total noise on the circuit: adding two numbers . . . . . 67
4.5.2 Total noise on the circuit: adding $\Theta$ numbers . . . . . 68
4.5.3 Further analysis of the addition function . . . . . 69
4.6 Comparing Parameters . . . . . 70

**A Matlab Code** **73**
A.1 KeyGen . . . . . 73
A.2 Encrypt . . . . . 74
A.3 Decrypt . . . . . 74
A.4 Cmod . . . . . 74
A.5 Hom_Enc_Z . . . . . 75
A.6 Error_calc . . . . . 75
A.7 Evaluate . . . . . 76
A.8 Optimize . . . . . 76
A.9 Other Scripts and Functions . . . . . 76

# List of Figures

2.1 Distribution of subset sums without additional security parameter . 22
2.2 Distribution of subset sums with additional security parameter . . . 23

4.1 Circuit representation of a full adder . . . . . 64
4.2 Simplified full adder circuit . . . . . 65
4.3 Circuit representation of the ripple-carry adder . . . . . 65

# List of Tables

2.1 List of parameters for the DGHV scheme . . . . . 12
2.2 Basic parameter constraints on the DGHV scheme . . . . . 24

4.1 List of new parameters for the squashed DGHV scheme . . . . . 55
4.2 All our parameter constraints for the DGHV scheme . . . . . 70
4.3 Parameter constraints of the DGHV scheme by its authors . . . . . 71
4.4 The "ideal" parameters suggested by the DGHV authors . . . . . 71

# Chapter 1

# Introduction

Cryptography has long held an important place in societies all around the world. From the Caesar cipher to the Enigma machines to AES to RSA and beyond, where there is a secret to be kept, cryptography is present. Since the advent of computers, research in cryptography has grown rapidly in prominence and complexity. In this thesis we will discuss once particular facet of cryptography, that of homomorphic encryption.

A number of basic concepts must be understood in order to be able to discuss topics in cryptography. We'll talk about *encryption* as the procedure by which we obscure data such that only the intended recipient may view it. We call raw, unencrypted data a *plaintext* and encrypted data a *ciphertext*. We then call *decryption* the process by which the intended (or other) recipient transforms the ciphertext back into its corresponding plaintext.

There exist two types of encryption schemes, or cryptosystems: symmetric-key and public-key (also called asymmetric-key). A *symmetric-key cryptosystem* is one where both the sender and intended recipient, commonly referred to as Alice and Bob, share a secret key in a secure manner, such that no one else has access to it. Alice then encrypts her data using their shared secret key and sends it to Bob, who can decrypt it using the same secret key.

While symmetric-key encryption schemes contain many useful properties, their main caveat lies in the necessity of sharing the secret key in a secure manner. This all changed in 1976 when Whitfield Diffie and Martin E. Hellman published *New Directions in Cryptography* [11]. Diffie and Hellman's paper completely changed the way people thought about cryptography by introducing the concept of a *public-key cryptosystem*. In this type of scheme, there is no longer a secret key shared between Alice and Bob, but rather a *private key* that Alice keeps to herself. She then creates a *public key* that is related in some manner to her private key and publishes her public key so that anyone may see it. If Bob wants to send sensitive data to Alice, he simply encrypts it using Alice's public key and sends it to her. Upon receiving Bob's

encrypted data, Alice uses her private key to decrypt it and recover the plaintext. Diffie and Hellman not only proposed the concept in their paper, but also provided an example scheme that would accomplish it.

Following this revolutionary invention, research in cryptography turned towards creating more public-key encryption schemes and testing their security. In 1977, RSA was born, and many other schemes have since followed. Another aspect of research on public-key encryption revolves around determining desirable properties for a scheme and creating schemes that implement these properties. One of those desirable properties is called *homomorphic encryption*.

Homomorphic encryption is a form of public-key encryption such that functions evaluated on ciphertexts generate a result which, when decrypted, corresponds to the equivalent functions evaluated on the plaintexts. Visually, homomorphic encryption may be represented by the following diagram.

$$\begin{array}{ccc} (\mathtt{m}_1, \ldots, \mathtt{m}_n) & \xrightarrow{f^n} & \mathtt{a} \\ {\scriptstyle Enc,\ldots,Enc}\downarrow & & \uparrow{\scriptstyle Dec} \\ (\mathrm{C}_1, \ldots, \mathrm{C}_n) & \xrightarrow[Enc(f^n)]{} & \mathrm{A} \end{array}$$

Homomorphic encryption is a classic problem in cryptography. No sooner had RSA been invented that two of its authors, Ronald Rivest and Len Adleman, along with Michael Dertouzos, were already thinking about the idea of homomorphic encryption. Their 1978 paper titled *On Data Banks and Privacy Homomorphisms* [28] contained their musings on special encryption functions "which permit encrypted data to be operated on without preliminary decryption of the operands," which they called "privacy homomorphisms." They laid down the groundwork for a theoretical homomorphic encryption scheme, detailing certain conditions necessary for the concept to work. They also provided some simple, though insecure, examples of homomorphic schemes, to demonstrate the validity of the concept. However, no secure homomorphic encryption scheme of any kind was presented for the next three decades.

In 2009, Craig Gentry published his PhD thesis, titled *A Fully Homomorphic Encryption Scheme* [14]. In it, he put forward the first fully homomorphic encryption scheme, bridging a 31-year gap between a theoretical concept and a physical scheme. Gentry's scheme's construction and security rely on hard problems on ideal lattices (as summarized in [16]). The following year, Gentry teamed up with Marten van Dijk, Shai Halevi, and Vinod Vaikuntanathan to create another homomorphic encryption scheme, this time over the integers [31]. The main goal of their paper was to demonstrate homomorphic encryption in a conceptually simpler manner. This scheme has since been referred to as the *DGHV scheme* and is the main object of our work.

This thesis will examine how homomorphic encryption over the integers, as presented by van Dijk et al., works as a scheme. In Chapter 2, we discuss the DGHV

scheme in its most basic form, providing an overview of the scheme, detailing the parameters and algorithms that go into its construction, and examining the first constraints on the parameters in terms of functionality and security. We also provide a toy example to demonstrate the scheme. Our main contributions in this chapter are to develop a toy example to concretely illustrate the scheme and to provide more detailed analysis on the accumulation and distribution of noise.

In Chapter 3, we explore the homomorphic capabilities of the scheme, focusing on the *depth* of the scheme, that is, how many operations in a row it can perform. We define homomorphic encryption generally, then examine the changes van Dijk et al. make to their scheme to make it homomorphic. We analyze the consequent new constraints on the parameters, including an upper bound on the depth of the scheme as it is defined. Our contributions consist of providing a continuation of the toy example as well as a more detailed analysis of the optimization algorithm and its implications on the noise accumulation.

Finally, in Chapter 4, we explore the modifications the DGHV authors make to their scheme in order to attain unlimited depth. We explain Gentry's concept of *bootstrappability* and how this can be used to obtain a fully homomorphic encryption scheme. We offer an alternative explanation to parts of their modifications, and compare them to the paper. Our contributions are to offer an alternative interpretation of the modified scheme, including a different example of a function used to implement it and its associated analysis.

Throughout the thesis, all logarithms are in base 2 unless otherwise specified. We have used Greek letters to denote the various parameters of the scheme, the `texttt` font to denote any variables that are bits, and the TEXTSC font to denote any variables that are encrypted in the DGHV scheme. For a value $x \in \mathbb{Q}$, we use $\lceil x \rfloor$ to mean $x$ rounded to the nearest integer, that is, $\lceil x \rfloor$ corresponds to the integer in the interval $\left(x - \frac{1}{2}, x + \frac{1}{2}\right]$.

We used Matlab to implement parts of the algorithm, based on the written description in [31], in order to better understand it and to produce the toy example described throughout the thesis. Appendix A contains descriptions of various scripts we created to replicate parts of the DGHV scheme.

In November 2015, the National Institute of Standards and Technology (NIST) in the United States released a report issuing recommendations in regards to cryptographic algorithms and key lengths [3]. They state in the report that "for the Federal government, a minimum security strength of 112 bits is required for applying cryptographic protection." Keeping this number in mind throughout the thesis, we will see that the DGHV scheme, while extremely useful in demonstrating how homomorphic encryption may be implemented as a scheme, requires some more optimization before it can become a practical fully homomorphic encryption scheme. As such, no complete implementation of it exists. On the other hand, both Microsoft [12], [6] and IBM [18] have since published software libraries that claim to implement fully ho-

momorphic encryption based on Gentry's original construction in [14] and [16]. The next natural step is to make both the lattice and the integer schemes more efficient, bringing fully homomorphic encryption to the public domain.

# Chapter 2

# The DGHV Scheme

In 2010, van Dijk, Gentry, Halevi, and Vaikuntanathan proposed in [31] a new homomorphic encryption scheme over the integers. The DGHV cryptographic scheme encrypts a bit by masking it with noise and a multiple of a secret odd integer $p$, such that modular reductions return the original message.

While we will explore its novel homomorphic properties in Chapters 3 and 4, in this chapter we describe the construction of the scheme in Section 2.1, illustrate it with a numerical example in Section 2.2, define the parameters in Section 2.3, and analyze these extensively with regards to both functionality and security in Section 2.4. Our contributions consist of the numerical example, Lemma 2.3.1, Theorem 2.4.2, and the noise distribution analysis in Section 2.4.4.

## 2.1 Overview of the Scheme

A public key cryptosystem is defined by three algorithms: the generation of the private and public keys, the encryption of a plaintext, and the decryption of a ciphertext; see [22, Def.10.1] for a general algorithmic definition. We will denote an encryption scheme by $\mathfrak{F} = (KeyGen, Enc, Dec)$ throughout this thesis.

### 2.1.1 Key generation

All public key encryption schemes require a private key and its corresponding public key. For example, in unpadded RSA, the private key consists of primes $p_1$ and $p_2$ and the decryption exponent $d$, whereas the public key is made up of the integer $n = p_1 p_2$ and the encryption exponent $e$ (the inverse of the decryption exponent $d$ modulo $(p_1 - 1)(p_2 - 1)$). The idea is that everyone has access to the public key; therefore, anyone can encrypt a plaintext, but only the owner of the private key can decrypt a ciphertext in order to retrieve the corresponding plaintext.

In the DGHV scheme [31], the private key $p$ is an odd integer. Let $\mathcal{K} = 2\mathbb{Z}^+ + 1$ denote the private key space. The public key is a set of near-multiples of $p$, denoted $x = \{x_i = pq_i + r_i \mid i = 0, 1, \ldots, \tau\}$, for a pre-determined $\tau$. We relabel the elements of the public key so that $x_0$ is the largest. The values $q_i$ and $r_i$ are chosen randomly from specific intervals. Let the resulting public key space be denoted by $\mathcal{K}'$. The private key $p$ and the elements of the public key as well as its size $\tau$ must satisfy certain conditions to ensure the functionality and security of the scheme. We will discuss all of these restrictions in Section 2.4.

### 2.1.2 Decryption of a ciphertext

We discuss decryption before encryption simply because defining encryption is much easier when we are able to do so in terms of decryption.

Decryption is a function that takes a ciphertext and the private key as inputs and outputs the plaintext. With that in mind, let's denote the ciphertext space by $\mathcal{C}$, the private key space by $\mathcal{K}$ as before, and the plaintext space by $\mathcal{M}$. Then the decryption function is

$$Dec : \mathcal{C} \times \mathcal{K} \longrightarrow \mathcal{M}.$$

As an example, recall the private key described for RSA. Taking a ciphertext $\mathrm{c} \in \mathcal{C}$, we obtain the following decryption function for unpadded RSA:

$$Dec(\mathrm{c}, (p_1, p_2, d)) = \mathrm{c}^d \mod p_1 p_2.$$

For van Dijk et al.'s scheme, the message space is $\mathcal{M} = \{0, 1\}$ since the scheme is designed to encrypt a single bit at a time. The ciphertext space is $\mathcal{C} = \mathbb{N} \mod x_0$. The decryption function then takes a ciphertext $\mathrm{c} \in \mathcal{C}$ and the private key $p \in \mathcal{K}$ and is evaluated as follows:

$$\begin{aligned} Dec(\mathrm{c}, p) &= (\mathrm{c} \ \operatorname{cmod} p) \mod 2 \\ &= \mathrm{c} - p \left\lceil \frac{\mathrm{c}}{p} \right\rfloor \mod 2, \end{aligned} \tag{2.1.1}$$

where cmod is *centered modular reduction* and is a modular reduction centered around 0; that is, it returns values between $-p/2$ and $p/2$. We will abbreviate the decryption function as $Dec(\mathrm{c})$ since the private key $p$ is set before the creation of the scheme and does not change throughout.

### 2.1.3 Encryption of a plaintext

To define encryption as a function, we need to distinguish between *deterministic* and *probabilistic* encryption. Let's use the notation from the previous section. In

unpadded RSA, for example, encryption takes a plaintext and the public key and outputs a ciphertext as follows:

$$\begin{aligned} Enc_{RSA} : \mathcal{M} \times \mathcal{K}' &\longrightarrow \mathcal{C} \\ (m, (e, n)) &\longmapsto m^e \mod n. \end{aligned}$$

Since $e$ and $n$ are created at the key generation stage, they are fixed. This means that if we input the same message multiple times into the encryption function, it will return the same ciphertext every time. This is called *deterministic encryption*. This is good for certain schemes, but in order to perform the homomorphic operations on the plaintexts of our scheme, we need it to be more pliable.

The DGHV scheme uses *probabilistic encryption* as described by Goldwasser and Micali [17]. In their definition, they discuss the use of a fair coin when encrypting a message. They explain that the encryption of a given message will depend on "the result of a sequence of coin tosses," thus adding an element of randomization to every encryption. Therefore, encrypting the same message multiple times will yield different ciphertexts each time. This adds to the inputs of the encryption function a set of randomization elements, denoted $\mathcal{R}$, such that encryption is now a function

$$Enc : \mathcal{M} \times \mathcal{K}' \times \mathcal{R} \longrightarrow \mathcal{C}.$$

When the public key is fixed, encryption is defined as a function that takes the plaintext and a randomization element and outputs the ciphertext. In DGHV, the randomization elements take the form of random subsets of the public key plus a random secondary noise term $r'$, so $\mathcal{R} \subseteq \{(S, r') | S \subseteq \{1, \ldots, \tau\}, r' \in \mathbb{Z}\}$. We will talk about the restrictions on the elements $r'$ later. We can then define the encryption function as

$$\begin{aligned} Enc : \mathcal{M} \times \mathcal{K}' \times \mathcal{R} &\longrightarrow \mathcal{C} \\ (\mathtt{m}, x, (S, r')) &\longmapsto \mathrm{c} = \left( \mathtt{m} + 2 \sum_{i \in S} x_i + 2r' \right) \mod x_0. \end{aligned} \tag{2.1.2}$$

Note that $x_0$ must be an odd integer, for if it was even we would have $\mathtt{m} = \mathrm{c} \mod 2$, so the message would easily be recovered. Furthermore, when we say a ciphertext $\mathrm{c}$ is an encryption of a message $\mathtt{m}$, we call $\mathrm{c}$ a *fresh ciphertext* and denote it $\mathrm{c} \in Enc(\mathtt{m})$, since a given message has many different encryptions. Expanding the above formula for $\mathrm{c}$, we see that it can be written as

$$\mathrm{c} = \mathtt{m} + 2p \sum_{i \in S} q_i + 2 \sum_{i \in S} r_i + 2r' - zpq_0 - zr_0, \tag{2.1.3}$$

for some quotient $z \in \mathbb{Z}$. There are a few remarks to be made about the structure of the ciphertext. First, we will sample $r'$ from a larger range than the $r_i$; we will discuss

this in Section 2.4.4. Next, we will see in Lemma 2.1.3 that the sum of the $r_i$ must be even. We achieve this by multiplying the sum by 2 rather than by making the $r_i$ even by construction for security reasons. If the $r_i$ were even, than the parity of each $x_i$ of the public key would match that of its corresponding $q_i$, and knowing the parity of the $q_i$'s gives a potential advantage to an attacker in solving the approximate gcd problem (defined in Section 2.4.3), as described by van Dijk et al. [31, Sec.4.1].

We now define some notation that will make it easier to understand the structure of the encryption function.

**Definition 2.1.1.** Let $\mathrm{c} \in \mathit{Enc}(\mathtt{m})$. The *noise term of a ciphertext*, $N(\mathrm{c})$, is the difference between the ciphertext and the nearest multiple of $p$, not including the bit $\mathtt{m}$. Thus, the *noise map* of the encryption scheme is the function that determines the noise term, and is defined as

$$\begin{aligned} N : \mathcal{K}' \times \mathcal{R} &\longrightarrow \mathbb{Z} \\ (x, (S, r')) &\longmapsto N(\mathrm{c}) := 2r' + 2\sum_{i \in S} r_i - zr_0. \end{aligned} \tag{2.1.4}$$

We appreciate in this definition that only the owner of the private key will know the value of $N(\mathrm{c})$ for a given ciphertext. Observe now that all ciphertexts have the form

$$\mathrm{c} = \mathtt{m} + kp + N(\mathrm{c}) \tag{2.1.5}$$

for some $k \in \mathbb{Z}^+$. The intent is that the ciphertexts should be near-multiples of $p$.

**Definition 2.1.2.** A ciphertext $\mathrm{c} \in \mathit{Enc}(\mathtt{m})$ is called a *valid ciphertext* if $\mathit{Dec}(\mathrm{c}) = \mathtt{m}$.

The requirement that a ciphertext be valid imposes certain restrictions on $N(\mathrm{c})$.

**Lemma 2.1.3.** Every $\mathrm{c} \in \mathit{Enc}(\mathtt{m})$ is valid if and only if the image of the noise map is contained in $2\mathbb{Z} \cap (-p/2, p/2 - 1)$.

**Proof:** Let $\mathrm{c} \in \mathit{Enc}(\mathcal{M})$. Recall from (2.1.1) that the decryption function is $\mathit{Dec}(\mathrm{c}) = (\mathrm{c} \mod p) \mod 2$, and from (2.1.5) that the ciphertext $\mathrm{c}$ may be written as $\mathrm{c} = \mathtt{m} + kp + N(\mathrm{c})$. Therefore, we obtain

$$\begin{aligned} \mathit{Dec}(\mathrm{c}) &= [(\mathtt{m} + kp + N(\mathrm{c})) \mod p] \mod 2 \\ &\equiv [(\mathtt{m} + N(\mathrm{c})) \mod p] \mod 2. \end{aligned}$$

We then have that $\mathrm{c}$ is valid if and only if

$$[(\mathtt{m} + N(\mathrm{c})) \mod p] \mod 2 \equiv m.$$

For the above to be true, since $p$ is odd, we must first have that $|\mathtt{m} + N(\mathrm{c})|$ is less than $p/2$ for any $\mathtt{m} \in \{0, 1\}$ so that centered modular reduction does not subtract an unwanted multiple of $p$, which would change the parity of $N(\mathrm{c})$. Therefore, we have

$$-\frac{p}{2} < N(\mathrm{c}) \quad \text{and} \quad 1 + N(\mathrm{c}) < \frac{p}{2} \implies -\frac{p}{2} < N(\mathrm{c}) < \frac{p}{2} - 1.$$

Furthermore, we must have that $N(\mathrm{c}) \bmod 2 \equiv 0$. Together, these two conditions are equivalent to saying that the image of the noise map must be contained in $2\mathbb{Z} \cap (-p/2, p/2 - 1)$. ∎

**Corollary 2.1.4.** Let $x_0 = pq_0 + r_0$ be the largest element of the public key. Then the encryption function produces valid ciphertexts only if $r_0$ is even.

**Proof:** As per Lemma 2.1.3 and (2.1.4), we see that

$$N(\mathrm{c}) = 2r' + 2\sum_{i \in S} r_i - zr_0$$

must always be even in order for c to be a valid ciphertext. Since the quotient $z$ may vary from one encryption to another, we must have that $r_0$ be even in order to guarantee that $N(\mathrm{c})$ is even. ∎

The image of the encryption function can therefore be described, using the previous notation, as

$$Enc(\mathcal{M}) = \left\{ \begin{array}{c} \mathtt{m} + kp + N(\mathrm{c}) \in \mathbb{N} \text{ such that } \mathtt{m} \in \mathcal{M}, \\ k \in \mathbb{Z}^+, N(\mathrm{c}) \text{ even}, -p/2 < N(\mathrm{c}) < p/2 - 1 \end{array} \right\}.$$

Note the use of $\mathcal{M}$ when the message $\mathtt{m}$ is not specified. This notation denotes the set of valid encryptions of an arbitrary $\mathtt{m} \in \{0, 1\}$; therefore, observe that $Enc(\mathcal{M}) \subseteq \mathcal{C} = \mathbb{N} \mod x_0$.

## 2.2 Toy Numerical Example

As an example, let's say Alice is looking to receive information from Bob without Eve gaining access to it. The information that Bob is looking to send is in the form of individual bits, so Alice decides to employ the DGHV scheme as her encryption scheme. We will show, in this section, how Alice will go about creating her keys, as well as what encryption and decryption look like.

Alice begins by choosing her private key $p = 927$. She then creates her public key, which consists of 34 near-multiples of $p$: for $i = 0 \ldots, 33$, she generates uniformly random $q_i \in [0, 2^{30}/927) = [0, 1158297]$ and $r_i \in [-7, 7]$. She relabels the elements so that $x_0$ is the largest. Her public key is listed below.

| | | | | |
|---|---|---|---|---|
| 1030997355 | 64164157 | 875035167 | 262365107 | 105433279 |
| 893696599 | 294035131 | 934796070 | 858242559 | 330596009 |
| 790883948 | 552622706 | 213914520 | 242086972 | 359840079 |
| 774789384 | 646413786 | 200993999 | 259974362 | 731809952 |
| 365847961 | 79808216 | 516081301 | 1004724321 | 684587639 |
| 308720670 | 757872558 | 397025761 | 204162487 | 774580813 |
| 570337679 | 429022092 | 543856991 | 821258037 | |

As an example for the reader, here is the breakdown of the first three elements of Alice's public key:

$$\begin{aligned} x_0 &= 1030997355 = 1112187 \cdot 927 + 6, \\ x_1 &= 64164157 = 69217 \cdot 927 - 2, \\ x_2 &= 875035167 = 943943 \cdot 927 + 6. \end{aligned}$$

Alice publishes her public key and informs Bob that, should he wish to send her a bit $\mathtt{m}$, he must take a random subset of her public key (not including $x_0$), generate a secondary noise term $r' \in [-15, 15]$, and compute, as per (2.1.2),

$$\mathrm{c} = \left( \mathtt{m} + 2 \sum_{i \in S} x_i + 2r' \right) \mod x_0.$$

Bob decides to encrypt five bits and send the corresponding ciphertexts to Alice. His first bit is $\mathtt{m}_0 = 1$. He generates a random bit string of length $\tau$ in order to create a random subset of Alice's public key. We will write this bit string as a vector $\vec{\mathtt{z}} = [101\,10011\,10110\,10011\,11011\,11101\,01000]$, where the first bit is denoted $\mathtt{z}_1$. His secondary noise term (generated uniformly randomly) is $r' = -12$. Thus, his first ciphertext becomes

$$\mathrm{c}_0 = \left( 1 + 2 \sum_{i=1}^{\tau} \mathtt{z}_i x_i + 2(-12) \right) \mod 1030997355 = 16222417.$$

In the same manner, Bob generates a random bit string and noise term for each bit he wants to encrypt. The five bits Bob is encrypting and their corresponding ciphertexts are listed below.

| Plaintext | Ciphertext |
|---|---|
| $\mathtt{m}_0 = 1$ | $\mathrm{c}_0 = 16222417$ |
| $\mathtt{m}_1 = 1$ | $\mathrm{c}_1 = 271326272$ |
| $\mathtt{m}_2 = 1$ | $\mathrm{c}_2 = 318596869$ |
| $\mathtt{m}_3 = 0$ | $\mathrm{c}_3 = 616274125$ |
| $\mathtt{m}_4 = 0$ | $\mathrm{c}_4 = 696078680$ |

We note that Eve cannot infer the message being sent from the parity of the ciphertext, as these don't always match.

Bob now sends these ciphertexts to Alice. Upon receiving them, Alice first computes $\mathrm{c}_0 \bmod p = -83$, then takes the result and reduces it modulo 2 to retrieve the bit $\mathtt{m}_0 = 1$ that Bob sent. Notice the necessity of the centered modular reduction; evaluating $\mathrm{c}_0 \bmod p$ returns 844 which, upon reduction modulo 2, would give the wrong bit 0. Alice decrypts the other four bits in the same manner to obtain the message Bob sent to her.

This numerical example was created using Matlab code; snippets of code from the different scripts may be found in Appendix A. This purpose of this example is to give the reader a better understanding of the relative sizes of the various elements of the scheme at a very low level of security. In the next sections, we will demonstrate how exactly to realize the scheme as well as address how the private key and message are hidden.

## 2.3 Parameters and Algorithms

This section provides details necessary for the interested reader to implement the DGHV scheme. Table 2.1 details the parameters used to define the DGHV scheme from [31]. These parameters will be used throughout the remainder of the thesis. All parameters are positive integers. The security parameter $\lambda$ refers to the number of *bits of security* of the scheme; in other words, the most efficient attacks known would take $2^\lambda$ operations to break the scheme. Our goal is to relate all other parameters to $\lambda$. The necessary constraints on these parameters to achieve this will be discussed in Sections 2.4, 3.3, 3.4, 4.3, and 4.6.

Denote by $\eta$ the bit-length of the private key $p$; that is, $p$ is chosen uniformly randomly from $(2^{\eta-1}, 2^\eta) \cap (2\mathbb{Z}+1)$. The public key contains $\tau+1$ near-multiples of $p$, of bit-length at most $\gamma$, defined as follows. For each $i = 0, \ldots, \tau$, choose an integer $q_i$ uniformly randomly from $[0, 2^\gamma/p]$, select an integer $r_i$ uniformly randomly from $(-2^\rho, 2^\rho)$, and generate $x_i = pq_i + r_i$. Therefore, the term near-multiple of $p$ is defined as an integer that is at most $2^\rho$ away from a multiple of $p$. Once the set $x$ is generated, relabel so that $x_0$ is the largest element.

| Parameter | Description |
|---|---|
| $\lambda$ | Security parameter |
| $\rho$ | Size of the $r_i$; also known as the noise parameter |
| $\rho'$ | Secondary noise parameter; size of $r'$ |
| $\eta$ | Size of the private key $p$ |
| $\gamma$ | Size of the $x_i$, the elements of the public key |
| $\tau$ | Number of elements in the public key |

Table 2.1: A list of parameters of the DGHV scheme. Here, a parameter's size refers to its bit-length.

As per Section 2.1.3, we must have that $x_0$ is an odd integer, and its associated noise $r_0$ must be even by Corollary 2.1.4. If one or both of these two conditions is not satisfied, the process to create a public key must be restarted.

**Lemma 2.3.1.** The expected value for the number of times the key generation process must be restarted is 4.

**Proof:** For the generated key to be valid, $x_0$ must be odd and $r_0$ even; therefore, $q_0$ must be odd. This combination of $q_0$ odd and $r_0$ even happens one quarter of the time, since variables are uniformly distributed. Let $T$ represent the number of times that the public key must be generated to obtain a valid key. Then $P(T = 1) = \frac{1}{4}$. Next, we have that the key must be generated exactly $n$ times if and only if the previous $n - 1$ iterations failed to create a valid key. Therefore, $P(T = n) = \left(\frac{3}{4}\right)^{n-1} \cdot \frac{1}{4}$. This allows us to calculate the expected number of times the key must be generated in order to obtain a valid key (using a known power series representation):

$$E(T) = \sum_{n=1}^{\infty} n \left(\frac{3}{4}\right)^{n-1} \cdot \frac{1}{4} = \frac{1}{\left(1 - \frac{3}{4}\right)^2} \cdot \frac{1}{4} = 4.$$

∎

As an aside, we note another circumstance in which the key generation algorithm would have to be restarted; namely, if $x_0$ has bit-length not equal to $\gamma$. We will discuss this in more detail in Section 2.4.3.

Given the likelihood of rejection of $x_0$, if we were to simply discard the top element and check the next largest element, then the largest element of a given public key would statistically be smaller than expected. Therefore, when $x_0$ is rejected, the entire key generation process is restarted to maintain randomness. Once $x_0$ satisfies the necessary constraints, we have a public key $x$ that may be published.

To encrypt a bit $\mathtt{m}$, we choose a random subset $\{x_i \mid i \in S\}$ of the public key as well as a secondary noise value $r'$. In order to obtain $S$, we generate a random bit

string of length $\tau$, denoted $\vec{\mathtt{z}} = (\mathtt{z}_1, \ldots, \mathtt{z}_\tau)$, and we define the set $S = \{i : \mathtt{z}_i = 1\}$. This way, the resulting probability distribution is uniform across all possible subsets. Note that the first and largest element of the public key, $x_0$, is excluded in this process since the encryption function involves modular reduction by $x_0$. The noise $r'$ is then chosen uniformly randomly from $\mathbb{Z} \cap (-2^{\rho'}, 2^{\rho'})$. Finally, we encrypt using the function (2.1.2) and decrypt using the function (2.1.1).

We have now detailed how to build the DGHV scheme in practice, and how to choose its various elements. In the next section, we will look at the constraints needed on the parameters in Table 2.1 in order for the scheme to function as needed.

## 2.4 Constraints on Parameters

For the scheme described in Section 2.1 to be considered a secure encryption scheme, it must satisfy a number of conditions, which we will paraphrase from [19, Sec.1.7.1]: for any private key, public key, plaintext, and ciphertext, it must be easy to compute a ciphertext as an encryption of a plaintext under the public key; it must be easy to retrieve a plaintext as the decryption of the ciphertext under the private key; it must be very difficult to retrieve a plaintext from its ciphertext without the private key; and it must still be difficult to achieve the latter even when given multiple other plaintext/ciphertext pairs. When talking about security, we use a generic definition, first introduced by Goldwasser and Micali [17], which is based on complexity theory.

**Definition 2.4.1.** A *secure encryption scheme* is an encryption scheme in which the following property holds: whatever is efficiently computable about the plaintext given the ciphertext, is also efficiently computable without the ciphertext.

This notion of security is also known as *semantic security*, and is equivalent to the concept of *indistinguishability of ciphertexts under a chosen-plaintext attack* as described by Katz and Lindell in [22, Def.10.4]. In other words, when given two plaintexts and the encryption of one of them, without knowing in advance of which one, it is computationally infeasible to distinguish to which plaintext the ciphertext corresponds. The interested reader may pursue in more detail the concepts of security against chosen-plaintext attacks and the stronger security against chosen-ciphertext attacks in public-key cryptography in [22, Ch.10]. In this section, we will give heuristic arguments as well as explain some of the problems on which the security of the DGHV scheme is based.

Public key encryption schemes are built on underlying mathematical problems that are considered to be hard to solve. In the case of unpadded RSA, to recover the private key from the public key or to decrypt a ciphertext, one must be able to factor $n$. The security of unpadded RSA is widely believed to be equivalent to factoring; indeed, under some attack models it has been shown to be equivalent (see [1]). For the

DGHV encryption scheme, the authors establish in [31] some very general constraints and a "convenient parameter set" to realize the scheme, which we describe in Section 4.6, but do not provide many details to justify their choices. Therefore, we will explore, in this section, the constraints on the parameters related to the functionality of the encryption scheme, followed by the various security constraints imposed on the parameters by the hardness assumptions of the underlying problems.

For each problem we will give a definition, describe its hardness based on the attacks that have been brought forth against them, and then set up bounds on the sizes of our parameters in order to ensure that these attacks fail. We will continue to use, in this section and throughout the remainder of the thesis, the parameters described in Table 2.1.

### 2.4.1 Functional constraints

We first recall from Section 2.1.3 that in our public key, $x_0$ must be odd and its associated noise $r_0$ must be even. This does not constrain our parameter sizes but is nonetheless a functional constraint. Next, observe that, since we need the noise to be no greater than $p/2$ for decryption to function properly, as per Lemma 2.1.3, we must definitely have that each $r$ and $r'$ be smaller than $p/2$ for any $p \in (2^{\eta-1}, 2^{\eta})$. This gives us the constraint $2^{\rho} \leq 2^{\eta-1}/2$, or

$$\rho \leq \eta - 2. \tag{2.4.1}$$

Similarly for $r'$, we obtain

$$\rho' \leq \eta - 2. \tag{2.4.2}$$

Furthermore, the elements of the public key should be large near-multiples of $p$ in order to effectively hide $p$, so in particular, we have

$$\eta < \gamma. \tag{2.4.3}$$

Finally, we examine the accumulation of noise on the ciphertext during the encryption process in order to constrain the parameters accordingly to maintain functionality. For the following proof, assume $x_0$ has bit-length exactly $\gamma$.

**Theorem 2.4.2.** Suppose that the parameters from Table 2.1 satisfy $2^{\rho'} \geq \tau 2^{\rho}$ (Section 2.4.4 demonstrates the origin of this condition). Then the encryption function will yield a valid ciphertext if

$$\rho' \leq \eta - 5. \tag{2.4.4}$$

**Proof:** We have, from (2.1.4), that our noise term is defined as

$$N(\mathrm{c}) = 2r' + 2\sum_{i \in S} r_i - zr_0.$$

To satisfy Lemma 2.1.3, we require $|N(\mathrm{c})| \leq p/2 - 1$ for any value of $p$; in particular, the minimum.

Using the parameters established in Section 2.3, we have that $p \geq 2^{\eta-1}$, $|r| \leq 2^{\rho}$, and $|r'| \leq 2^{\rho'}$. What's more, the size of the subset of the public key, $|S|$, is bounded above by $\tau$. This allows us to construct an upper bound for the noise:

$$\begin{aligned} |N(\mathrm{c})| &= \left| 2r' + 2\sum_{i \in S} r_i - zr_0 \right| \\ &\leq 2|r'| + 2\sum_{i \in S} |r_i| + |z||r_0| \quad \text{by the Triangle Inequality} \\ &\leq 2 \cdot 2^{\rho'} + 2\tau 2^{\rho} + z2^{\rho}. \end{aligned}$$

To give a more accurate bound, we need a bound on $z$. Let $a$ be the unmodded version of $\mathrm{c}$, that is,

$$a = \mathtt{m} + 2r' + 2\sum_{i \in S} x_i.$$

Then by definition $z = \left\lfloor \frac{a}{x_0} \right\rfloor$, so since $\mathtt{m} + 2r' \leq 1 + 2^{\rho'+1} < 2^{\gamma-1} \leq x_0$ by (2.4.2) and (2.4.3), and $x_i < x_0$ for all $i$, we obtain

$$z = \left\lfloor \frac{\mathtt{m} + 2r' + 2\sum_{i \in S} x_i}{x_0} \right\rfloor < \frac{1 + 2^{\rho'+1}}{x_0} + \frac{2\tau x_0}{x_0} < 1 + 2\tau.$$

Now by the hypothesis $\tau 2^{\rho} < 2^{\rho'+1}$ we can calculate an upper bound on the noise:

$$|N(\mathrm{c})| < 2^{\rho'+1} + \tau 2^{\rho+1} + (1 + 2\tau)(2^{\rho}) < 2^{\rho'+1} + 2^{\rho'+1} + 2^{\rho} + 2^{\rho'+1} < 2^{\rho'+3}.$$

Finally, if $\rho' \leq \eta - 5$, we obtain $2^{\rho'+3} \leq 2^{\eta-2} < p/2$ and, by our strict inequalities, we have

$$|N(\mathrm{c})| \leq \frac{p}{2} - 1$$

for any $p \in (2^{\eta-1}, 2^{\eta})$, and encryption will yield a valid ciphertext. ∎

We will note here that, because of the strict inequalities above, we have, for a ciphertext $\mathrm{c} \in Enc(\mathcal{M})$, that

$$|N(\mathrm{c})| + 1 < 2^{\rho'+3}. \tag{2.4.5}$$

This will be important in Chapter 3.

### 2.4.2 Security: the subset-sum problem?

Recall the formula given for a ciphertext in the DGHV scheme is simply

$$c = \left( m + 2\sum_{i \in S} x_i + 2r' \right) \mod x_0.$$

We see that the encryption of a ciphertext includes the sum of a random subset of the public key; therefore, it is natural to question whether part of the security of the scheme is based on the hardness of the subset-sum problem.

**Definition 2.4.3** (Subset-sum problem (SSP))**.** Let $S$ be a set of positive integers and $t \in \mathbb{Z}^+$ be a target integer. The subset-sum problem asks to find whether there exists a subset $S' \subseteq S$ such that $t = \sum_{s \in S'} s$.

This definition is taken from [9, Sec.34.5.5]. That section also proves that the subset-sum problem is NP-complete. Notice here, however, that part of the encryption function is to reduce modulo $x_0$ in order to obtain a ciphertext. This leads to a slightly different problem, described by Micciancio and Goldwasser in [25, Ch.3, Sec.2] and adapted here.

**Definition 2.4.4** (Modular subset-sum problem (MSSP))**.** Given $S$ and $t$ as before and letting $x_0$ be the modulus, find whether there exists a subset $S' \subseteq S$ such that $\sum_{s \in S'} s = t \mod x_0$.

Following this definition, Micciancio and Goldwasser explain that any algorithms used to solve the subset-sum problem may be modified to solve the modular subset-sum problem; therefore, MSSP is no more difficult than SSP. They go on to define the concept of the *density of the modular subset-sum problem* to be the ratio between the number of elements in the set and the size of the modulus. For SSP, density is defined as the ratio between the number of elements in the set and the size of its largest element. In terms of the parameters of the DGHV scheme, this means that the density, denoted by $\partial$, is defined as

$$\partial = \frac{\tau}{\gamma}.$$

The SSP and MSSP can be divided into two classes of problems: *high-density* (when $\partial > 1$) and *low-density* (when $\partial \leq 1$). In practice, the DGHV scheme uses the high-density version, but we will examine the low-density class first.

Coster et al. [10] detail an algorithm that relies on a solution to the *shortest vector problem* (SVP) to solve certain low-density cases. The shortest vector problem is well-studied in mathematics. There exist simple algebraic algorithms to solve the

SVP in lattices of dimension 2, but solving the problem becomes very difficult in lattices of larger dimensions. What's more, Ajtai [2] proved that the SVP in lattices with the $\ell_2$ norm is NP-hard under certain specific conditions he calls "randomized reductions". The most efficient algorithm used to approximate the SVP in a lattice of dimension $n$ is the LLL algorithm (described in [25, Ch.2]). Coster et al.'s algorithm to solve the MSSP requires a lattice oracle; in other words, the algorithm needs to be given an exact solution to the SVP, rather than an approximation (more on what approximation LLL gives in the next section). Therefore, it is only a theoretical attack on the subset-sum problem; however, if a lattice oracle were to exist, the algorithm would be able to solve all subset-sum problems with a density of $\partial < 0.9408$.

Let us now examine subset-sum problems that fall in the high density category. Coster et al. [10] explain that in the high-density case, there are multiple subsets that sum to the same target, which is not true for the low-density case. Micciancio and Goldwasser go into more detail in [25, Ch.3, Sec.2]; according to them, for a subset with density $\partial$, on average each target $t \in [0, x_0)$ has $2^{\partial}$ subsets which sum to it. Therefore, any attack based on knowing a solution to the subset-sum problem is no longer effective when the density is greater than 1, as even if there exists a solution, it is not unique and thus is not useful.

Before we advance any further, let us have a closer look at the problem at hand. The structure of the ciphertext does indeed include a subset-sum, but does the security of this scheme actually rely on the subset-sum problem? In reality, we are not really trying to hide the subset used in the encryption; rather, we are trying to distribute the ciphertexts over the interval $[0, x_0)$ in order to effectively hide $\mathtt{m}$. Thus, at a minimum, we want

$$\tau > \gamma, \tag{2.4.6}$$

since, when $\partial > 1$, the function that takes a subset to its sum is "likely to be surjective" [25, Ch.3, Sec.2], and as mentioned above each sum will on average have multiple preimages.

Taking the security one step further, when examining possible ciphertexts, we see that the number of elements in $Enc(\mathcal{M})$ is bounded above by $\tau 2^{\gamma-\eta} \cdot 2^{\rho'+4}$, based on the maximum subset sums and noise terms. While it would seem desirable, for security, that $Enc(\mathcal{M}) = [0, x_0)$, this would imply choosing $\tau 2^{\gamma-\eta+\rho'+4} \geq 2^{\gamma}$, or

$$\log \tau + \rho' \geq \eta - 4,$$

which contradicts (2.4.4). In particular, although van Dijk et al. don't mention it in [31], the distribution of fresh ciphertexts is not uniform in $[0, x_0)$, but rather clustered around multiples of $p$ modulo $x_0$.

We have now seen that the DGHV scheme's security is not based on the subset-sum problem after all. In fact, the DGHV authors detail a reduction of the security of

their scheme to the hardness of the approximate gcd problem, under the assumption

$$\tau \geq \gamma + \lambda, \tag{2.4.7}$$

which is stronger than (2.4.6) above. We will examine the approximate gcd problem next.

### 2.4.3 Security: the approximate gcd problem

Recall from Section 2.1.1 that the elements of the public key are near-multiples of $p$ of the form $x_i = pq_i + r_i$. In order to find parameter choices for which it is expected to be hard to recover $p$ from the set of $x_i$'s, we need to define and analyze the hardness of the approximate gcd problem. We will adapt Black's definition in [5, Sec.2.1] to establish the problem.

**Definition 2.4.5** (Approximate gcd problem)**.** Let $p \in \mathbb{Z}^+$. For $i = 0, \ldots, \tau$, let $q_i, r_i \in \mathbb{Z}$ be chosen randomly as in Section 2.3. Given the set of $\tau$ integers of the form $x_i = pq_i + r_i$, find $p$.

Several authors have considered and analyzed the approximate gcd problem including [5], [7], [8], [13], [20], [25], [26]. The DGHV authors detail a reduction of the security of their scheme to the hardness of the approximate gcd problem [31, Sec.4.1] In the following paragraphs, we briefly summarize some known attacks on the approximate gcd problem, and outline the parameter constraints needed to render these attacks infeasible.

#### A brute-force attack

A first attempt to attack the system, described by van Dijk et al. in [31], is a brute-force attack, where one takes two $x_i$'s, say $x_1$ and $x_2$, and attempts to guess their associated noise values $r_1$ and $r_2$. Let $\tilde{r}_1$ and $\tilde{r}_2$ be the guessed values. In order to verify the guess, compute $\gcd(x_1 - \tilde{r}_1, x_2 - \tilde{r}_2)$ to see if a factor of the correct size, that is, in the range $(2^{\eta-1}, 2^{\eta})$, appears.

This attack has a running time of approximately $2^{2\rho}$. Since the conception of the DGHV scheme, better attacks have been put forth, including results by Chen and Nguyen in [7] whose attack has a running time of $2^{3\rho/2}$. These attacks are exponential in the size of the noise, so the constraint

$$\frac{3\rho}{2} \geq \lambda$$

enables the scheme to remain secure against this type of attack; however, in practice, for a higher level of security, the following constraint is used instead in [31], namely

$$\rho \geq \lambda. \tag{2.4.8}$$

When thinking about brute-force attacks, one must take into consideration other factors that could render the search space of the attack smaller, subsequently giving an attacker an advantage. One such factor is the size of $x_0$, as mentioned in Section 2.3. Because of the inherent randomness of the creation of the public and private keys, it is possible that certain parameters allow $2^\gamma = qp + \varepsilon$ for $\varepsilon < 2^\rho$. If that holds, it is then possible for $x_0 = pq_0 + r_0$ to be such that $r_0 > \varepsilon$, in which case the bit-length of $x_0$ would be greater than $\gamma$. Since the public key and underlying parameters (such as $\gamma$) are visible to an attacker, they would then know that $0 < \varepsilon < r_0$ and the number of elements through which to search when conducting a brute-force attack would be reduced, giving them an advantage in discovering $p$. We remove this advantage by ensuring, at the key generation stage, that $x_0$ has bit-length at most $\gamma$; in fact, for practical purposes, we will ensure that $x_0$ has bit-length equal to $\gamma$, which is not explicitly stated in [31].

A slightly different approach used by Black [5, Sec.5.1.3] describes an alternative extended Euclidean algorithm that allows for "noisy multiples of $p$" in order to retrieve $p$ from two $x_i$'s. This algorithm works when $\gamma + \rho \leq 2\eta$; therefore, another security requirement for our parameters is

$$\gamma \geq 2\eta - \rho. \tag{2.4.9}$$

### A lattice attack on the approximate gcd of two numbers

Howgrave-Graham [20] summarizes previous work that uses continued fractions to attempt to solve the approximate gcd problem, then devises a lattice-based algorithm to find the approximate gcd of two integers. He calls the problem the *general approximate common divisor problem* (GACDP) and describes a sub-problem which he names the *partially approximate common divisor problem* (PACDP). The latter occurs when one of the two $r_i$'s is 0; in other words, one of the two integers in question is a perfect multiple of $p$.

The lattice-based algorithms he devises perform only under certain conditions; Howgrave-Graham gives an upper bound on his parameter $\beta = \rho/\gamma$ with respect to his parameter $\alpha = \eta/\gamma$ in order for his algorithms to solve the PACDP and GACDP. These results are neatly summarized in a graph relating $\beta$ to $\alpha$ [20, Fig.61]. The lattice-based algorithm used to solve the PACDP performs as expected for the largest values of $\beta$: it can recover the approximate gcd of two numbers as long as $\beta \leq \alpha^2$. Therefore, in order to make the DGHV scheme secure against not only Howgrave-Graham's algorithms but also anyone who uses a reduction from the GACDP to the PACDP to create an attack (such as described in [7]), we require $\beta > \alpha^2$ which, for our parameters, corresponds to

$$\rho > \frac{\eta^2}{\gamma}. \tag{2.4.10}$$

### A lattice attack on the approximate gcd of multiple numbers

More recently, Cohn and Heninger [8] generalized Howgrave-Graham's method from [20] to create a lattice-based attack using many $x_i$'s rather than two. Let $x_i = pq_i + r_i$ for $i = 1, \ldots, m$. They start their algorithm by constructing polynomials $Q_1, \ldots, Q_m$, each of $m$ variables, in such a manner that

$$Q_i(r_1, \ldots, r_m) = 0$$

for all $r_1 \ldots, r_m$ (without knowing the $r_i$'s in advance, but knowing their upper bounds). They do this by creating an integer lattice based on the $x_i$ and the upper bounds of the $r_i$, then creating the polynomials in the $x_i$ only. Each polynomial $Q_i$ corresponds to a given vector in the lattice. From there, they bound $|Q_i(r_1, \ldots, r_m)|$ by taking the $\ell_1$ norm of the corresponding vector.

The polynomials are set up such that, if their bound is small enough, a polynomial relation between the $r_i$'s is brought forth. Therefore, bounding the polynomials involves bounding the length of the corresponding vectors; thus, their algorithm relies on a solution to the SVP as mentioned in Section 2.4.2.

Recall that the most efficient algorithm used to approximate the SVP in a lattice of dimension $n$ is the LLL algorithm. Unfortunately, even the LLL doesn't fully solve the SVP. Instead, it returns a vector that is within an exponential factor of the shortest vector. Micciancio and Goldwasser show in [25, Ch.2, Sec.2] that this exponential factor is equal to $\left(2/\sqrt{3}\right)^n$; however, Cohn and Heninger use a weaker approximation of $2^n$ in their calculations.

To ensure the security of the DGHV scheme with respect to this attack, we must ensure that the output of the LLL algorithm be exponentially large compared to a small power of $p$. The authors of the DGHV scheme calculate this bound in [31] and conclude that, in order to maintain security against lattice-based attacks, they require

$$\frac{\gamma}{\eta^2} \geq \lambda. \tag{2.4.11}$$

## 2.4.4 Security: the secondary noise parameter

So far, we have discussed how to securely hide the private key within a public key of near-multiples of $p$, as well as how the use of a subset-sum is important to the encryption scheme, but not so important to the security of it. Another aspect that van Dijk et al. discuss in [31] is making the distribution of the noise terms on ciphertexts as close as possible to uniform. Doing so reduces the advantage an attacker might have in attempting to retrieve plaintexts from their corresponding ciphertexts.

The authors of the DGHV scheme focus on the distribution of the noise terms of the elements of the public key. Recall that elements of the public key are of the form $x_i = pq_i + r_i$ and that the noise terms $r_i$ are chosen uniformly randomly from the

interval $(-2^\rho, 2^\rho)$. Let $X$ be the random variable representing the $r_i$. The distribution of $X$ is of course uniform, denoted $U(-2^\rho, 2^\rho)$, and has a mean $\mu = 0$ and variance $\sigma^2 = \frac{1}{12}2^{2\rho+2}$. Now let $Y$ be the random variable representing the sum of the $r_i$'s. If the subset-sums were of the same size, then $Y$ would follow what is called the *Irwin-Hall distribution*, as per [30, Ch.4]. However, the actual distribution of $Y$ is more complex, since subsets vary in size in each iteration. To simplify our analysis of the distribution, we will assume that all subsets are of size $\tau$. What's more, because $\tau$ is large, we are able to use the Central Limit Theorem (see [30, Ch.5, Sec.4]) to approximate the distribution of $Y$ by a normal distribution with mean $\mu = 0$ and with variance

$$\sigma^2 = \frac{\tau}{12}2^{2\rho+2}.$$

We note that, despite these simplifications, the actual distribution may still be approximated with a normal curve. Experimental data supports this approximation. We generated, using Matlab, 1000 subsets of $r_i \in [-7, 7]$, ranging in size from 1 to $\tau = 250$, and computed their sums. Their distribution may be seen in Figure 2.1. Observe that the mean is not exactly 0, due to discrete nature and low resolution of this sample.

The non-uniform distribution of $Y$ inserts bias into the distribution of the ciphertexts which could be exploited by an attacker. Thus, we want to modify the distribution of the total noise so that its corresponding density function is constant over at least the interval $(-2^\rho, 2^\rho)$. We do so by adding a secondary noise term $r'$ which is distributed according to the random variable $Z \sim U(-2^{\rho'}, 2^{\rho'})$. We next determine constraints on $\rho'$ that achieve this smoothing.

If we let $f$ and $g$ represent the density functions of the random variables $Y$ and $Z$ respectively, then the density function of $Y + Z$ is given by the *convolution* of $f$ and $g$, denoted $f * g$, which is the function defined by [30, Ch.2, Sec.7]

$$(f * g)(w) = \int_{-\infty}^{\infty} g(w - y)f(y)\, dy.$$

We see that $g(w - y)f(y) = 0$ unless $g(w - y) \neq 0$. Since $g$ is the density function of a uniformly distributed random variable on the bounded set $(-2^{\rho'}, 2^{\rho'})$, $g(w - y) \neq 0$ if and only if $w - y \in (-2^{\rho'}, 2^{\rho'})$ or

$$y \in (w - 2^{\rho'}, w + 2^{\rho'}).$$

Therefore, the convolution may be rewritten as

$$(f * g)(w) = 2^{-\rho'-1} \int_{w-2^{\rho'}}^{w+2^{\rho'}} f(y)\, dy.$$

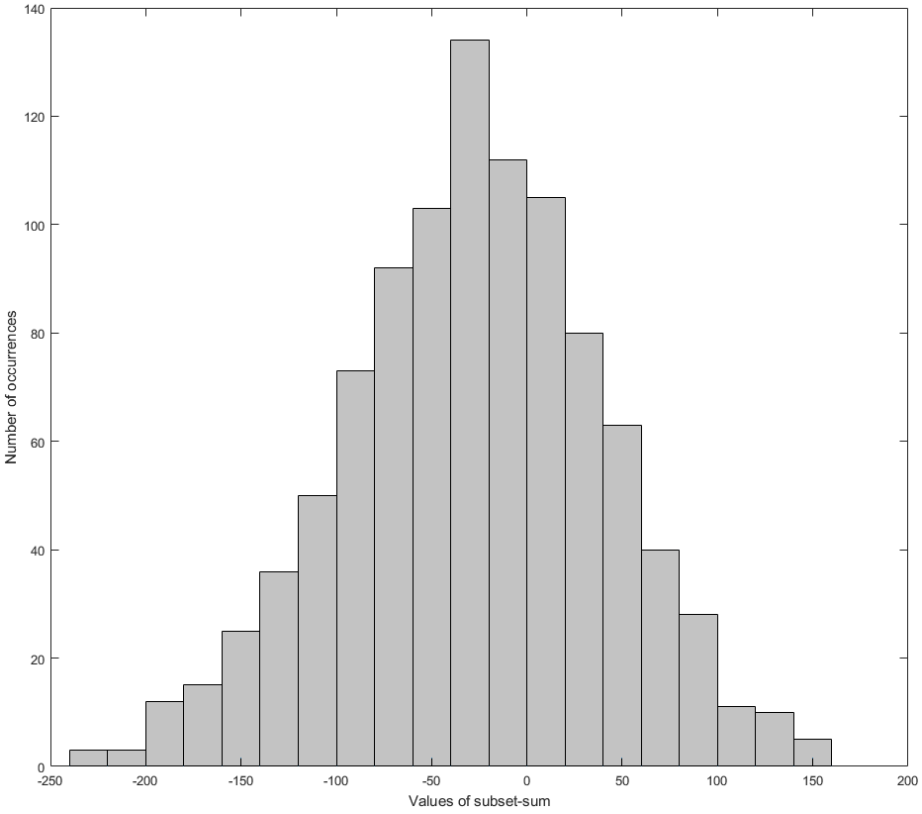


Figure 2.1: Distribution of 1000 random subset sums in Matlab. Subsets range in size from 1 to $\tau = 250$, and the elements are between -7 and 7.

We are approximating $Y$ by a normal distribution, but in fact $f(y) = 0$ when $y < -\tau 2^{\rho}$ or when $y > \tau 2^{\rho}$, meaning

$$\int_{-\tau 2^{\rho}}^{\tau 2^{\rho}} f(y)dy = 1.$$

Therefore, for all $w$ such that $(-\tau 2^{\rho}, \tau 2^{\rho}) \subseteq (w - 2^{\rho'}, w + 2^{\rho'})$, we have $(f * g)(w) = 2^{-\rho'-1}$. For this to hold for all $w \in (-2^{\rho}, 2^{\rho})$, we want

$$-2^{\rho} + 2^{\rho'} \geq \tau 2^{\rho}$$

or $2^{\rho'} \geq 2^{\rho}(\tau + 1)$, which is equivalent to

$$\rho' \geq \rho + \log(\tau + 1). \tag{2.4.12}$$

Once again, experimental data supports this calculations, even with much smaller sample sizes and slightly weaker assumptions. Figure 2.2 depicts the distribution of

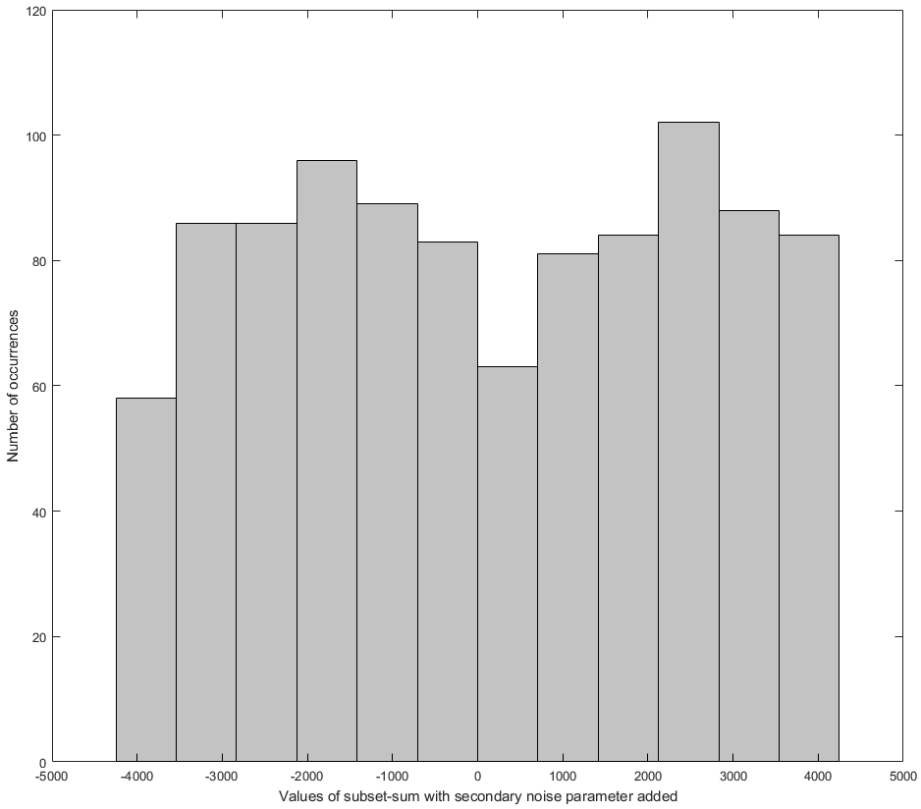


Figure 2.2: Distribution of 1000 random subset sums in Matlab. Subsets range in size from 1 to $\tau = 250$, and the elements are between -7 and 7, with a random integer between -2047 and 2047 added to each sum.

the same data as Figure 2.1, but since $\rho = 3$, we take $\rho' = 11 \geq 3 + \log(251)$, so a random $r' \in [-2047, 2047]$ is added to each subset-sum. Observe that the distribution no longer appears normal, but is in fact much closer to uniform as desired, despite the small size of the example.

### 2.4.5 Summary

We now offer a summary of all constraints detailed in the previous sections. Combining (2.4.8), (2.4.12), (2.4.4), (2.4.9), and (2.4.7) gives the following chain of inequalities:

$$\lambda \leq \rho < \rho' < \eta < \gamma < \tau. \tag{2.4.13}$$

Next, observe that a few inequalities are replaced with stronger versions in later sections. We see that (2.4.4) supersedes (2.4.2); the chain of inequalities means that (2.4.4) is stronger than (2.4.1); that (2.4.9) and (2.4.1) together imply (2.4.3); and that, because of (2.4.8), we have that (2.4.11) is stronger than (2.4.10). What's

| Equation | Reference |
|---|---|
| $\lambda \leq \rho < \rho' < \eta < \gamma < \tau$ | (2.4.13) |
| $\eta \geq \rho' + 5 \geq \rho + \log(\tau + 1) + 5$ | (2.4.4) and (2.4.12) |
| $\gamma \geq \lambda\eta^2$ | (2.4.11) |
| $\tau \geq \gamma + \lambda$ | (2.4.7) |
| $x_0$ odd and $r_0$ even | Corollary 2.1.4 |

Table 2.2: Basic parameter constraints on the DGHV scheme

more, for $\lambda \geq 2$, (2.4.11) implies (2.4.9). With those equations no longer in use, we summarize the remaining equations in Table 2.2.

Note that the numerical example from Section 2.2 satisfies all the constraints from Table 2.2, except for (2.4.11) and (2.4.12), for $\lambda = 3, \rho = 3, \rho' = 4, \eta = 10, \gamma = 30$, and $\tau = 33$. This is simply due to the fact that in order to properly demonstrate how the scheme works, a small enough example had to be created, and it is impossible to satisfy these constraints on such a small scale. For the numerical examples in Sections 3.2.1 and 3.3.2, these parameters will be kept, only because generating proper parameters for even a small $\lambda$ does not give way to a sensible-looking example (as we will see next).

Using the constraints from Table 2.2, we can choose the parameters in the following way: once we set $\lambda$, set $\rho = \lambda$. Then, we see that (2.4.7) and (2.4.11) imply

$$\eta \geq \lambda + \log(\lambda\eta^2 + \lambda + 1) + 5.$$

For example, we have that $\eta = 2\lambda$ satisfies the above inequality for $\lambda \geq 20$, but numerical methods may also be used to determine $\eta$ for a given $\lambda$. After that, we set $\gamma = \lambda\eta^2$, $\tau = \gamma + \lambda$, and $\rho' = \lceil \rho + \log(\tau + 1) \rceil$ to complete our set of parameters.

### Alice chooses new parameters

Alice realizes that her scheme doesn't quite satisfy all the security constraints outlined previously, so she decides to choose her parameters more carefully. She decides that $\lambda = 10$ will be sufficient for her [1]. She then sets $\rho = 10$. Next, she determines that she needs

$$\eta \geq 15 + \log(10\eta^2 + 11).$$

Using an iterative method, she finds that $\eta = 28$ is the smallest value that satisfies this inequality, but in order to give herself a little more room, she uses $\eta = 30$. From there, she sets up her remaining parameters:

$$\gamma = \lambda\eta^2 = 9000$$

[1] As we saw in Chapter 1, NIST recommends $\lambda = 112$, but we want to keep a manageable numerical example.

$$\tau = \gamma + \lambda = 9010$$
$$\rho' = \lceil \rho + \log(\tau + 1) \rceil = 24.$$

With these parameter constraints, the DGHV scheme successfully encrypts single bits as very large integers such that it is infeasible to recover the plaintext bit from the ciphertext using known attacks. We will discuss how the DGHV scheme is in fact a homomorphic encryption scheme and modify our current constraints to accommodate multiple operations in the next chapter.

# Chapter 3

# Homomorphic Aspects of the DGHV Scheme

Continuing our analysis of the DGHV scheme [31], we describe in this chapter how to modify it to allow a limited homomorphic capacity. We explain the concept and goal of homomorphic encryption in Section 3.1 and demonstrate how to perform one homomorphic operation in the DGHV scheme in Section 3.2. We then examine, in Section 3.3, one of the necessary optimizations proposed by van Dijk et al. detailing how it affects the parameter constraints previously established in Table 2.2 and modifies the public key from Chapter 2. Finally, we extend the homomorphic capabilities of the DGHV scheme to include more than one operation and test the limits of the current scheme, in order to modify it in Chapter 4 to function for any number of operations.

Our contributions in this chapter are comprised of the continuation of the numerical example in Sections 3.2.1 and 3.3.2 as well as the noise accumulation analysis in Proposition 3.2.1 and Section 3.3.4. We have also refined van Dijk et al.'s results regarding the optimization algorithm in Section 3.3.1.

## 3.1 Overview of Homomorphic Encryption

When discussing homomorphic encryption, it is useful to use ring homomorphisms as a frame of reference. Recall that, given two rings $(\mathcal{M}, \oplus, \otimes)$ and $(\mathcal{C}, +, \cdot)$, a *ring homomorphism* $f : \mathcal{M} \longrightarrow \mathcal{C}$ is a map satisfying

$$\begin{aligned} f(a \oplus b) &= f(a) + f(b), \\ f(a \otimes b) &= f(a) \cdot f(b), \\ f(\mathbb{1}_{\mathcal{M}}) &= \mathbb{1}_{\mathcal{C}}. \end{aligned}$$

Thus, it could make sense to ask if a deterministic encryption function is a ring homomorphism. For example, unpadded RSA partially satisfies the above conditions,

since $Enc_{RSA}(1) = 1$ and, for messages $m_1$ and $m_2$,

$$\begin{aligned} Enc_{RSA}(m_1 \cdot m_2) &= (m_1 \cdot m_2)^e \mod n \\ &= m_1^e \cdot m_2^e \mod n \\ &= (Enc_{RSA}(m_1) \cdot Enc_{RSA}(m_2)) \mod n. \end{aligned}$$

However, $(m_1 + m_2)^e \neq m_1^e + m_2^e \mod n$ and as such, RSA encryption is not a ring homomorphism but rather just a homomorphism of multiplicative groups.

The scheme of interest in our work, the DGHV scheme, uses probabilistic encryption rather than deterministic. Since probabilistic encryption is better represented as a collection of functions, it cannot be defined as a ring homomorphism; thus, we have to relax the definition a bit to accommodate DGHV. In fact, our definition is somewhat closer to saying homomorphic decryption instead. Suppose, in particular, that the ciphertext space $\mathcal{C}$ is not a ring. We define the notion of a homomorphic encryption scheme as follows.

**Definition 3.1.1.** Let $\mathfrak{F}$ be an encryption scheme with decryption function $Dec$, ciphertext space $\mathcal{C}$ and such that the message space $(\mathcal{M}, \oplus, \otimes)$ is a ring. Suppose $\mathcal{C}$ contains a subset $\mathcal{C}'$ that contains all fresh ciphertexts and there exist two functions

$$\begin{aligned} \mathfrak{A} &: \mathcal{C}' \times \mathcal{C}' \longrightarrow \mathcal{C} \\ \mathfrak{M} &: \mathcal{C}' \times \mathcal{C}' \longrightarrow \mathcal{C} \end{aligned}$$

called addition and multiplication, such that the following diagrams commute:

$$\begin{array}{ccc} \mathcal{M} \times \mathcal{M} & \xrightarrow{\oplus} & \mathcal{M} \\ {\scriptstyle (Dec,Dec)}\uparrow & & \uparrow{\scriptstyle Dec} \\ \mathcal{C}' \times \mathcal{C}' & \xrightarrow[\mathfrak{A}]{} & \mathcal{C} \end{array} \qquad \begin{array}{ccc} \mathcal{M} \times \mathcal{M} & \xrightarrow{\otimes} & \mathcal{M} \\ {\scriptstyle (Dec,Dec)}\uparrow & & \uparrow{\scriptstyle Dec} \\ \mathcal{C}' \times \mathcal{C}' & \xrightarrow[\mathfrak{M}]{} & \mathcal{C} \end{array}$$

Then $(\mathfrak{F}, \mathfrak{A}, \mathfrak{M})$ is a *depth-one homomorphic encryption scheme.*

We note that $\mathfrak{A}, \mathfrak{M}$ don't need to be associative, or even composable, in this definition, so $\mathcal{C}$ is not necessarily a ring and $Dec$ and $Enc$ are not ring homomorphisms in the scheme.

Of course, we have to be a little more careful with our definitions. It is always true that, if we define the above functions as

$$\begin{aligned} \mathfrak{A}(\mathrm{c}_1, \mathrm{c}_2) &= Enc(Dec(\mathrm{c}_1) \oplus Dec(\mathrm{c}_2)) \\ \mathfrak{M}(\mathrm{c}_1, \mathrm{c}_2) &= Enc(Dec(\mathrm{c}_1) \otimes Dec(\mathrm{c}_2)), \end{aligned}$$

then $\mathfrak{F}$ is a depth-one homomorphic encryption scheme. However, this is neither useful nor secure. Therefore, we require that our addition and multiplication functions have

two extra properties: the first is that they must be efficiently computable *without* the private key; the second is that the existence of such functions must not compromise the security of the scheme.

Harkening back to our RSA example, we see that it respects one of the commutative diagrams (multiplication) but not the other. We call such schemes *partially homomorphic encryption schemes*.

From the definition, it is not clear if $\mathfrak{A}$ or $\mathfrak{M}$ are composable in any way. We rectify this with the following definition.

**Definition 3.1.2.** An encryption scheme $(\mathfrak{F}, \mathfrak{A}, \mathfrak{M})$ is called *depth-d homomorphic* if the functions $\mathfrak{A}$ and $\mathfrak{M}$ are composable $d$ times.

In the literature, generically such schemes are called *somewhat homomorphic encryption schemes* (SHE) with no specifications on their depth.

Finally, a natural enhancement to a SHE scheme would be to allow us to compose any number of products and sums. A scheme with this property is called a *fully homomorphic encryption scheme* (FHE). We will discuss how the DGHV authors make their scheme fully homomorphic in Chapter 4.

## 3.2 A First Attempt at SHE with the DGHV Scheme

We saw in Chapter 2 that the plaintexts of the DGHV scheme are bits and the ciphertexts are large integers. In order for this encryption scheme to be deemed homomorphic, we must have well-defined addition and multiplication functions as mentioned in Definition 3.1.1. We offer a first intuition as to what these might be, and will refine how these functions are defined later, in Section 3.3.3.

**Proposition 3.2.1.** Let $\mathfrak{F}$ be the DGHV scheme. There exists a subset $\mathcal{C}_0' \subseteq \mathcal{C}$ such that, if we define the functions $\mathfrak{A}_0$ and $\mathfrak{M}_0$ by

$$\mathfrak{A}_0(\mathrm{c}_1, \mathrm{c}_2) = \mathrm{c}_1 + \mathrm{c}_2$$
$$\mathfrak{M}_0(\mathrm{c}_1, \mathrm{c}_2) = \mathrm{c}_1 \cdot \mathrm{c}_2,$$

for all $\mathrm{c}_1, \mathrm{c}_2 \in \mathcal{C}_0'$, then $(\mathfrak{F}, \mathfrak{A}_0, \mathfrak{M}_0)$ is depth-one homomorphic.

**Proof:** Let the subset $\mathcal{C}_0'$ be defined as

$$\mathcal{C}_0' = \left\{ \mathrm{c} \in \mathcal{C} \,\middle|\, |N(\mathrm{c})| + 1 \leq \sqrt{\frac{p}{2}} \right\}.$$

Using the commutative diagrams from Definition 3.1.1, we need only prove that for all $\mathrm{c}_1, \mathrm{c}_2 \in \mathcal{C}'_0$ such that $Dec(\mathrm{c}_i) = \mathtt{m}_i$,

$$Dec(\mathrm{c}_1 + \mathrm{c}_2) = \mathtt{m}_1 \oplus \mathtt{m}_2 \quad \text{and} \quad Dec(\mathrm{c}_1 \cdot \mathrm{c}_2) = \mathtt{m}_1\mathtt{m}_2.$$

We know from (2.1.5) that $\mathrm{c}_1$ and $\mathrm{c}_2$ have the form $\mathrm{c}_1 = \mathtt{m}_1 + k_1 p + N(\mathrm{c}_1)$ and $\mathrm{c}_2 = \mathtt{m}_2 + k_2 p + N(\mathrm{c}_2)$ for some $k_1, k_2 \in \mathbb{N}$. From here, we use the decryption function in (2.1.1) on the sum of the two ciphertexts to obtain

$$Dec(\mathrm{c}_1 + \mathrm{c}_2) = [\mathtt{m}_1 + \mathtt{m}_2 + (k_1 + k_2)p + N(\mathrm{c}_1) + N(\mathrm{c}_2)] \mod p \mod 2.$$

Observe that, for $p > 8$,

$$|N(\mathrm{c}_1) + N(\mathrm{c}_2) + 2| \leq |N(\mathrm{c}_1)| + 1 + |N(\mathrm{c}_2)| + 1 \leq 2\sqrt{\frac{p}{2}} < \frac{p}{2}.$$

Therefore, when evaluating the centered modular reduction, we obtain

$$[\mathtt{m}_1 + \mathtt{m}_2 + (k_1 + k_2)p + N(\mathrm{c}_1) + N(\mathrm{c}_2)] \mod p = \mathtt{m}_1 + \mathtt{m}_2 + N(\mathrm{c}_1) + N(\mathrm{c}_2).$$

What's more, Lemma 2.1.3 tells us that $N(\mathrm{c}_1)$ and $N(\mathrm{c}_2)$ are both even; therefore, their sum is even. Thus,

$$Dec(\mathrm{c}_1 + \mathrm{c}_2) = \mathtt{m}_1 + \mathtt{m}_2 + N(\mathrm{c}_1) + N(\mathrm{c}_2) \mod 2 = \mathtt{m}_1 \oplus \mathtt{m}_2.$$

Next, if we multiply $\mathrm{c}_1$ and $\mathrm{c}_2$, we obtain

$$\begin{aligned} \mathrm{c}_1\mathrm{c}_2 &= \mathtt{m}_1\mathtt{m}_2 + k_1 p\mathrm{c}_2 + k_2 p\mathrm{c}_1 - k_1 k_2 p^2 + \mathtt{m}_1 N(\mathrm{c}_2) + \mathtt{m}_2 N(\mathrm{c}_1) + N(\mathrm{c}_1)N(\mathrm{c}_2) \\ &= (\mathtt{m}_1 + N(\mathrm{c}_1))(\mathtt{m}_2 + N(\mathrm{c}_2)) + (k_1\mathrm{c}_2 + k_2\mathrm{c}_1 - k_1 k_2 p)p. \end{aligned} \tag{3.2.1}$$

Since $\mathrm{c}_1, \mathrm{c}_2 \in \mathcal{C}'$, $(|N(\mathrm{c}_1)| + 1)(|N(\mathrm{c}_2)| + 1) \leq p/2$, so the centered modular reduction removes $(k_1\mathrm{c}_2 + k_2\mathrm{c}_1 - k_1 k_2 p)p$ without adding any other multiple of $p$. Observe also that, since $N(\mathrm{c}_1)$ and $N(\mathrm{c}_2)$ are even, all terms in (3.2.1) that contain one or the other are even. Thus, we obtain

$$\begin{aligned} Dec(\mathrm{c}_1 \cdot \mathrm{c}_2) &= (\mathtt{m}_1 + N(\mathrm{c}_1))(\mathtt{m}_2 + N(\mathrm{c}_2)) + (k_1\mathrm{c}_2 + k_2\mathrm{c}_1 - k_1 k_2 p)p \mod p \mod 2 \\ &= (\mathtt{m}_1 + N(\mathrm{c}_1))(\mathtt{m}_2 + N(\mathrm{c}_2)) \mod 2 \\ &= \mathtt{m}_1\mathtt{m}_2. \end{aligned}$$

∎

Thus, for the DGHV scheme, the homomorphic functions analogous to bit addition and multiplication are integer addition and multiplication, respectively.

We can draw one more conclusion from the analysis done in the above proof.

**Corollary 3.2.2.** Let $N(\mathtt{c}_1)$ be the noise term on the ciphertext $\mathtt{c}_1 \in Enc(\mathtt{m}_1)$ and let $N(\mathtt{c}_2)$ be the noise term of a ciphertext $\mathtt{c}_2 \in Enc(\mathtt{m}_2)$. Then we have

$$|N(\mathtt{c}_1) + N(\mathtt{c}_2)| \leq |N(\mathtt{c}_1 + \mathtt{c}_2)| \leq |N(\mathtt{c}_1)| + |N(\mathtt{c}_2)| + 2$$

for the addition of $\mathtt{c}_1$ and $\mathtt{c}_2$, and

$$|N(\mathtt{c}_1)N(\mathtt{c}_2)| \leq |N(\mathtt{c}_1\mathtt{c}_2)| \leq (|N(\mathtt{c}_1)| + 1)(|N(\mathtt{c}_2)| + 1)$$

for the multiplication of $\mathtt{c}_1$ and $\mathtt{c}_2$.

### 3.2.1 Numerical example continued

Recall Alice's implementation of the DGHV scheme in Section 2.2. We saw that Bob sent Alice five ciphertexts, which Alice decrypted to obtain five plaintexts. For this section, we will be focusing on two of those pairs: $(\mathtt{m}_1, \mathtt{c}_1) = (1, 271326272)$ and $(\mathtt{m}_2, \mathtt{c}_2) = (1, 318596869)$.

Let's say that Alice is looking to evaluate $\mathtt{m}_1 \oplus \mathtt{m}_2$ and $\mathtt{m}_1\mathtt{m}_2$, but that she is too busy to perform the operations herself. Alice was clever when picking her encryption scheme, because she knows of the homomorphic properties of DGHV. She decides to outsource her computations to Eve. Alice sends Eve her ciphertexts $\mathtt{c}_1$ and $\mathtt{c}_2$, asking Eve to send her back the resulting addition and multiplication of the two. Eve willingly obliges, hoping to gain some knowledge about Alice's key or plaintexts along the way.

Eve starts by adding the two ciphertexts:

$$\mathtt{c}_1 + \mathtt{c}_2 = 271326272 + 318596869 = 589923141.$$

She then takes the product of the two ciphertexts:

$$\mathtt{c}_1\mathtt{c}_2 = 271326272 \cdot 318596869 = 86443700736642368.$$

Eve sighs, frustrated. She is no closer to discovering Alice's private key. Suddenly, she realizes two things. The first is that, because the product of two ciphertexts is much larger than the ciphertexts themselves, Eve will always be able to recognize the product of ciphertexts when she eavesdrops on Alice's communications. Secondly, knowing that the plaintexts are bits, Eve therefore knows that the product of two ciphertexts will decrypt to 0 three quarters of the time. Satisfied that she has learned something, Eve returns the results to Alice.

As Eve gives the results back to Alice, she lets slip that the product that she computed will probably decrypt to 0. Panicked, Alice leaves to decrypt what Eve gave her. Her results are

$$Dec(\mathtt{c}_1 + \mathtt{c}_2) = 589923141 \mod p \mod 2 = -192 \mod 2 = 0,$$

which corresponds to $\mathtt{m}_1 \oplus \mathtt{m}_2$, and

$$Dec(\mathrm{c}_1\mathrm{c}_2) = 86443700736642368 \mod p \mod 2 = -49 \mod 2 = 1,$$

which corresponds to $\mathtt{m}_1\mathtt{m}_2$.

Relieved that Eve was in fact wrong about the decryption of the product, Alice nevertheless comes to the same conclusions: it is possible for an attacker to deduce something about her computations, and perhaps even about the result, just by looking at the size of the ciphertexts. Alice resolves to find a way of standardizing the sizes of her ciphertexts (we will see her solution in Section 3.3.2) and to stop outsourcing her computations to Eve.

## 3.3 Optimization – A Successful Attempt at SHE

As we saw in the numerical example above, even though the DGHV scheme in its current form is depth-one homomorphic, a ciphertext that is the result of a multiplication is much larger than a newly generated ciphertext. This is undesirable for two reasons: it is more difficult to work with larger ciphertexts and, more importantly, ciphertexts that lie outside of the range $[0, x_0)$ are recognizable and thus leak information about the underlying plaintexts. As such, we need all ciphertexts, regardless of how many functions have been evaluated on them, to be contained in the interval $[0, x_0)$.

The DGHV authors include in [31] a few optimizations to be implemented when running the encryption scheme. The one we will focus on is an iterated modular reduction algorithm that uses new public key elements to reduce the sizes of ciphertexts as they are being added and/or multiplied. This algorithm ensures that all ciphertexts remain less than $x_0$, without compromising decryption.

### 3.3.1 Optimization algorithm

The optimization algorithm is run when the addition or multiplication of ciphertexts generates a value larger than $x_0$. Recall from (2.1.3) that ciphertexts have the form

$$\mathrm{c} = \mathtt{m} + 2p\sum_{i\in S} q_i + 2\sum_{i\in S} r_i + 2r' - zpq_0 - zr_0.$$

Let $\mathrm{c}_{out}$ be a ciphertext that is the result of an operation on other ciphertexts, such that $\mathrm{c}_{out} > x_0$. The first idea would be to reduce it modulo $x_0$ once more in order to reduce its size. Let $\mathrm{c}'_r$ be the reduced version of $\mathrm{c}_{out}$. Then

$$\mathrm{c}'_r = \mathrm{c}_{out} \mod x_0$$

$$= \mathrm{C}_{out} - \left\lfloor \frac{\mathrm{C}_{out}}{x_0} \right\rceil x_0$$
$$= \mathrm{C}_{out} - pq_0 \left\lfloor \frac{\mathrm{C}_{out}}{x_0} \right\rceil - r_0 \left\lfloor \frac{\mathrm{C}_{out}}{x_0} \right\rceil.$$

When looking at the noise term, we see that

$$|N(\mathrm{C}'_r)| = |N(\mathrm{C}_{out})| + \left| r_0 \left\lfloor \frac{\mathrm{C}_{out}}{x_0} \right\rceil \right|.$$

This noise is too large to decrypt to the right bit. For example, if $\mathrm{C}_{out} = \mathfrak{M}_0(\mathrm{C}_1, \mathrm{C}_2)$ for $\mathrm{C}_1, \mathrm{C}_2 \in Enc(\mathcal{M})$, then $r_0 \left\lfloor \frac{\mathrm{C}_{out}}{x_0} \right\rceil$ could be as large as $r_0 2^\gamma$; thus, $\mathrm{C}'_r$ fails Lemma 2.1.3 and decryption will be incorrect.

To avoid the introduction of too much noise, we use the optimization described by van Dijk et al. in [31]. We create new, much larger near-multiples of $p$ to add to the public key, then reduce the ciphertext modulo each of them in order to reduce its size a little at a time.

Before running the optimization algorithm, we must create the new elements of our public key. These elements are near-multiples of $p$; that is, for $i = 0, \ldots, \gamma$, $x'_i = 2\left(q'_i \cdot p + r'_i\right)$, where $r'_i \in \mathbb{Z} \cap (-2^\rho, 2^\rho)$ (as before) and $q'_i \in \mathbb{Z} \cap \left[\frac{2^{\gamma+i-1}}{p}, \frac{2^{\gamma+i}}{p}\right)$. This gives a vector $x'$ whose $\gamma + 1$ elements are all much larger than the elements of the public key – in fact, $x'_i \in [2^{\gamma+i}, 2^{\gamma+i+1})$. We concatenate $x'$ to our original public key $x$.

One might imagine that it would be possible to obtain $x'_i$ as large as $2^{\gamma+i+1} + 2^\rho$; however, if $x'_i > 2^{\gamma+i+1}$, then the same type of brute-force attack as described in Section 2.4.3 would be possible, giving an advantage to an attacker. Therefore, to maintain the security of the scheme, we must absolutely have $x'_i \in [2^{\gamma+i}, 2^{\gamma+i+1})$.

Once these additional elements are created, the optimization algorithm can be described as a function

$$Opt : \mathcal{C} \longrightarrow \mathcal{C}$$
$$\mathrm{C} \longmapsto \mathfrak{O}(\mathrm{C})$$

and proceeds as follows: we take the ciphertext C, the result of some sequence of additions and multiplications such that $x_0 < \mathrm{C} < 2^{2\gamma}$. We compare C to each element of $x'$ (starting with the largest) until we find a value $M$ such that $x'_M < \mathrm{C}$. Then we reduce the ciphertext by evaluating C modulo $x'_M$, then modulo $x'_{M-1}$, and so on for all remaining elements of $x'$. We will describe this process more algorithmically later, after a numerical example. At the end, we also reduce the result modulo $x_0$.

This last step is not mentioned in [31]; in fact, the last reduction the DGHV authors perform is with $x'_0$. However, it is absolutely essential for an optimized ciphertext $\mathrm{C}'$ to be contained in the same range as a newly encrypted one, that is

we must have $c' \in [0, x_0)$. Based on the construction of the $x'_i$, we have only that $x'_0 \in [2^\gamma, 2^{\gamma+1})$, so it is possible to reduce a ciphertext modulo $x'_0$ and obtain a result that is still larger than $x_0$. This is why we add the last modular reduction by $x_0$, because we would ideally want a result that is indistinguishable from a fresh ciphertext.

We now illustrate the optimization algorithm with a numerical example, after which we will redefine our depth-one homomorphic scheme and impose some new parameter restrictions.

### 3.3.2 Numerical example of optimization

Continuing the example from Sections 2.2 and 3.2.1, we see that Alice has found a solution to her ciphertext size problem. She has decided to keep her computations in-house and she has added much larger near-multiples of $p = 927$ to her public key in order to use the optimization technique that the DGHV authors outline in their research. She adds, for $i = 0, \ldots, 30$, integers of the form $x'_i = 2(pq'_i + r'_i)$, where the $r'_i$ are integers chosen uniformly randomly from $[-7, 7]$, and the $q'_i$ are integers chosen uniformly randomly from

$$\left[\frac{2^{30+i-1}}{927}, \frac{2^{30+i}}{927}\right).$$

These new elements of Alice's public key are

| | | | |
|---|---|---|---|
| 974272371 | 297177587706 | 136445104876458 | 34648527357872728 |
| 2510386456 | 976208369274 | 218165110256920 | 42687506702508348 |
| 7437693658 | 1578743212822 | 446713371246628 | 142655904944831860 |
| 12463340726 | 2872692868266 | 732536561020194 | 160129155860841430 |
| 22270622516 | 6644443607524 | 1385406504954586 | 406035851137403132 |
| 40852183632 | 15787504102378 | 2690370004645432 | 727615533691966432 |
| 81330267128 | 24252752742796 | 5272400324895868 | 1972897472284582960 |
| 223423313354 | 53931505864352 | 12931624117492554 | |

With these new public key elements, Alice looks at the ciphertext that Eve had given her earlier. Recall that $c = 86443700736642368$ was the product of $c_1$ and $c_2$, and is so much bigger than a regular ciphertext that Eve tried to guess what it might decrypt to (though she turned out to be wrong).

Alice uses her new public key to optimize $c$. She programs an optimization algorithm into the distant, secure server that is performing her computations. This way, Eve can intercept the reply but has no knowledge of the intermediate operations. Alice's algorithm begins by comparing the ciphertext to each element, starting from the largest. She knows that $c \mod Z = c$ if $c < Z$, so she is looking to remove all elements that are larger than $c$. Alice's algorithm finds that $x'_{25} = 42687506702508348$

is smaller than $c$, so she first evaluates

$$86443700736642368 \equiv 1068687331625672 \mod 42687506702508348.$$

On this new value, Alice performs modular reduction by each of the $x'_i$ in turn, from $x'_{24}$ down to $x'_0$, as well as one last modular reduction by $x_0$. She sees that the next reduction to produce a new result is $x'_{19} = 732536561020194$:

$$1068687331625672 \equiv 336150770605478 \mod 732536561020194.$$

Alice performs all the required modular reductions, finally obtaining her optimized ciphertext $\mathfrak{O}(c) = 234616167$. Observe that $\mathfrak{O}(c) < x_0$, so it looks just like a "fresh" ciphertext.

Satisfied that Eve can no longer gain information from the size of the ciphertexts, Alice finally checks that $\mathfrak{O}(c)$ decrypts to $\mathtt{m}_1\mathtt{m}_2 = 1$ as expected. She evaluates the decryption function on her ciphertext, obtaining

$$234616167 \ \text{cmod}\ 927 \mod 2 = -117 \mod 2 = 1.$$

She notes that the noise of her optimized ciphertext, while still well within range, is larger than it was before.

Alice now has a fully functioning implementation of the DGHV scheme, capable of encrypting bits to large integers and of performing a single multiplication or addition homomorphically, without having the resulting ciphertext reveal the computation.

### 3.3.3 Depth-one homomorphic DGHV

Now that we have seen that optimization is required in order to keep the DGHV scheme secure, we have redefined our public key to include the larger elements. To our knowledge, no one has yet identified a weakness inherent to adding more near-multiples of $p$ of greater length. Next, we must redefine our addition and multiplication functions, $\mathfrak{A}$ and $\mathfrak{M}$, as well as the subset of small-noise ciphertexts $\mathcal{C}'$, from Proposition 3.2.1. We incorporate the optimization algorithm as follows:

$$\mathfrak{A}(c_1, c_2) = \mathfrak{O}(c_1 + c_2)$$
$$\mathfrak{M}(c_1, c_2) = \mathfrak{O}(c_1 \cdot c_2).$$

Since optimization is a function of the ciphertexts, it introduces more noise while making the ciphertext smaller. The following theorem proposes an upper bound on that additional noise.

**Theorem 3.3.1** (Optimization noise accumulation theorem)**.** Let $\mathrm{c} \in (2^\gamma, 2^{2\gamma})$ be a ciphertext to be optimized and let $\mathrm{c}' = \mathfrak{D}(\mathrm{c}) < 2^\gamma$ be the smaller, optimized version of c. Then

$$|N(\mathrm{c}')| \leq |N(\mathrm{c})| + (3\gamma + 3)\, 2^\rho.$$

We will prove this theorem in Section 3.3.4.

Combining this theorem and Corollary 3.2.2, we obtain an upper bound on the noise of a single operation in the DGHV scheme.

**Corollary 3.3.2.** Let $\mathrm{c}_1, \mathrm{c}_2 \in \mathit{Enc}(\mathcal{M})$ be fresh ciphertexts of the DGHV scheme. For simplicity, let each ciphertext have noise term at most $N(\mathrm{c})$. Then, the noise on one operation satisfies

$$|N(\mathfrak{M}(\mathrm{c}_1, \mathrm{c}_2))| \leq (|N(\mathrm{c})| + 1)^2 + 3(\gamma + 1)2^\rho. \tag{3.3.1}$$

Based on this bound, we can now redefine our restricted subset $\mathcal{C}' \subseteq \mathcal{C}$ as

$$\mathcal{C}' = \left\{ \mathrm{c} \in \mathcal{C} \,\middle|\, (|N(\mathrm{c})| + 1)^2 + 3(\gamma + 1)2^\rho < \frac{p}{2} \right\}$$

The following corollaries help to introduce new constraints on our existing parameters in order to achieve depth-one homomorphic encryption.

**Corollary 3.3.3.** Under the parameter constraints of Table 2.2, we have

$$3(\gamma + 1)2^\rho < \frac{p}{2}.$$

**Proof:** We saw that

$$\begin{aligned} \eta &\geq \rho + \log(\tau + 1) + 5 \quad \text{by (2.4.4) and (2.4.12)} \\ \eta - 2 &> \rho + \log(\gamma + 1) + 3 \quad \text{by (2.4.7)} \\ \eta - 2 &> \rho + \log(\gamma + 1) + \log 3 \\ \frac{p}{2} &\geq 2^{\eta - 2} > 3(\gamma + 1)2^\rho, \end{aligned}$$

as required. ∎

**Corollary 3.3.4.** If the parameter constraints from Table 2.2 and the additional constraint

$$\rho' < \frac{\eta}{2} - 5 \tag{3.3.2}$$

are met, then the DGHV scheme is depth-one homomorphic.

**Proof:** We are looking to prove that under the above conditions we have

$$(|N(\mathrm{c})| + 1)^2 < \frac{p}{2} - 3(\gamma + 1)2^{\rho}.$$

We first recall (2.4.5), which put an upper bound on the noise on a fresh ciphertext:

$$|N(\mathrm{c})| + 1 < 2^{\rho'+3}.$$

Therefore, we want to prove that

$$2^{2\rho'+6} < 2^{\eta-2} - 2^{\log 3+\log(\gamma+1)+\rho}.$$

Let's temporarily relabel the exponents to better illustrate the inequality. Let $k = 2\rho' + 6$, $\ell = \eta - 2$, and $m = \log 3 + \log(\gamma + 1) + \rho$. We are looking to prove $2^k < 2^\ell - 2^m$, which is equivalent to

$$2^{k-m} < 2^{\ell-m} - 1,$$

since $m < \ell$ by Corollary 3.3.3.

We will prove that $k < \ell - 1$, since that implies $k - m < \ell - m - 1$ and

$$2^{k-m} < 2^{\ell-m-1} < 2^{\ell-m} - 1$$

for $\ell - m > 1$, which we have from Corollary 3.3.3.

Therefore, we prove $2\rho' + 6 < \eta - 3$, which we see immediately from (3.3.2), and the DGHV scheme in depth-one homomorphic. ∎

The DGHV authors refer to two more closely related concepts inherent to the security of their homomorphic scheme: *compactness* [31, Def.2.4] and *circuit privacy* [31, App.C]. They define an evaluation on ciphertexts as compact if the output of the evaluation is bounded in length by a polynomial in the security parameter $\lambda$, regardless of the number of operations involved. They then define circuit privacy informally to mean that the ciphertext generated by some evaluation on ciphertexts does not reveal information about what operations were performed. Gentry's PhD thesis defines the concept in a little more detail [14, Ch.20, Def.2.1.6] and points to work of Sander, Young, and Yung [29] and Beaver [4] to claim there exists a statistically circuit-private version of any SHE scheme. In other words, the distribution of $Enc(\mathcal{M})$ and that of the outputs of compositions of $\mathfrak{A}$ and $\mathfrak{M}$ are statistically indistinguishable.

However, the DGHV scheme as defined here is not proven to have statistical circuit privacy. Therefore, we relax the definition here.

**Definition 3.3.5.** A (somewhat or fully) homomorphic encryption scheme is said to have *minimal circuit privacy* if the outputs of $\mathfrak{A}$ and $\mathfrak{M}$ are in the same range as outputs of *Enc*.

With the above constraints, we have attained minimal circuit privacy, so we have now transformed the DGHV scheme into a depth-one homomorphic scheme. After proving the optimization noise accumulation theorem in the next section, we will move on to depth-$d$ homomorphic encryption in Section 3.4.

### 3.3.4 Proof of the optimization noise accumulation theorem

In this section, we analyze the optimization algorithm carefully and use this analysis to prove Theorem 3.3.1.

We know that for ciphertexts $\mathrm{c}_1, \mathrm{c}_2 \in Enc(\mathcal{M})$ we have

$$\mathrm{c}_1 < 2^{\gamma} \quad \text{and} \quad \mathrm{c}_2 < 2^{\gamma},$$

hence, we obtain

$$\mathrm{c}_1 + \mathrm{c}_2 < 2^{\gamma+1} \quad \text{and} \quad \mathrm{c}_1 \cdot \mathrm{c}_2 < 2^{2\gamma}.$$

Let $\mathrm{c}$ represent the result of a sequence of one or more operations such that

$$x_0 < \mathrm{c} < 2^{2\gamma} \leq x'_{\gamma}.$$

Then let $M$, $0 \leq M \leq \gamma - 1$ be such that

$$x'_M \leq \mathrm{c} < x'_{M+1}.$$

Let $\mathrm{c}_{M+1} = \mathrm{c}$. In each step of the optimization process, we define, for $i$ from $M$ down to 0,

$$\mathrm{c}_i = \mathrm{c}_{i+1} - a_i x'_i,$$

where $a_i = \left\lfloor \frac{\mathrm{c}_{i+1}}{x'_i} \right\rfloor \in \mathbb{N}$. We call the $a_i$ the *coefficients of the modular reductions*.

Therefore, for each $i = 0, \ldots, M$, we have

$$\mathrm{c}_i = \mathrm{c} - a_M x'_M - a_{M-1} x'_{M-1} - \ldots - a_i x'_i$$

and

$$\mathrm{c} = a_M x'_M + a_{M-1} x'_{M-1} + \ldots + a_0 x'_0 + \mathrm{c}_0. \tag{3.3.3}$$

Taking the above two equations together, we obtain

$$\mathrm{c}_i = a_{i-1} x'_{i-1} + \ldots + a_0 x'_0 + \mathrm{c}_0. \tag{3.3.4}$$

Note that $\mathfrak{D}(\mathrm{c}) = \mathrm{c}' = \mathrm{c}_0$.

In particular, observe that $N(\mathrm{C}') = \sum_{i=0}^{M} a_i r'_i + N(\mathrm{C})$. Let us therefore characterize the possible sequences $(a_0, \ldots, a_M)$ and the maximum possible value of this sum in order to calculate a bound on the noise created by these modular reductions. We explore allowed values for the $a_i$ in the next few lemmas.

**Lemma 3.3.6.** The coefficients of the successive modular reductions satisfy

$$0 \leq a_i \leq 3 \quad \forall\, i = 0, \ldots, M.$$

**Proof:** We saw in Section 3.3.1 that $x'_i \in [2^{\gamma+i}, 2^{\gamma+i+1})$ for each $i = 0, \ldots, M$, so

$$\frac{x'_{i+1}}{x'_i} \leq \frac{2^{\gamma+i+2}}{2^{\gamma+i}} = 2^2 = 4.$$

Next, since for every $i = 0, \ldots, M$ we have, by definition above, that $\mathrm{C}_i = \mathrm{C}_{i+1} \bmod x'_i$, we see easily that $\mathrm{C}_i < x'_i$. Finally, we use the definition of $a_i$ from above:

$$a_i = \left\lfloor \frac{\mathrm{C}_{i+1}}{x'_i} \right\rfloor \leq \frac{\mathrm{C}_{i+1}}{x'_i} < \frac{x'_{i+1}}{x'_i} \leq 4;$$

since the $a_i$ are integers, for every $i = 0, \ldots, M$, we have that $0 \leq a_i \leq 3$. ∎

We are looking to bound the $a_i$ more tightly, which leads to the following results.

**Lemma 3.3.7.** Let $i = 0, \ldots, M-1$. Then, for any $0 \leq n \leq M - i$,

$$a_{i+n} x'_{i+n} + \ldots + a_i x'_i < x'_{i+n+1}.$$

**Proof:** Note that, by definition, for each $k = 0, \ldots, M$ we have $\mathrm{C}_k < x'_k$. Using this and equation (3.3.4), we obtain, for any $0 \leq n \leq M - i$,

$$\sum_{j=i}^{i+n} a_j x'_j \leq \sum_{j=0}^{i+n} a_j x'_j + \mathrm{C}_0 = \mathrm{C}_{i+n+1} < x'_{i+n+1}$$

as required. ∎

In order to tighten our bounds further we will construct an upper bound on the $a_i$ that is only dependent on $n$, rather than on any of the $x'_i$. The next lemma was produced from a recursive relation we saw when analyzing the optimization process. Indeed, by definition, using (3.3.3), we have

$$\mathrm{C} = a_M x'_M + a_{M-1} x'_{M-1} + \ldots + a_2 x'_2 + \underbrace{a_1 x'_1 + \overbrace{a_0 x'_0 + \mathrm{C}_0}^{<x'_1}}_{<x'_2}.$$

This brought to light the following:

$$\begin{aligned} a_0x'_0(a_1+1) &= a_1a_0x'_0 + a_0x'_0 < a_1x'_1 + a_0x'_0 \\ a_0x'_0(a_1+1)(a_2+1) &= a_2(a_0x'_0(a_1+1)) + a_0x'_0(a_1+1) < a_2x'_2 + a_1x'_1 + a_0x'_0 \\ &\vdots \\ a_0x'_0(a_1+1)(a_2+1)\cdots(a_{i+n}+1) &< a_0x'_0 + a_1x'_1 + \ldots + a_{i+n}x'_{i+n}. \end{aligned}$$

We now prove this recursive formula by induction on $n$.

**Lemma 3.3.8.** For $i = 0, \ldots, M-1$ and $1 \leq n \leq M-i$, we have

$$a_ix'_i(a_{i+1}+1)\ldots(a_{i+n}+1) < a_{i+n}x'_{i+n} + \ldots + a_ix'_i.$$

**Proof:** When $n = 0$, we see that the inequality is not strict; therefore, we begin at $n = 1$. We have that

$$a_ix'_i(a_{i+1}+1) = a_{i+1}a_ix'_i + a_ix'_i.$$

From Lemma 3.3.7, we have $a_ix'_i < x'_{i+1}$; thus,

$$a_{i+1}a_ix'_i + a_ix'_i < a_{i+1}x'_{i+1} + a_ix'_i,$$

whence $a_ix'_i(a_{i+1}+1) < a_{i+1}x'_{i+1} + a_ix'_i$. So the statement holds for $n = 1$.

Now suppose that the statement holds for some $1 \leq n \leq M-i-1$. Therefore, we have

$$\begin{aligned} a_ix'_i(a_{i+1}+1)\cdots(a_{i+n}+1)(a_{i+n+1}+1) & \\ &< (a_{i+n+1}+1)\left[a_{i+n}x'_{i+n} + \ldots + a_ix'_i\right] \\ &= a_{i+n+1}(a_{i+n}x'_{i+n} + \ldots + a_ix'_i) + a_{i+n}x'_{i+n} + \ldots + a_ix'_i \\ &< a_{i+n+1}x'_{i+n+1} + a_{i+n}x'_{i+n} + \ldots + a_ix'_i \quad \text{by Lemma 3.3.7.} \end{aligned}$$

So the statement holds for $n+1$, completing the proof. ∎

If we combine Lemmas 3.3.7 and 3.3.8, we obtain

$$a_ix'_i(a_{i+1}+1)\ldots(a_{i+n-1}+1)(a_{i+n}+1) < x'_{i+n+1},$$

which can be rewritten more compactly as

$$a_i\prod_{j=i+1}^{i+n}(a_j+1) < \frac{x'_{i+n+1}}{x'_i}.$$

Using $2^{\gamma+i} \leq x'_i$ and $x'_{i+n+1} \leq 2^{\gamma+i+n+2}$, we obtain a numerical bound on the product, dependent on $n$, proving the following proposition.

**Proposition 3.3.9.** For $i = 0, \ldots, M-1$ and $1 \leq n \leq M - i$

$$a_i \prod_{j=i+1}^{i+n} (a_j + 1) < \frac{2^{\gamma+i+n+2}}{2^{\gamma+i}} = 2^{n+2}.$$

**Example 3.3.10.** For $n = 1$, this gives

$$a_i \prod_{j=i+1}^{i+1} (a_j + 1) = a_i(a_{i+1} + 1) < 2^3 = 8.$$

This means that, when analyzing the sequence of $a_i$, for any $i$,

$$(a_i, a_{i+1}) \notin \{(2,3), (3,2), (3,3)\}.$$

**Example 3.3.11.** For $n = 2$, we get

$$a_i \prod_{j=i+1}^{i+2} (a_j + 1) = a_i(a_{i+1} + 1)(a_{i+2} + 1) < 2^4 = 16.$$

This implies that, for any $i$,

$$(a_i, a_{i+1}, a_{i+2}) \notin \{(2,1,3), (2,2,2), (3,1,2), (3,1,3)\}.$$

Similarly, taking $n = 3$ gives

$$(a_i, a_{i+1}, a_{i+2}, a_{i+3}) \notin \{(2,1,1,3), (2,1,2,2), (2,2,1,2), (3,1,1,2), (3,1,1,3)\},$$

and with $n = 4$ we obtain

$$\begin{aligned}(a_i, a_{i+1}, a_{i+2}, a_{i+3}, a_{i+4}) \notin \{&(2,1,1,1,3), (2,1,1,2,2), (2,1,2,1,2),\\ &(2,2,1,1,2), (3,1,1,1,2), (3,1,1,1,3)\}.\end{aligned}$$

We begin to notice a pattern with respect to the disallowed sequences and their corresponding sums; we will prove the validity of that pattern next.

**Lemma 3.3.12.** Suppose $I = \{i, i+1, \ldots, i+n\}$ is a sequence of consecutive indices between 0 and $M$ such that $a_k \neq 0 \ \forall k \in I$. Then

$$\sum_{k \in I} a_k \leq n + 3.$$

**Proof:** Let $\{a_i, \ldots, a_{i+n}\}$ be the sequence in question. We have, from Lemma 3.3.6 and our hypothesis, that $a_i \in \{1, 2, 3\}$, so we will use the result from Proposition 3.3.9 in the following three cases:

**Case 3.1.** $a_i = 3$.

Suppose that at least one of the $a_j$'s is at least 2 and the rest are at least 1. Then we have

$$a_i \prod_{j \in I \setminus \{i\}} (a_j + 1) \geq 3 \cdot 3 \cdot 2^{n-1} = 9 \cdot 2^{n-1} > 2^{n+2},$$

which contradicts the Proposition. Note, in contrast, that if $a_j = 1$ for all $i + 1 \leq j \leq i + n$, then

$$a_i \prod_{j \in I \setminus \{i\}} (a_j + 1) = 2^n \cdot 3 < 2^{n+2},$$

so we cannot exclude a sequence of the form $3, 1, 1, 1, \ldots$ with $n$ ones.

Therefore, if $a_i = 3$, the rest of the integers in the nonzero sequence must be ones. The sum of this sequence is then

$$\sum_{k \in I} a_k = 3 + (n)(1) = n + 3.$$

**Case 3.2.** $a_i = 2$.

First, suppose that for at least one $j \in \{i + 1, \ldots, i + n\}$ we have $a_j = 3$. Then

$$a_i \prod_{j \in I \setminus \{i\}} (a_j + 1) \geq 2 \cdot 4 \cdot 2^{n-1} = 2^{n+2},$$

which contradicts the Proposition. So there are no threes in the sequence.
Similarly, if at least two of the $a_j$ are equal to 2, then

$$a_i \prod_{j \in I \setminus \{i\}} (a_j + 1) > 2 \cdot 3^2 \cdot 2^{n-2} = 9 \cdot 2^{n-1} > 2^{n+2},$$

which is again a contradiction.
In contrast, note that if at most one of the $a_j$'s is 2, then we obtain

$$a_i \prod_{j \in I \setminus \{i\}} (a_j + 1) \leq 2 \cdot 3 \cdot 2^{n-1} = 3 \cdot 2^n < 2^{n+2}$$

as required.

Therefore, if $a_i = 2$, the rest of the $a_j$ are either all ones, or there is exactly one 2 amongst them. The sum of this sequence is then

$$\sum_{k \in I} a_k \leq 2 + 2 + (n - 1)(1) = n + 3.$$

**Case 3.3.** $a_i = 1$.

Let $f \geq 1$ be maximal such that

$$a_i = a_{i+1} = \ldots = a_{i+f-1} = 1.$$

If $f = n + 1$, then

$$\sum_{k \in I} a_k = n + 1 < n + 3.$$

If not, then $a_{i+f} \in \{2, 3\}$, so then, by the previous cases, we get

$$\sum_{k \in I} a_k = \sum_{k=i}^{i+f-1} a_k + \sum_{k=i+f}^{i+n} a_k \leq f + (n - f + 1) + 2 = n + 3.$$

In all cases, the sum of the elements of the sequence is no greater than $n + 3$. ∎

We next state a proposition that ties in the previous Lemmas to the value $M$ defined at the beginning.

**Proposition 3.3.13.** Suppose $x'_M \leq \mathrm{c} < x'_{M+1}$. The sum of the sequence of coefficients of the modular reductions of c is bounded above by $\left(\frac{3}{2}M + 3\right)$.

**Proof:** If all coefficients in the sequence are non-zero, in other words $a_i \neq 0$ for all $i = 0, \ldots, M$, then by Lemma 3.3.12 we have

$$\sum_{i=0}^{M} a_i \leq M + 3.$$

Suppose then that some of the $a_i$ are zero. The sequence of $a_i$ can be separated into $m$ blocks $b_1, \ldots, b_m$, where each block is a maximal consecutive subsequence of indices such that for all $k \in b_j$ we have $a_k \neq 0$, and such that block $b_j$ has $\gamma_j$ terms. There are therefore at least $m - 1$ zeros in the sequence, so by comparing to the total number of terms, we obtain

$$\gamma_1 + \gamma_2 + \ldots + \gamma_m + m - 1 \leq M + 1,$$

that is,

$$\sum_{j=1}^{m} \gamma_j \leq M - m + 2. \tag{3.3.5}$$

Since each $\gamma_j \geq 1$, we have that (3.3.5) implies $m \leq M - m + 2$, so

$$m \leq \frac{M}{2} + 1, \tag{3.3.6}$$

thus giving a bound on $m$.
From Lemma 3.3.12 we know that, for a given block $b_j$ of $\gamma_j$ terms, we have

$$\sum_{k \in b_j} a_k \leq \gamma_j + 2.$$

This then gives

$$\begin{aligned}
\sum_{i=0}^{M} a_i &\leq (\gamma_1 + 2) + \ldots + (\gamma_m + 2) \\
&= \sum_{j=1}^{m} \gamma_j + 2m \\
&\leq M - m + 2 + 2m \quad \text{by (3.3.5)} \\
&= M + m + 2 \\
&\leq M + \frac{M}{2} + 3 \quad \text{by (3.3.6)}
\end{aligned}$$

as required. ∎

**Proof:** (**of theorem 3.3.1**)
Since

$$x'_i = 2q'_i \cdot p + 2r'_i \quad \forall i,$$

the upper bound on the noise of each $x'_i$ is $2^{\rho+1}$. Therefore, by setting $M = \gamma - 1$ and simplifying Proposition 3.3.13, the noise on the optimized ciphertext $\mathrm{C}'$ satisfies

$$|N(\mathrm{C}')| \leq |N(\mathrm{C})| + \left(\frac{3}{2}(\gamma - 1) + 3\right) 2^{\rho+1} = |N(\mathrm{C})| + 3(\gamma + 1)2^{\rho}.$$

∎

We saw in Lemma 3.3.12 that in many cases, the noise introduced by the optimization process would be bounded by a smaller term than what we have just presented. It is therefore natural to ask if the upper bound on the sum of the coefficients could ever occur in practice. We see with the next theorem that it is in fact possible to attain the bound.

**Theorem 3.3.14.** For $\gamma > 8$, the bound from Proposition 3.3.13 is the optimal bound.

**Proof:** Let c be the ciphertext to be optimized, let $\{x'_0, \ldots, x'_M\}$ be the sequence of integers with which we will be optimizing c, and let the sequence of modular reduction coefficients be denoted by $\{a_0, \ldots, a_M\}$.

There are two sequences that meet the bound from the Proposition:

$$\{3, 0, 3, 0, \ldots, 3\} \text{ for } M \text{ even, and}$$
$$\{1, 3, 0, 3, 0, \ldots, 3\} \text{ for } M \text{ odd.}$$

Note that the sum of the second sequence is $\frac{3}{2}M + \frac{5}{2} = \left\lfloor \frac{3}{2}M + 3 \right\rfloor$; what's more, if the second sequence exists, we can easily modify it to obtain the first. Therefore, suppose $M = 2k + 1$ for some $k \in \mathbb{Z}$.

If c were a ciphertext giving the sequence of modular reduction coefficients $\{1, 3, 0, 3, 0, \ldots, 3\}$, then by (3.3.3) it must have the form

$$c = x'_{2k+1} + 3x'_{2k} + 0x'_{2k-1} + 3x'_{2k-2} + \ldots + 3x'_0 + c_0.$$

We know, from Section 3.3.1, that for all $i = 0, \ldots, M$, we have

$$x'_i = 2\left(pq'_i + r'_i\right) \quad \text{where } q'_i \in \left[\frac{2^{\gamma+i-1}}{p}, \frac{2^{\gamma+i}}{p}\right).$$

Therefore, for the sake of our proof let $r'_i = 0$ for all $i$ and let our sequence of $x'_i$ be defined, for $m = 0, \ldots, k$, as

$$x'_{2m} = 2p\left(\left\lfloor \frac{2^{\gamma+2m-1}}{p} \right\rfloor + 1\right), \qquad x'_{2m+1} = 2p\left\lfloor \frac{2^{\gamma+2m+1}}{p} \right\rfloor.$$

Note that, for any $m$,

$$x'_{2m+1} > 2p\left(\frac{2^{\gamma+2m+1}}{p} - 1\right) = 2^{\gamma+2m+2} - 2p \geq 2^{\gamma+2m+2} - 2^{\eta+1}. \tag{3.3.7}$$

We will also choose $c_0$ such that

$$0 \leq c_0 \leq 2^{\gamma-1} - 6p(k+1). \tag{3.3.8}$$

**Claim**: $2^{\gamma-1} - 6p(k+1) > 0$.
By (2.4.11), we have $\eta < \sqrt{\gamma}$, so $p < 2^{\eta} < 2^{\sqrt{\gamma}}$. What's more, since $k = \frac{M-1}{2}$ and $M \leq \gamma - 1$, we have that $k + 1 \leq \gamma/2$, so

$$6p(k+1) < 6 \cdot 2^{\sqrt{\gamma}}\left(\frac{\gamma}{2}\right),$$

which is less than $2^{\gamma-1}$ for $\gamma > 8$, [32], which holds for any $\lambda \geq 1$ with parameter constraints from Chapter 2.

What remains to prove is that our $\mathrm{c}$ and choices of $x'_i$ give the required sequence of modular reduction coefficients. Recall, from (3.3.4), that

$$\mathrm{c}_i = a_{i-1}x'_{i-1} + \ldots + a_0 x'_0 + \mathrm{c}_0.$$

We will prove that for $i = 0, \ldots, 2k+1$, $\mathrm{c}_i < x'_i$. Since for every $m = 0, \ldots, k$ $a_{2m+1} = 0$, we have $\mathrm{c}_{2m} = \mathrm{c}_{2m-1}$. Therefore, it suffices to show that for all $m = 0, \ldots, k$, $\mathrm{c}_{2m+1} < x'_{2m+1}$.

First note that $\sqrt{\gamma} < \gamma - 2$ for $\gamma \geq 4$, so $\eta < \gamma - 2$. We then have that

$$\begin{aligned}
\mathrm{c}_{2m+1} &= 3x'_{2m} + 0x'_{2m-1} + 3x'_{2m-2} + \ldots + 3x'_0 + \mathrm{c}_0 \\
&= 3\sum_{j=0}^{m} x'_{2j} + \mathrm{c}_0 \\
&\leq 3\sum_{j=0}^{m} 2p\left(\frac{2^{\gamma+2j-1}}{p} + 1\right) + \mathrm{c}_0 \\
&= 6\sum_{j=0}^{m} 2^{\gamma+2j-1} + 6p(k+1) + \mathrm{c}_0 \\
&\leq 6 \cdot 2^{\gamma-1}\sum_{j=0}^{m} 4^j + 2^{\gamma-1} \quad \text{by (3.3.8)} \\
&= 3 \cdot 2^{\gamma}\left(\frac{4^{m+1}-1}{3}\right) + 2^{\gamma-1} \\
&= 2^{\gamma+2m+2} - 2^{\gamma} + 2^{\gamma-1} \\
&= 2^{\gamma+2m+2} - 2^{\gamma-1} \\
&\leq 2^{\gamma+2m+2} - 2^{\eta+1} \\
&< x'_{2m+1} \quad \text{by (3.3.7)}.
\end{aligned}$$

Thus, we have shown that it is possible to construct a sequence of $x'_i$ and a ciphertext $\mathrm{c}$ that generate the sequence of modular reduction coefficients $\{1, 3, 0, 3, \ldots, 3\}$; as such, the bound

$$\sum_{i=0}^{M} a_i \leq \frac{3}{2}M + 3$$

is the optimal bound. ∎

## 3.4 Depth-$d$ Homomorphic Encryption

Recall Definition 3.1.2 which stated that an encryption scheme is depth-$d$ homomorphic if the addition and multiplication functions are composable $d$ times. We saw in Corollary 3.3.4 that only one additional constraint was needed on the parameters of the DGHV scheme to make it depth-one homomorphic. We will now first examine the maximum depth obtainable under the current constraints, followed by the adjustments necessary to attain larger $d$.

We must begin by establishing the difference in noise accumulation between the addition and multiplication of ciphertexts. We see immediately that the noise accumulates more quickly when ciphertexts are multiplied rather than added, but how many additions can one perform before the noise accumulation is comparable to that of a single multiplication?

Our first intuition is of course that the multiplication of two $\gamma$-bit numbers is similar to $2^\gamma$ additions of $\gamma$-bit numbers. While this is true, clearly the latter is extremely inefficient and would dramatically increase the noise through frequent optimization. On the subject of optimization during addition, we offer the following corollary.

**Corollary 3.4.1.** Let $\mathrm{c}_1, \mathrm{c}_2 \in Enc(\mathcal{M})$ each have noise term at most $N(\mathrm{c})$. Then the noise on their sum satisfies

$$|N(\mathfrak{A}(\mathrm{c}_1, \mathrm{c}_2))| \leq 2(|N(\mathrm{c})| + 1) + 3 \cdot 2^{\rho+1}.$$

**Proof:** We saw in Section 3.3.4 that since $\mathrm{c}_1, \mathrm{c}_2$ are bounded above by $2^\gamma$, their sum is naturally bounded above by $2^{\gamma+1}$. Recall from Section 3.3.1 that, for $i = 0, \ldots, \gamma$, the public key elements created for the optimization algorithm are $x'_i \in [2^{\gamma+i}, 2^{\gamma+i+1})$. Therefore, we see that at worst, the optimization of the sum of two ciphertexts involves a reduction modulo $x'_0$. Using the results from Theorem 3.3.1 together with those from Corollary 3.2.2, we obtain the desired result. ∎

Extending the above corollary to be able to perform any number of additions naturally modifies the noise term in the following manner.

**Corollary 3.4.2.** Let $\mathrm{c}_1, \ldots, \mathrm{c}_w \in Enc(\mathcal{M})$ each have noise term at most $N(\mathrm{c})$. Then the noise on their sum satisfies

$$|N(\mathfrak{A}(\mathrm{c}_1, \ldots \mathrm{c}_w))| \leq w(|N(\mathrm{c})| + 1) + 3(w-1)2^{\rho+1}.$$

Recall (3.3.1), which established the noise accumulation of one multiplication:

$$|N(\mathfrak{M}(\mathrm{c}_1, \mathrm{c}_2))| \leq (|N(\mathrm{c})| + 1)^2 + 3(\gamma+1)2^\rho.$$

We observe that adding $\gamma/2$ ciphertexts gives us comparable noise accumulation results from the optimization portion of the two equations. However, we see from Table 2.2 that $\gamma < \tau < 2^{\rho'-\rho} < 2^{\rho'+3}$ (the latter being the upper bound on $|N(\mathrm{C})|+1$ as per (2.4.5)), so we could perform many more additions while still remaining below the noise term generated by a single multiplication. While we wouldn't be able to perform as many as $2^{\rho'+3}$ additions, even taking a slightly smaller amount of ciphertexts, say $2^{\rho'+2}$, yields

$$\begin{aligned}
|N(\mathfrak{A}(\mathrm{C}_1,\ldots,\mathrm{C}_{2^{\rho'+2}}))| &\leq 2^{\rho'+2}(|N(\mathrm{C})|+1) + 3(2^{\rho'+2}-1)2^{\rho+1} \\
&\leq 2^{\rho'+2}\cdot 2^{\rho'+3} + 2^{\rho'+\rho+5} - 2^{\rho+2} \\
&\leq 2^{2\rho'+5} + 2^{2\rho'+5} - 2^{\rho+2} \quad \text{by (2.4.12)} \\
&\leq 2^{2\rho'+6},
\end{aligned}$$

which is the maximum value of the square of the noise term. For the purpose of our analysis, we state

$$2^{\rho'+2} \text{ additions } \mathfrak{A} \equiv 1 \text{ multiplication } \mathfrak{M}.$$

In Chapter 4, we will be referring to these functions as circuits, and we will say that the *total noise on a circuit* of $2^{\rho'+2}$ additions is equivalent to the total noise on a circuit of one multiplication.

We are thus analyzing the multiplicative depth of the DGHV scheme, taking $2^{\rho'+2}$ additions to be equivalent to one multiplication. We know from Section 3.3 that when two ciphertexts are multiplied, the result must then be optimized. Thus, performing $d$ multiplications also requires $d$ optimizations. Extending the results from Section 3.3.3, we obtain the following corollary.

**Corollary 3.4.3.** Let $\mathrm{C}_1, \mathrm{C}_2, \ldots, \mathrm{C}_{d+1} \in Enc(\mathcal{M})$ be fresh ciphertexts and suppose each have noise term at most $N(\mathrm{C})$. Then the noise on the result of $d$ multiplications satisfies

$$|N(\mathfrak{M}(\mathrm{C}_1, \mathrm{C}_2, \ldots, \mathrm{C}_{d+1}))| \leq (|N(\mathrm{C})|+1)^{d+1} + 3d(\gamma+1)2^{\rho}.$$

In Chapter 4, we will refer to this as the *total noise on the circuit* of $d$ multiplications.

Therefore, for depth-$d$ homomorphic DGHV, we define the restricted set of ciphertexts to be

$$\mathcal{C}'_d = \left\{ \mathrm{C} \in \mathcal{C} \,\middle|\, (|N(\mathrm{C})|+1)^{d+1} + 3d(\gamma+1)2^{\rho} < \frac{p}{2} \right\} \tag{3.4.1}$$

In terms of constraints, we know that $|N(\mathrm{C})| \leq 2^{\rho'+3}$, so we see that for the DGHV scheme to be depth-$d$ homomorphic for any choice of $p \in [2^{\eta-1}, 2^{\eta}]$, we must have

$$(2^{\rho'+3})^{d+1} + 3d(\gamma+1)2^{\rho} < 2^{\eta-2}.$$

To find the maximum $d$ that allows this inequality to hold, we will employ the same method as before; that is, we look only at the optimization portion of the inequality first.

**Lemma 3.4.4.** For all $d \geq 1$, we have that the noise accumulation from $d$ iterations of the optimization process is less than the maximum noise accumulation from $d$ multiplications; that is,

$$3d(\gamma+1)2^{\rho} < (2^{\rho'+3})^{d+1}.$$

**Proof:** We know that

$$\begin{aligned} 3(\gamma+1)2^{\rho} &= 3 \cdot 2^{\log(\gamma+1)}2^{\rho} \\ &\leq 3 \cdot 2^{\log(\tau+1)}2^{\rho} \quad \text{by (2.4.7)} \\ &\leq 3 \cdot 2^{\rho'-\rho}2^{\rho} \quad \text{by (2.4.12)} \\ &= 3 \cdot 2^{\rho'} \\ &\leq 2^{\rho'+2}. \end{aligned}$$

Therefore, we obtain

$$\begin{aligned} 3d(\gamma+1)2^{\rho} &\leq d2^{\rho'+2} \\ &< (2^{\rho'+2})^{d} \cdot 2^{\rho'+2} \quad \text{since } d < a^{d} \ \ \forall\, a > 1 \\ &< (2^{\rho'+3})^{d+1} \end{aligned}$$

as required. ∎

Using the results of this lemma, we see that, if

$$(2^{\rho'+3})^{d+1} < 2^{\eta-3}, \tag{3.4.2}$$

then $3d(\gamma+1)2^{\rho}$ is also bounded above by $2^{\eta-3}$ and the sum of the two will satisfy

$$(2^{\rho'+3})^{d+1} + 3d(\gamma+1)2^{\rho} < 2^{\eta-2}.$$

Therefore, we want (3.4.2) to be true, which we rewrite as $(\rho'+3)(d+1) < \eta-3$ and then as an upper bound on the depth as follows:

$$d < \frac{\eta-3}{\rho'+3} - 1. \tag{3.4.3}$$

When looking at this bound, we see that in order to increase $d$, we must either increase $\eta$ or decrease $\rho'$. The latter poses a security risk, as we recall from Section 2.4.4 that $\rho'$ is the parameter that makes the noise of the public key elements more

uniformly distributed. Thus, we must increase $\eta$ in order to accommodate a larger depth.

Unfortunately, from (2.4.13), we see that an increase in $\eta$ implies also increasing $\gamma$ and $\tau$ by a factor of $\lambda$ (as per (2.4.7) and (2.4.11)). In turn, because of (2.4.12) we must also increase $\rho'$, albeit only by a logarithmic factor of $\tau$. Even so, since (3.4.3) depends in part on $\rho'$, an increase in $\eta$ does not immediately translate to an increase in $d$.

Observe that (3.4.3) clearly implies that once we have chosen $\eta$ and $\rho'$ for our scheme, the depth $d$ is fixed and represents the maximum depth of any circuit that the somewhat homomorphic DGHV scheme can accommodate. This poses a problem since it is not always possible to know the needed depth in advance.

The authors of the DGHV scheme offer a slightly different theorem relating to allowed depth of their scheme. They define in [31, Lem.3.5] the depth $d$ to be the number of multiplications performed on ciphertexts, but they also frame their definition differently. One can write multiplications and additions of ciphertexts algebraically, giving a polynomial in the inputs. Under this frame of reference, the depth corresponds to one less than the degree of the polynomial. Following this, they define $a$ to be the number of additions performed on ciphertexts, then state that their encryption scheme's depth satisfies

$$d \leq \frac{\eta - 4 - \log a}{\rho' + 2}.$$

We note that the main differences between their formula and ours lie in the assumptions made: van Dijk et al.'s formula does not take into account the noise accumulation due to the optimization process, but it does include the potential noise accumulation from repeated additions. What's more, the DGHV authors state that the noise on a fresh ciphertext is bounded above by $2^{\rho'+2}$. We cannot justify this bound without additional assumptions on the parameters, so we maintain our upper bound of $2^{\rho'+3}$.

Despite the differences in the two formulas, the same conclusions as above may be drawn: raising $d$ creates a circular argument with respect to modifying all the other parameters. Therefore, we require a modification to some part of the scheme in order to be able to augment $d$ without issue. We will discuss this in Chapter 4.

### Alice increases her depth

Alice looks at her parameter set from Section 2.4.5 to determine the maximum depth her implementation of the DGHV scheme can handle. She calculates

$$d < \frac{\eta - 3}{\rho' + 3} - 1 = \frac{30 - 3}{24 + 3} - 1 = 0$$

and quickly realizes that her original setup is far too constrained to be able to compose operations in any way.

Alice decides that a depth of 3 will be sufficient for her work. She does not want to compromise her security, so she cannot reduce her $\rho'$ value, so she instead looks at augmenting $\eta$. Using $\rho' = 24$, $d = 3$ and $(\rho' + 3)(d + 1) < \eta - 3$ she obtains

$$\eta > (27)(4) + 3 = 111.$$

She chooses $\eta = 112$ and sets up her remaining parameters in the same way as Section 2.4.5:

$$\begin{aligned}\gamma &= \lambda\eta^2 = 125440\\ \tau &= \gamma + \lambda = 125450\\ \rho' &= \lceil \rho + \log(\tau + 1) \rceil = 27.\end{aligned}$$

Since Alice has changed her value of $\rho'$, she must reverify her earlier calculations to see if her new parameters allow for the correct depth. She sees that

$$\frac{\eta - 3}{\rho' + 3} - 1 = \frac{112 - 3}{27 + 3} - 1 \approx 2.63,$$

meaning that her scheme can still not process a circuit of depth 3.

Using $\rho' = 27$ to determine $\eta$ in turn pushes $\rho'$ up to 28. Therefore, she uses $\rho' = 28$ to obtain

$$\eta > (31)(4) + 3 = 127,$$

which in turn, using $\eta = 128$, gives her

$$\begin{aligned}\gamma &= 163840\\ \tau &= 163850\\ \rho' &= 28.\end{aligned}$$

Since her $\rho'$ hasn't changed this time, Alice knows she has chosen the correct set of parameters to allow for a depth-3 homomorphic implementation of the DGHV scheme.

The authors of the DGHV scheme detail in [31, Sec.6] how to modify the scheme once again in order to accommodate unlimited depth and transform the scheme into one that is fully homomorphic. We will interpret their results in the next chapter.

# Chapter 4

# A Fully Homomorphic DGHV Scheme

So far, we have seen that the DGHV scheme is somewhat homomorphic but that the depth of the circuit it can evaluate is limited by the parameter sizes of the scheme. In this chapter, we interpret van Dijk et al.'s methods in [31] to modify the scheme in order to make it fully homomorphic.

We explain in Section 4.1 the idea behind the transformation of the scheme from SHE to FHE, outlining the main bottleneck of the scheme as it is described presently in Section 4.2. We then describe van Dijk et al.'s proposed modifications to the scheme and how they work in Sections 4.3 and 4.4, analysing an alternate formulation of one key part of their modified scheme in Section 4.5. Finally, we compare our analysis and constraints throughout the thesis to the proposed parameters and methods of the DGHV authors in Section 4.6.

It is convenient to use the language of circuits throughout this chapter since modular addition and multiplication can be thought of as basic components of any circuit on binary inputs. We therefore call the decryption function from Section 2.1.2 a circuit for the entirety of this chapter.

While we do not offer an implementation of this modified scheme, we do provide an interpretation of how to express decryption as a circuit in order to encrypt it. Our contributions are to elaborate on decryption as a circuit in Section 4.4.1, including details on how encryption would work in Section 4.4.2, and our analysis of the ripple-carry adder as an alternate explanation of part of the circuit in Section 4.5.

## 4.1 From Somewhat Homomorphic to Fully Homomorphic

As we saw in Section 3.4, in order for the DGHV scheme to attain a set depth $d$, we must choose the parameters as a function of $d$, and their sizes will increase significantly with $d$. Therefore, to accommodate a circuit of greater depth would imply restarting the process with new, larger parameters. Since the main factor which led to increasing the size of the parameters in the DGHV scheme is the accumulation of noise on ciphertexts as they are evaluated, the most obvious way to modify the scheme to accommodate a larger depth is to reduce the noise in some manner while performing operations on ciphertexts.

One method that can be used to accomplish this is to decrypt ciphertexts and then re-encrypt the corresponding plaintexts, generating fresh ciphertexts with small noise terms on which we can continue to operate. This would involve a lot of extra communication between the owner of the private key and the system performing the calculations, so it would be very time-consuming. What's more, having the owner of the private key periodically decrypt intermediate steps of the calculations being performed and potentially guess those calculations is contradictory to the purpose of homomorphic encryption, as it contravenes Gentry's notion of circuit privacy [14, Def.2.1.6], which we mentioned in Section 3.3.3.

Recall that in depth-$d$ DGHV the ciphertext space is denoted $\mathcal{C}$ and the restricted ciphertext space, that which allows us to attain depth $d$, is, as per (3.4.1),

$$\mathcal{C}'_d = \left\{ \mathrm{c} \in \mathcal{C} \,\middle|\, (|N(\mathrm{c})| + 1)^{d+1} + 3d(\gamma + 1)2^{\rho} < \frac{p}{2} \right\}.$$

**Definition 4.1.1.** A *refresh function* in a fully homomorphic encryption scheme is a function

$$Ref : \mathcal{C} \longrightarrow \mathcal{C}'_{d'}$$

for some $d' \geq 1$ such that, for every $\mathrm{c} \in Enc(\mathcal{M})$,

$$Dec(Ref(\mathrm{c})) = Dec(\mathrm{c}).$$

We have seen that the noise $N(\mathrm{c})$ is distributed around the value $\mathtt{m} + kp$, which is known to only the private key owner. At this point, being able to reduce the noise seems to be synonymous with being able to decrypt ciphertexts.

The significant contribution of Gentry in his thesis [14], elaborated by him in [15] and by van Dijk et al. in [31], is to view the decryption function as a circuit on binary inputs which can therefore be encrypted and evaluated homomorphically. They still decrypt and re-encrypt data, stripping away the excess noise that has accumulated, but they do so *while the data is still encrypted*. Therefore, for the DGHV scheme, we realize *Ref* by effectively encrypting the decryption circuit. If we are able to express

our decryption function as a binary circuit, then it may be encrypted just as any other binary circuit in the scheme.

Gentry's method, which he calls "Recrypt", is defined as follows. Let DEC represent the encrypted decryption circuit and let C be a ciphertext that is the result of some number of operations; in other words, it is not fresh, as per our definition in Section 2.1.3. Let $\mathrm{C} = \mathtt{c}^{\gamma-1}\mathtt{c}^{\gamma-2}\ldots\mathtt{c}^{1}\mathtt{c}^{0}$ be its binary representation (this notation will be used in later sections as well). What we are looking to accomplish can be represented by the following diagram. In this context, $\mathrm{C}' = Ref(\mathrm{C})$.

$$\begin{array}{ccc} \mathrm{C} = \mathtt{c}^{\gamma-1}\mathtt{c}^{\gamma-2}\ldots\mathtt{c}^{1}\mathtt{c}^{0} & \xrightarrow{Dec} & \mathtt{m} \\ \downarrow{\scriptstyle (Enc,\ldots,Enc)} & \searrow^{Ref} & \uparrow{\scriptstyle Dec} \\ \mathrm{C}^{\gamma-1}\mathrm{C}^{\gamma-2}\ldots\mathrm{C}^{1}\mathrm{C}^{0} & \xrightarrow[\text{DEC}]{} & \mathrm{C}' \end{array} \tag{4.1.1}$$

In other words, the goal is that our scheme be such that one can "re-encrypt" ciphertexts and then apply the decryption circuit homomorphically, thus removing the noise associated with the first encryption and refreshing the ciphertext, allowing for more operations. The resulting refreshed ciphertext could not, however, be a true "fresh" ciphertext, in the sense of being an element of $Enc(\mathcal{M})$, since as we saw in Chapter 3, the process of evaluating a circuit itself introduces noise on the ciphertext.

Suppose the depth of the decryption circuit (that is, the multiplicative depth as described in Section 3.4) is $d_c$. Then we need $d_c < d$ in order for this scheme to yield a refresh function. Furthermore, we must have that $N(\mathrm{C}') < N(\mathrm{C})$. Indeed, for a ciphertext $\mathrm{C} = \mathtt{m} + kp + N(\mathrm{C})$, the decryption function removes all but the bit $\mathtt{m}$, but the encrypted decryption circuit will reintroduce some noise based on what operations it involves. It is therefore important that the depth of the decryption circuit be such that it removes more noise than it introduces.

The following definition is adapted from [14, Def.4.1.3]; it represents the minimum requirement.

**Definition 4.1.2.** A *bootstrappable encryption scheme* is a SHE scheme of depth great enough to be able to evaluate its own decryption circuit, plus one more operation.

As described by Gentry [14], [15], in order to achieve fully homomorphic encryption, one can start with a somewhat homomorphic encryption scheme and modify it to make it first bootstrappable, then fully homomorphic.

Let us describe how a bootstrappable cryptographic scheme could produce a refresh function. We begin with our plaintext $\mathtt{m}$ and a private key and public key pair, denoted $(sk, pk)$. The plaintext is encrypted under the public key as before, generating a ciphertext

$$\mathrm{C}_0 \in Enc_{pk}(\mathtt{m}).$$

We then perform any necessary operations (additions and multiplications) on this and other ciphertexts up to a certain depth $d' = d - d_c$ (in the minimum requirements above, $d' = 1$, because it takes the rest of the depth to evaluate the decryption circuit). Let $\mathrm{c}$ be the ciphertext resulting from these operations.

As the owner of the private key, we must encrypt it in order to be able to employ the bootstrappability property of the scheme, so we calculate

$$\mathrm{sk}' \in Enc_{pk}(sk).$$

Finally, the ciphertext $\mathrm{c}$ is encrypted a second time, bit by bit, giving $\mathrm{C} = Enc_{pk}(\mathrm{c})$, where C can be thought of as a vector of integers. Since we have encrypted the private key, we may now encrypt the decryption circuit and evaluate it homomorphically on the doubly encrypted ciphertext, using the bootstrappability property. The result is then

$$\mathrm{Dec}(\mathrm{C}, \mathrm{sk}') = \mathrm{c}',$$

a "refreshed" encryption of our plaintext $\mathtt{m}$ under $pk$, that is, it satisfies $Dec(\mathrm{c}') = Dec(\mathrm{c})$.

As previously mentioned, by evaluating the decryption circuit, we are removing the noise associated with encrypting under $pk$ the first time together with the noise accumulated through $\mathfrak{M}$ and $\mathfrak{A}$ operations. If the noise removed is larger than the noise introduced while evaluating the circuit, then the resulting ciphertext has been refreshed and contains less total noise, thus enabling more operations to be performed on it before having to refresh again. Running this bootstrappable scheme recursively gives us a *fully homomorphic encryption* (FHE) *scheme*. This method of modifying a somewhat homomorphic scheme to make it bootstrappable, its first successful implementation, and the proof that, recursively, this brings about fully homomorphic encryption was first proposed by Gentry in his 2009 PhD thesis [14].

## 4.2 Bootstrappability Implementation Problems

To make the DGHV scheme bootstrappable, we must first be able to express the decryption function as a binary circuit.

We know from Section 2.1.2 that decryption as a function corresponds to $\mathrm{c} \bmod p \bmod 2$. Let us first simplify the function slightly. Let $b = \mathrm{c} \bmod p$. Then $b = \mathrm{c} - p\lceil \mathrm{c}/p \rfloor$. Now, since $p$ is odd, we find that $b \equiv \mathrm{c} - \lceil \mathrm{c}/p \rfloor \bmod 2$, that is they have the same parity. Therefore, our decryption function could be implemented as

$$Dec(\mathrm{c}) = \mathrm{c} - \left\lceil \frac{\mathrm{C}}{p} \right\rfloor \mod 2. \tag{4.2.1}$$

In order to express this as a binary circuit, we must understand what it means to divide one binary number by another, and how complex that operation is.

| Parameter | Description |
|---|---|
| $\Theta$ | Number of new public key elements |
| $\kappa$ | Bits of precision of new public key elements |
| $\theta$ | Size of sparse subset |
| $n$ | Number of bits of precision kept in the $z_i$ |

Table 4.1: A list of new parameters for the modified, squashed DGHV scheme

Division of binary numbers is known to be a very complex circuit; on many processors, parts of the calculations are relegated to lookup tables and software implementation rather than encoding the algorithm as a sequence of gates. One classic algorithm, proposed by Knuth [23, Sec.4.6.1, Alg.D], has complexity $O(\eta(\gamma - \eta))$. The corresponding circuit's depth (as measured by the number of AND gates) is thus certainly larger than $\eta(\gamma - \eta)$, so it is impossible to satisfy (3.4.3).

We must therefore find a way to modify the scheme once again in order to make it bootstrappable. Gentry's novel method involves "squashing the decryption circuit" [14],[15],[31]. He modifies the private key in such a way that some of the most complex steps of the decryption function can be pre-computed. This is done by modifying the SHE scheme so that it contains a "hint" to $p$; namely, a set that contains a hidden sparse subset that sums to $1/p$.

## 4.3 Squashing the Private Key

As we search for ways to simplify the decryption circuit, we see that we could replace the $c/p$ operation with a much simpler subset-sum problem, thus using addition rather than division. Just as in Section 3.3.1, we are once again modifying the existing scheme in order to reduce the complexity (and depth) of the decryption circuit sufficiently to make the scheme bootstrappable. In Gentry's thesis [14] (and in subsequent work by van Dijk et al. [31]), he calls this "squashing the decryption circuit." The first step is to modify the private key to allow for a simpler decryption function. We call this step "squashing the private key."

We first define four new variables: $\Theta, \kappa, \theta$, and $n$. Their uses are described in Table 4.1. We will be modifying the current version of the DGHV by adding more elements to the public key and by using a modification of the private key for decryption.

Let $\Theta$ be a large positive integer and $\theta$ a significantly smaller positive integer (to be determined in Sections 4.3.1 and 4.3.2). We begin by generating a sparse subset $S \subset \{1, \ldots, \Theta\}$ that is of size $\theta$. We then identify the elements of the subset with an

indicator vector: let $s = [\mathtt{s}_1, \mathtt{s}_2, \ldots, \mathtt{s}_\Theta]$ be such that

$$\mathtt{s}_i = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise.} \end{cases}$$

This vector $s$ is the new private key, replacing $p$. We will also add $\Theta$ new elements to the public key, as follows.

We create a vector $y = [y_1, \ldots, y_\Theta]$ with the following properties: first, for each $i = 1, \ldots, \Theta$, the corresponding $y_i \in \mathbb{Q} \cap [0, 2)$ has the form

$$y_i = \sum_{j=-\kappa}^{0} \mathtt{y}_i^j \cdot 2^j = \mathtt{y}_i^0 . \mathtt{y}_i^{-1} \mathtt{y}_i^{-2} \ldots \mathtt{y}_i^{-\kappa},$$

where $\mathtt{y}_i^j \in \{0, 1\}$, such that we express the binary representation of this rational number. Next, the set satisfies that $\sum_{i \in S} y_i - \frac{1}{p}$ is a even integer, which we write compactly as

$$\left[ \sum_{i \in S} y_i \right]_2 = \left[ \frac{1}{p} \right]_2, \tag{4.3.1}$$

reading this as "mod 2." Thus, the private key $s$ indicates which $\theta$ elements amongst the $\Theta$ $y_i$'s are needed to decrypt. We see that the owner of $s$ can easily recover an estimate of $1/p$, rounded to $\kappa + 1$ binary digits, from $y$.

In this modified scheme, encryption of plaintexts as well as addition and multiplication of ciphertexts function the same way as before, because we have made no modifications to the original public key. However, the decryption of ciphertexts changes.

Let $\mathrm{c}$ be the ciphertext to be decrypted. For $i = 1, \ldots, \Theta$, evaluate $z_i = [\mathrm{c} \cdot y_i]_2 \in \mathbb{Q}$, which we will denote as the vector $z = [z_1, z_2, \ldots, z_\Theta]$. We note that each $z_i$ is not a bit, but rather the binary representation of a rational number, and we will truncate it to keep $n$ bits of precision after the binary point. We will choose $n$ in Section 4.3.1. This part of the operation uses only public information, so may be pre-computed by anyone.

We now observe that, when rounded to the nearest integer, we have

$$\left\lceil \sum_{i=1}^{\Theta} \mathtt{s}_i z_i \right\rfloor = \left\lceil \sum_{i=1}^{\Theta} \mathtt{s}_i \cdot \mathrm{c} \cdot y_i \right\rfloor = \left\lceil \mathrm{c} \sum_{i \in S} y_i \right\rfloor \equiv \left\lceil \frac{\mathrm{c}}{p} \right\rfloor \mod 2.$$

We have now managed to express the division of $\mathrm{c}$ by $p$ as a sum of certain other elements, replacing the complex division circuit with one involving rounding a sum of rational numbers to the nearest integer and taking its parity.

Therefore, we see that (4.2.1) becomes

$$Dec(\mathrm{c}) = \mathrm{c} - \left\lceil \sum_{i=1}^{\Theta} \mathtt{s}_i z_i \right\rfloor \mod 2.$$

We know that, since our final operation is a reduction modulo 2, we need only compare the parities of the two integers involved. Given an $n$-bit integer $a = \mathtt{a}^{n-1} \dots \mathtt{a}^1 \mathtt{a}^0$, we have that $\mathtt{a}^0 = a \mod 2$, which is to say the parity of an integer is equal to its least significant bit. Therefore, for any two binary numbers $a$ and $\bar{a}$, $a - \bar{a} \mod 2 = \mathtt{a}^0 \oplus \bar{\mathtt{a}}^0$.

Next, given a rational number $b$ in binary notation with $m$ binary digits, that is, $b = \mathtt{b}^0.\,\mathtt{b}^{-1}\mathtt{b}^{-2} \dots \mathtt{b}^{-m+1}$, we round $b$ to the nearest integer by looking at $\mathtt{b}^0$ and $\mathtt{b}^{-1}$ as follows.

| $\mathtt{b}^0.\,\mathtt{b}^{-1}$ | $\lceil b \rfloor$ |
|---|---|
| 0.0 | 0 |
| 0.1 | 1 |
| 1.0 | 1 |
| 1.1 | 0 |

This can be represented as $\mathtt{b}^0 \oplus \mathtt{b}^{-1}$.

Therefore, denoting

$$\mathrm{c} = \mathtt{c}^{\gamma-1}\mathtt{c}^{\gamma-2} \dots \mathtt{c}^1\mathtt{c}^0,$$

$$\left[ \sum_{i=1}^{\Theta} \mathtt{s}_i z_i \right]_2 = z_{add} = \mathtt{z}^0_{add}.\,\mathtt{z}^{-1}_{add}\mathtt{z}^{-2}_{add} \dots \mathtt{z}^{-n}_{add}, \tag{4.3.2}$$

decryption is expressed as the following circuit:

$$Dec(\mathrm{c}, z_1, \dots, z_\Theta) = \mathrm{c}^0 \oplus \mathtt{z}^0_{add} \oplus \mathtt{z}^{-1}_{add}. \tag{4.3.3}$$

Note that the computation of $z_{add}$, which is the complex part of this circuit, will be explained in Section 4.4.

Since we have once again added more information to our public key, we must therefore examine the possible impact and set our new parameters $\theta, n, \kappa$, and $\Theta$ accordingly; namely, we must ensure that the modified encryption scheme is still functional and that there has been no compromise to its security.

### 4.3.1 Parameter constraints of the squashed key: functionality

As we have done in previous chapters, we first examine the functionality constraints on our new parameters.

Recall (4.3.1); since the $y_i$'s are rational numbers with $\kappa + 1$ binary digits, we may rewrite the sum as

$$\sum_{i \in S} y_i = \frac{1}{p} + \Delta p,$$

where $\Delta p$ is the rounding error. Next, as in Section 4.3, we evaluate $z_i = [\mathrm{C} \cdot y_i]_2$ and truncate the corresponding binary representations to $n$ bits. Setting $\Delta z_i = z_i - \mathrm{C}y_i$, representing the error from the truncation of binary digits, we obtain

$$\begin{aligned}
\sum_{i \in S} z_i &= \sum_{i \in S} (\mathrm{C}y_i + \Delta z_i) \\
&= \mathrm{C} \sum_{i \in S} y_i + \sum_{i \in S} \Delta z_i \\
&= \mathrm{C} \left( \frac{1}{p} + \Delta p \right) + \sum_{i \in S} \Delta z_i \\
&= \frac{\mathrm{C}}{p} + \mathrm{C}\Delta p + \sum_{i \in S} \Delta z_i.
\end{aligned}$$

Recall that for the scheme to work, we require that

$$\left\lceil \sum_{i \in S} z_i \right\rfloor \equiv \left\lceil \frac{\mathrm{C}}{p} \right\rfloor \mod 2.$$

We therefore want

$$\left\lceil \frac{\mathrm{C}}{p} \right\rfloor = \left\lceil \frac{\mathrm{C}}{p} + \mathrm{C}\Delta p + \sum_{i \in S} \Delta z_i \right\rfloor \tag{4.3.4}$$

in order for the modified scheme to return a valid ciphertext.

Recall the form of a ciphertext from (2.1.5); we see that

$$\frac{\mathrm{C}}{p} = k + \frac{\mathtt{m} + N(\mathrm{C})}{p}.$$

Since we have chosen our parameters to be at least depth-1 homomorphic, that is, $Enc(\mathcal{M}) \in \mathcal{C}_1$ using (3.4.1) with $d = 1$, we know that the noise on a fresh ciphertext is bounded above by $1/4$. Therefore, the latter part of the above is less than $1/4$. For (4.3.4) to hold, we must therefore have

$$\mathrm{C}\Delta p + \sum_{i \in S} \Delta z_i < \frac{1}{4}.$$

We will therefore look to bound each term above by $1/8$.

Starting with the first term, since the sum of the $y_i$'s gives an estimate for $1/p$ with $\kappa$ bits of precision after the binary point, we can choose the $y_i$'s so that $\Delta p \leq 2^{-\kappa-1}$. What's more, we saw in Sections 2.4.3 and 3.3.1 that a ciphertext, when optimized, satisfies $c < 2^{\gamma}$. Therefore, to ensure that $c\Delta p \leq 2^{-3}$, we require $2^{\gamma} \cdot 2^{-\kappa-1} \leq 2^{-3}$, or

$$\gamma + 2 \leq \kappa. \tag{4.3.5}$$

Looking now at the second term, we know that, since the $z_i$'s were truncated to $n$ bits after the binary point, each $\Delta z_i$ is bounded above by $2^{-n}$. Therefore, we have

$$\left| \sum_{i \in S} \Delta z_i \right| < \theta 2^{-n}.$$

To guarantee that this is less than $1/8$, we need $\theta 2^{-n} < 2^{-3}$, or

$$\log \theta + 3 < n. \tag{4.3.6}$$

Finally, let us address how to create, as in the previous section, a set such that there exists a sparse subset whose sum is congruent to $1/p$ modulo 2. These $y_i$'s may be created by first creating the indicator vector $s$ of the elements of the subset $S$, then generating $\Theta$ uniformly random integers $u_i \in \mathbb{Z} \cap [0, 2^{\kappa+1})$, then re-setting the element $u_\theta$ to be such that

$$\sum_{i \in S} u_i \equiv \left\lceil \frac{2^{\kappa}}{p} \right\rfloor \mod 2^{\kappa+1}.$$

We then obtain the desired rational numbers by setting $y_i = u_i / 2^{\kappa}$.

### 4.3.2 Parameter constraints of the squashed key: security

By including a "hint" to $1/p$ in the new elements added to the public key, we are potentially making our scheme easier to attack. We now examine the constraints that ensure this is not the case.

Unlike in Section 2.4.2, where we determined that the security of the scheme did not rely on hiding the subset used in the subset-sum, in this case it is vital that an attacker be unable to discover the proper subset. The security assumption on which we therefore rely for protecting the squashed private key is related to that of the *sparse subset-sum problem* (SSSP) or *low-weight knapsack problem*. Recall the definition of the subset-sum problem (SSP) from Section 2.4.2. Nguyen and Stern [27] explain that the difference between the SSP and the SSSP is that, for the SSSP, the number of elements that make up the subset – in this case, $\theta$ – is much smaller than half the number of elements there are in total, that is, $\theta < \Theta/2$.

Recall Micciancio and Goldwasser's definition of density as the ratio between the number of elements in a set and the size of its largest element [25, Ch.3, Sec.2]. Here, we have $\Theta$ elements, each of which has $\kappa+1$ binary digits (but only $\kappa$ after the binary point), so the density is

$$\partial = \frac{\Theta}{\kappa}.$$

As stated in Section 2.4.2, Coster et al. showed in [10] that if the density is less than 0.9408, then the corresponding subset-sum problem is easy to solve.

Actually, in this setting, we would like there to be multiple subset sums congruent to $1/p$, but we would also like there to be multiple subset sums congruent to $1/q$ for odd integers $q$ in the same range as $p$, that is, $q \in (2^{\eta-1}, 2^{\eta})$. As per [25, Ch.3, Sec.2], for density $\partial$ one expects on average $2^{\partial}$ preimages for any given sum. Gentry states in [14, Ch.11] that for security, we want there to be exponentially many subsets whose sum is congruent to $1/p$ or to any other $1/q$ in the same range. We therefore want $2^{\partial}$ to correspond to at least $2^{\lambda}$. This brings forth

$$\Theta \geq \kappa\lambda. \tag{4.3.7}$$

Finally, we examine the size requirement for $\theta$. A fairly straightforward brute-force attack against the squashed key would be to establish all possible subsets of the $y_i$'s of size $\theta$, compute all of their sums, and keep the sums that are in the range $[2^{-\eta}, 2^{-\eta-1}]$, as they are potential candidates for $1/p$. To make this brute-force attack infeasible, we need to have the number of subsets be at least exponential in the security parameter; that is, $2^{\theta} \geq 2^{\lambda}$ or

$$\theta \geq \lambda. \tag{4.3.8}$$

Following Coster et al.'s results, Nguyen and Stern [27] adapted the lattice attacks on low-density subset-sum problems to be able to solve instances of the SSSP. They too came to the same conclusion, that (4.3.7) and (4.3.8) thwart their lattice-based attack.

## 4.4 The Squashed Decryption Circuit

Now that we have modified our decryption function, reducing its depth by changing division into addition, and we have expressed part of the decryption as a binary circuit in (4.3.3), we will examine the means by which we make (4.3.2) into a binary circuit to encrypt it and evaluate it homomorphically.

We start, as before, with a plaintext message $\mathtt{m} \in \{0, 1\}$ and encrypt it to obtain $\mathrm{c} \in \mathbb{Z}$. Using the vector $y = [y_1, \ldots, y_\Theta]$ from the above modified encryption scheme, we evaluate, for each $i$, $z_i = [\mathrm{c} \cdot y_i]_2$, using the binary representation of $z_i$ and keeping

only $n$ bits of precision after the binary point as well as only the least significant bit before the binary point. As such, each $z_i$ will have the form

$$z_i = \sum_{j=-n}^{0} \mathtt{z}_i^j \cdot 2^j = \mathtt{z}_i^0 . \mathtt{z}_i^{-1} \mathtt{z}_i^{-2} \ldots \mathtt{z}_i^{-n}.$$

We note here that the above computations are all done using public information; therefore, anyone may perform these computations and, though they are not part of the decryption function, they are accomplishing some of the work required to decrypt.

These $z_i$'s, along with the squashed private key $s$, are the new inputs to the decryption function; in order to better understand what happens when the circuit and data are encrypted, let us first evaluate the decryption circuit explicitly using the squashed key. There will be one step of the process that we will leave as a black box for this section; we will come back to it in Section 4.5.

### 4.4.1 Explicit computation of the decryption function

We have the $z_i$'s, each an $(n+1)$-bit number, as well as the private key $s = [\mathtt{s}_1, \ldots, \mathtt{s}_\Theta]$ from the modified key. Evaluate

$$\mathtt{s}_i z_i \quad \text{for all } i = 1, \ldots, \Theta. \tag{4.4.1}$$

Next, let $F$ denote the function that performs binary addition modulo 2 on $\Theta$ binary rational numbers of $(n+1)$ bits each. That is, the function adds binary numbers of length $(n+1)$ and keeps all the bits of precision after the binary point, the least significant bit of the integer part, and nothing more (in other words, it discards any more significant bits of the integer part). Denote the output of this function

$$w_{add} = F(w_1, \ldots, w_\Theta), \tag{4.4.2}$$

where each $w_i$ is $(n+1)$ bits long, as is the output $w_{add}$. A key point is to choose an efficient implementation for the function $F$ as a binary circuit. An example of a circuit implementing $F$ is the ripple-carry adder, which we will discuss in Section 4.5, but there are many different algorithms that can do this job.

We apply this function to our set $\{\mathtt{s}_i z_i \mid i = 1, \ldots, \Theta\}$ to obtain

$$F(\mathtt{s}_1 z_1, \ldots, \mathtt{s}_\Theta z_\Theta) = z_{add} = \mathtt{z}_{add}^0 . \mathtt{z}_{add}^{-1} \mathtt{z}_{add}^{-2} \ldots \mathtt{z}_{add}^{-n}.$$

Then we have

$$z_{add} = \left[ \sum_{i=1}^{\Theta} \mathtt{s}_i z_i \right]_2 = \left[ \sum_{i \in S} z_i \right]_2$$

since $\mathtt{s}_i = 1$ if and only if $i \in S$.

We now see that $z_{add}$ is the same as (4.3.2); therefore, we obtain (4.3.3):

$$Dec(\mathrm{C}, z_1, \ldots, z_\Theta) = \mathsf{z}^0_{add} \oplus \mathsf{z}^{-1}_{add} \oplus \mathsf{c}^0. \tag{4.4.3}$$

In summary, the decryption function can be described as a three-part process: (4.4.1), which consists of $\Theta$ multiplications, each a depth-one circuit; (4.4.2), which we will analyze in Section 4.5, and (4.4.3), which consists of two bitwise additions. We next examine what it means to encrypt the decryption circuit.

### 4.4.2 Encryption of the decryption circuit

Now that we have an explicit circuit with which to decrypt data, we will examine what is involved when encrypting the circuit. We note that throughout this section, we maintain the convention that an encryption of the bit $\mathsf{a}$ is the integer A.

Rather than thinking about the $z_i$'s as rational numbers in binary form, they will be stored bit by bit as row vectors, so we define

$$\vec{z_i} = [\mathsf{z}^0_i, \mathsf{z}^{-1}_i, \ldots, \mathsf{z}^{-n}_i].$$

The first step is to apply, for $i = 1, \ldots, \Theta$, the encryption function to each bit of each $\vec{z_i}$ as well as each element of the private key $s$, to obtain

$$Enc : \vec{z_i} = [\mathsf{z}^0_i, \mathsf{z}^{-1}_i, \ldots, \mathsf{z}^{-n}_i] \to \vec{\mathrm{Z}_i} = [\mathrm{Z}^0_i, \mathrm{Z}^{-1}_i, \ldots, \mathrm{Z}^{-n}_i],$$

where now each $\vec{\mathrm{Z}_i}$ is a vector of integers for $i = 1, \ldots, \Theta$, and

$$Enc : s = [\mathsf{s}_1, \ldots, \mathsf{s}_\Theta] \to \mathrm{S} = [\mathrm{S}_1, \ldots, \mathrm{S}_\Theta].$$

Next, evaluate

$$\overrightarrow{\mathrm{S}_i\mathrm{Z}_i} = [\mathrm{S}_i\mathrm{Z}^0_i, \mathrm{S}_i\mathrm{Z}^{-1}_i, \ldots, \mathrm{S}_i\mathrm{Z}^{-n}_i], \quad i = 1, \ldots, \Theta,$$

where we use the shorthand $\mathrm{S}_i\mathrm{Z}^j_i = \mathfrak{M}(\mathrm{S}_i, \mathrm{Z}^j_i)$ for simplicity, recalling that we optimize.

Now suppose that F is the encrypted version of the function $F$ defined in (4.4.2). We apply F on these vectors of integers. We elaborate on this function in Section 4.5. It is important to note that we are now "adding" **vectors of integers** as if they were **bit strings**.

$$\mathrm{F}(\overrightarrow{\mathrm{S}_1\mathrm{Z}_1}, \ldots, \overrightarrow{\mathrm{S}_\Theta\mathrm{Z}_\Theta}) = \overrightarrow{\mathrm{Z}_{add}} = [\mathrm{Z}^0_{add}, \mathrm{Z}^{-1}_{add}, \ldots, \mathrm{Z}^{-n}_{add}].$$

What remains is to encrypt the least significant bit (or parity) of the ciphertext $\mathrm{C} = \mathsf{c}^{\gamma-1}\mathsf{c}^{\gamma-2}\ldots\mathsf{c}^1\mathsf{c}^0$:

$$Enc : \mathsf{c}^0 \to \mathrm{C}^0.$$

Finally, evaluate the decryption circuit on the encrypted data,

$$\mathrm{c}' := \mathrm{Z}^0_{add} + \mathrm{Z}^{-1}_{add} + \mathrm{C}^0,$$

where $\mathrm{Z}^0_{add} + \mathrm{Z}^{-1}_{add} + \mathrm{C}^0$ is shorthand for $\mathfrak{A}(\mathfrak{A}(\mathrm{Z}^0_{add}, \mathrm{Z}^{-1}_{add}), \mathrm{C}^0)$ (we will use this shortened notation for the remainder of the thesis). Finally, $\mathrm{c}' = \mathit{Ref}(\mathrm{c})$ is the resulting "refreshed" version of the ciphertext c. By the construction (4.1.1), $\mathit{Dec}(\mathrm{c}') = \mathtt{m}$.

## 4.5 Example Adder: the Ripple-Carry Adder

The main computation in the squashed decryption circuit above is $F$, the addition of numbers in binary representation. Many different implementations of $F$ exist. In this section, we analyze the classic implementation of $F$, called the *ripple-carry adder*. Our goal is to estimate the *total noise on the circuit*, that is, the noise introduced as a consequence of encrypting this circuit in the DGHV scheme. This will show why it is inefficient to simply think about straightforward algorithms for binary addition. In fact, van Dijk et al. use in [31] a far more optimized circuit for $F$, the need for which we will motivate in Section 4.5.2 but which we will not describe in detail nor analyze.

Let us first define the classic circuit for the addition of two bits, called a *full adder* [24, Sec.4.3]. The full adder takes three bits as inputs: the two bits $\mathtt{x}$ and $\mathtt{y}$ to be added as well as a "carry in" bit, denoted $\mathtt{C}_{in}$, from any previous computations ($\mathtt{C}_{in}$ is initialized to 0 when adding bits for the first time). The adder outputs two bits: $\mathtt{S}$ the sum modulo 2 of $\mathtt{x}$ and $\mathtt{y}$ and $\mathtt{C}_{out}$ the "carry out" bit, which computes the carry generated by adding $\mathtt{x}$ and $\mathtt{y}$.

Naturally, we can write $\mathtt{S}$ algebraically as

$$\mathtt{S} = \mathtt{x} + \mathtt{y} + \mathtt{C}_{in} \mod 2. \tag{4.5.1}$$

To construct an algebraic formula for $\mathtt{C}_{out}$, let's look at the two possibilities for $\mathtt{C}_{in}$:

**Case 4.1.** $\mathtt{C}_{in} = 0$.

In this case, the carry out bit is equal to 1 if and only if both $\mathtt{x}$ and $\mathtt{y}$ are 1. This then can be written as

$$\mathtt{C}_{out} = \mathtt{xy} \mod 2.$$

**Case 4.2.** $\mathtt{C}_{in} = 1$.

Here, the carry out bit is 1 if one or both of $\mathtt{x}$ and $\mathtt{y}$ is 1. We can write this algebraically as

$$\mathtt{C}_{out} = \mathtt{xy} + \mathtt{x} + \mathtt{y} \mod 2.$$

We can combine these two cases by noting that the terms $\mathtt{C}_{in}\mathtt{x}$ and $\mathtt{C}_{in}\mathtt{y}$ give $\mathtt{x}$ and $\mathtt{y}$, respectively, if $\mathtt{C}_{in} = 1$ and give 0 otherwise. This means we can write $\mathtt{C}_{out}$ algebraically as

$$\mathtt{C}_{out} = \mathtt{xy} + \mathtt{C}_{in}\mathtt{x} + \mathtt{C}_{in}\mathtt{y} = \mathtt{xy} + \mathtt{C}_{in}(\mathtt{x} + \mathtt{y}) \mod 2. \tag{4.5.2}$$

Note that this is simply the symmetric polynomial of degree two in three variables; it is 1 if at least two of the three variables are 1.

From the algebraic formulas we construct a circuit for the full adder in Figure 4.1. Each multiplication corresponds to an AND gate and each bitwise sum is an XOR gate. Note that this is not the same circuit that is presented in [24, Fig.4.5], as they use an OR gate in addition to AND and XOR gates.

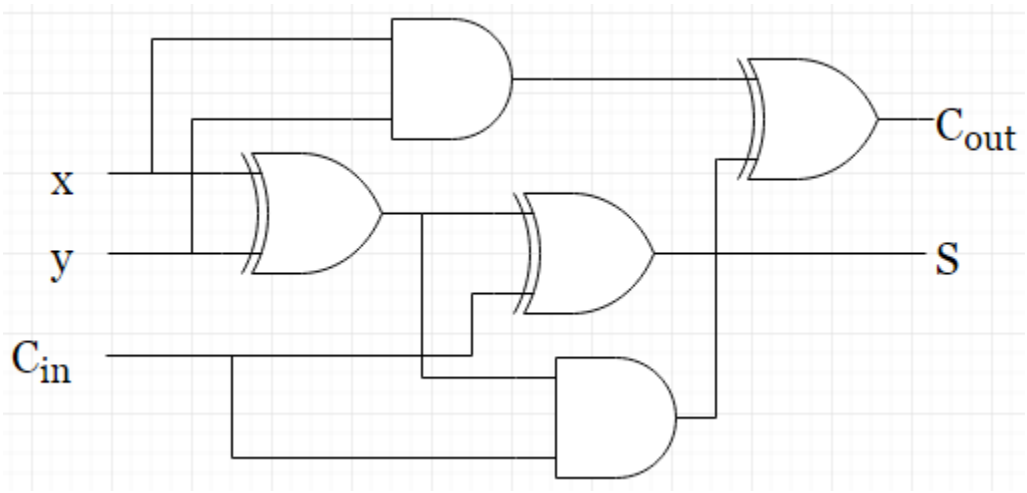


Figure 4.1: Circuit representation of a full adder, where the symbol  represents an AND gate, or binary multiplication, and the other symbol represents an XOR gate, or binary addition.

To make the next part easier to illustrate, we will condense the full adder circuit into that of Figure 4.2.

In order to add two numbers $x$ and $y$ of $n$ bits each, with binary representations $x = \mathtt{x}^n\mathtt{x}^{n-1}\ldots\mathtt{x}^2\mathtt{x}^1$ and $y = \mathtt{y}^n\mathtt{y}^{n-1}\ldots\mathtt{y}^2\mathtt{y}^1$ respectively, we can chain full adders together to obtain a ripple-carry adder. Moving right to left, the least significant bits of $x$ and $y$ are added with a full adder, and the carry out of the first adder, $\mathtt{C}^1_{out}$, becomes the carry in of the second adder. We denote $\mathtt{C}^0_{out} = \mathtt{C}_{in}$. Algebraically, we obtain, by recursion on (4.5.2) and (4.5.1),

$$\mathtt{C}^i_{out} = \mathtt{x}^i\mathtt{y}^i + \mathtt{C}^{i-1}_{out}(\mathtt{x}^i + \mathtt{y}^i) \tag{4.5.3}$$

$$\mathtt{S}^i = \mathtt{x}^i + \mathtt{y}^i + \mathtt{C}^{i-1}_{out}. \tag{4.5.4}$$

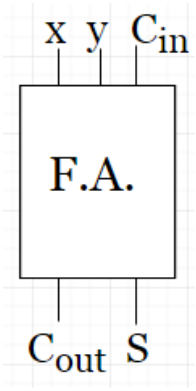


Figure 4.2: Simplified full adder circuit representation, with inputs above and outputs below

Using the condensed full adder circuit, we represent the ripple-carry adder circuit in Figure 4.3. The sum of $x$ and $y$, written bit-wise, is then $\mathtt{C}^n_{out}\mathtt{S}^n\mathtt{S}^{n-1}\ldots\mathtt{S}^2\mathtt{S}^1$. We note

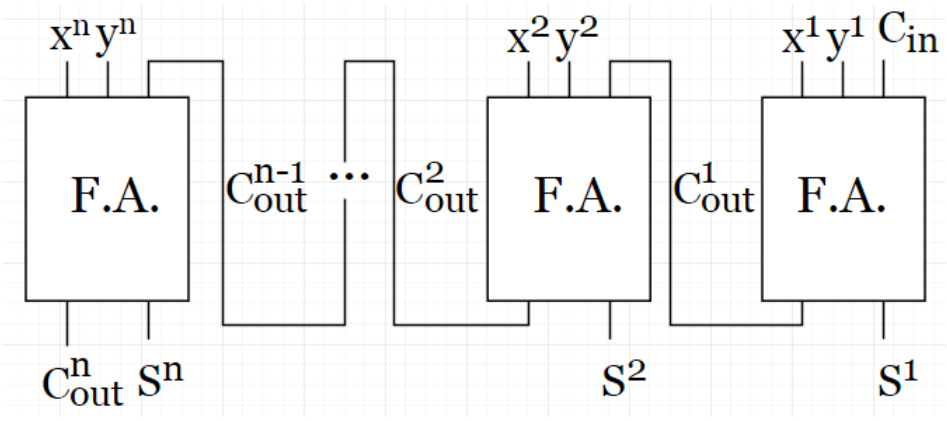


Figure 4.3: Circuit representation of the ripple-carry adder

that, for the DGHV scheme, the function $F$ adds two $(n+1)$-bit numbers and produces another $(n+1)$-bit number; thus, with this notation, we will not be evaluating the last carry-out bit.

Before we move on to bounding the noise accumulation of the ripple-carry adder, there is a discussion about depth to be had. Recall Definition 3.1.2; we have so far talked about the depth of a circuit in terms of composition of multiplications (with $2^{\rho'+2}$ additions being equivalent to one multiplication). However, we can also think of the depth of a circuit as the maximal number of AND gates that a given input is sent through while evaluating the circuit. Using algebraic notation to denote a circuit, as we have done above, we define a circuit as a polynomial on the input bits, and thus

define the depth as one less than the degree of the polynomial. With this in mind, we examine the depth of the ripple-carry adder.

**Lemma 4.5.1.** Let $x$ and $y$ be two $n$-bit numbers that are added using the ripple-carry adder in the context of the DGHV scheme. The depth of the ripple-carry adder circuit is then $n-1$.

**Proof:** Since (4.5.3) and (4.5.4) are the polynomials representing each output bit of the ripple-carry adder, it suffices to find the maximal degree of any polynomial and subtract one to obtain the depth of the circuit.

We see that

$$deg(\mathtt{C}^1_{out}) = deg(\mathtt{x}^1\mathtt{y}^1) = 2 \quad \text{and} \quad deg(\mathtt{S}^1) = deg(\mathtt{x}^1 + \mathtt{y}^1) = 1.$$

Suppose that, for $1 \leq i \leq n-1$,

$$deg(\mathtt{C}^{i-1}_{out}) = i \quad \text{and} \quad deg(\mathtt{S}^{i-1}) = i-1;$$

this is our induction hypothesis. Then, since by (4.5.3)

$$\mathtt{C}^i_{out} = \mathtt{x}^i\mathtt{y}^i + \mathtt{C}^{i-1}_{out}\mathtt{x}^i + \mathtt{C}^{i-1}_{out}\mathtt{y}^i,$$

we therefore obtain

$$deg(\mathtt{C}^i_{out}) = deg(\mathtt{C}^{i-1}_{out}) + 1 = i + 1.$$

Likewise, by (4.5.4) we have $deg(\mathtt{S}^i) = deg(\mathtt{C}^{i-1}_{out}) = i$.

Since we are evaluating the adder in the context of the DGHV scheme, we do not need to evaluate $\mathtt{C}^n_{out}$. The degree of the entire polynomial is then $\max\{deg(\mathtt{C}^{n-1}_{out}), deg(\mathtt{S}^n)\} = n$. As described above, the depth of the circuit is one less than the degree of the polynomial, so the depth of the ripple-carry adder circuit is $n-1$. ∎

The above Lemma only accounts for the multiplicative depth of the circuit; in fact, there are many additions in our formulas, so the actual depth of the ripple-carry adder will be greater than $n-1$. What's more, this depth does not accurately represent the number of gates that all the bits go through, only the very first bits, as the ripple-carry adder takes the output $\mathtt{C}^{i-1}_{out}$ as an input for the $i^{th}$ iteration. For those reasons, the depth of this circuit is actually an $n$-tuple of numbers, one for each output bit. In the next section, we will use a lower bound on these depths to estimate the minimal noise accumulation on the entire circuit.

### 4.5.1 Total noise on the circuit: adding two numbers

Let us now encrypt the ripple-carry adder in order to analyze the total noise on the circuit. Let $x, y \in \mathbb{Z}$ be written in binary representation as in the previous section. Then, let's encrypt each bit. Now, as we evaluate the ripple-carry adder circuit, the noise on the encryptions of $x$ and $y$ accumulates, as described in Chapter 3; let $g_i$ denote the noise on the encryption of $\mathtt{C}^i_{out}$ and $u_i$ denote the noise on the encryption of $\mathtt{S}^i$. Then, by observing the circuits from Section 4.4, we obtain a formula for the total accumulation of noise on one instance of the ripple-carry adder, shown in the following theorem.

**Theorem 4.5.2.** Let $x$ and $y$ be $n$-bit integers encrypted bitwise. Suppose the noise term of each encrypted bit is bounded below by $\mathbf{e}$. Then, the noise on a given bit $\mathtt{S}^i$ satisfies

$$u_i \geq \begin{cases} 2\mathbf{e}, & \text{for } \ i = 1 \\ \sum_{j=0}^{i-1} 2^j \mathbf{e}^{j+2} + (2\mathbf{e})^i + 2\mathbf{e}, & \text{for } \ 2 \leq i \leq n. \end{cases}$$

**Proof:** Since the noise on a multiplication/addition is at least equal to the product/sum of the noises (as per Lemma 3.2.2), we can establish recursive formulas for $u_i$ and $g_i$ using (4.5.3) and (4.5.4). Observing the circuits, we obtain, for $i > 1$,

$$\begin{aligned} g_1 &\geq \mathbf{e}^2 + 2\mathbf{e}, & u_1 &\geq 2\mathbf{e}, \\ g_i &\geq \mathbf{e}^2 + 2\mathbf{e}g_{i-1}, & u_i &\geq 2\mathbf{e} + g_{i-1}. \end{aligned}$$

We solve these recurrences as follows. For $i = 2, 3, 4$, we observe

$$\begin{aligned} g_2 &\geq \mathbf{e}^2 + 2\mathbf{e}\left[\mathbf{e}^2 + 2\mathbf{e}\right] = \mathbf{e}^2 + 2\mathbf{e}^3 + (2\mathbf{e})^2, \\ g_3 &\geq \mathbf{e}^2 + 2\mathbf{e}\left[\mathbf{e}^2 + 2\mathbf{e}^3 + (2\mathbf{e})^2\right] = \mathbf{e}^2 + 2\mathbf{e}^3 + 2^2\mathbf{e}^4 + (2\mathbf{e})^3, \\ g_4 &\geq \mathbf{e}^2 + 2\mathbf{e}\left[\mathbf{e}^2 + 2\mathbf{e}^3 + 2^2\mathbf{e}^4 + (2\mathbf{e})^3\right] = \mathbf{e}^2 + 2\mathbf{e}^3 + 2^2\mathbf{e}^4 + 2^3\mathbf{e}^5 + (2\mathbf{e})^4. \end{aligned}$$

Let us now prove by induction that the general formula for the lower bound on $g_i$ for $1 \leq i \leq n$ is

$$g_i \geq \sum_{j=0}^{i-1} 2^j \mathbf{e}^{j+2} + (2\mathbf{e})^i. \tag{4.5.5}$$

This hold for $i = 1$, so suppose the formula is valid for some $1 \leq i \leq n$. Then

$$\begin{aligned} g_{i+1} &\geq \mathbf{e}^2 + 2\mathbf{e}g_i \\ &\geq \mathbf{e}^2 + 2\mathbf{e}\left(\sum_{j=0}^{i-1} 2^j \mathbf{e}^{j+2} + (2\mathbf{e})^i\right) \quad \text{by the inductive hypothesis} \end{aligned}$$

$$
\begin{aligned}
&= \mathbf{e}^2 + \sum_{j=0}^{i-1} 2^{j+1}\mathbf{e}^{j+3} + (2\mathbf{e})^{i+1} \\
&= \mathbf{e}^2 + \sum_{\ell=1}^{i} 2^{\ell}\mathbf{e}^{\ell+2} + (2\mathbf{e})^{i+1} \quad \text{where } \ell = j+1 \\
&= \sum_{\ell=0}^{i} 2^{\ell}\mathbf{e}^{\ell+2} + (2\mathbf{e})^{i+1},
\end{aligned}
$$

so the formula is true for $i+1$. Therefore, by mathematical induction, (4.5.5) holds. Finally, since $u_i \geq 2\mathbf{e} + g_{i-1}$, we obtain a general lower bound on $u_i$ as required. ∎

In particular, we note a minor corollary to Theorem 4.5.2 for later use.

**Corollary 4.5.3.** For $2 \leq i \leq n$, the depth of the $i^{th}$ output of the full adder circuit is at least 1. Thus, if each input has noise at least $\mathbf{e}$, the resulting noise on each output satisfies $u_i \geq \mathbf{e}^2$.

We note that bounding the full adder circuit's depth by 1 is a significant underestimate, based on Figure 4.1. We also note that although the noise on $u_1$ does not satisfy this bound, the noise on every other bit is significantly more than $\mathbf{e}^2$, so we could include $u_1$ in the above corollary without issue.

### 4.5.2 Total noise on the circuit: adding Θ numbers

Before we describe what happens when we add $\Theta$ elements using the ripple-carry adder rather than only two, let us examine our noise term $\mathbf{e}$. Let $\mathrm{c}$ be a ciphertext. We chose parameters in Section 2.4.4 such that the distribution of $N(\mathrm{c})$ is uniform over $[-2^{\rho}, 2^{\rho}]$ and has a density function equal to $2^{-\rho'-1}$ over this interval. In this section, we assume the noise on the encryptions of each bit of $x$ and $y$ is at least 4 in absolute value in order to estimate a corresponding lower bound on the error introduced by the encryption of $F$. This is valid since

$$P(|N(\mathrm{c})| \leq 4) = 8 \cdot 2^{-\rho'-1} = \frac{1}{2^{\rho'-2}} < \frac{1}{2^{\lambda-2}},$$

that is, the probability that $|N(\mathrm{c})|$ is less than 4 is negligible.

To add $\Theta$ numbers efficiently, one adds the elements pairwise, implying that each element goes through the ripple-carry adder $\log\Theta$ times. Assuming each encrypted element carries a minimum noise of $\mathbf{e}$, and using the extremely conservative estimate on the total noise on the ripple-carry adder circuit from Corollary 4.5.3, this yields a noise term of $(((\mathbf{e}^2)^2)^{\cdots})^2$ ($\log\Theta$ times); that is, a total noise term of at least $\mathbf{e}^{2^{\log\Theta}} = \mathbf{e}^{\Theta}$.

With the assumption above that $\mathbf{e} > 4$, the total noise on the circuit then satisfies $\mathbf{e}^{\Theta} \geq 2^{2\Theta}$.

The decryption gives a valid ciphertext if and only if $\mathbf{e}^{\Theta} < 2^{\eta-1}$. Since we have $\eta < \gamma$ from (2.4.9), this implies

$$2^{2\Theta} < 2^{\gamma}.$$

Under these circumstances, we would in practice have $2\Theta < \kappa - 2$ by (4.3.5), so

$$\Theta < \kappa\lambda,$$

contradicting the security constraint (4.3.7). It is therefore infeasible to use the ripple-carry adder to evaluate the decryption circuit as it introduces too much noise.

### 4.5.3 Further analysis of the addition function

The main problem with the ripple-carry adder as a circuit is that the addition of two integers implies *at least* one multiplication on each pair of bits, and usually significantly more. This gives our circuit a depth of at least $\Theta$, but there exist elements of depth much greater than that. This means we have

$$\begin{aligned} d &\geq \Theta \\ &\geq \kappa\lambda \quad \text{by (4.3.7)} \\ &\geq (\gamma + 2)\lambda \quad \text{by (4.3.5)} \\ &\geq \eta^2\lambda^2 \quad \text{by (2.4.11).} \end{aligned}$$

This clearly shows that the depth of the ripple-carry adder circuit is such that it is impossible to satisfy the depth constraint in (3.4.3). There is simply too much noise accumulation for the ripple-carry adder to be a feasible implementation of $F$.

The DGHV authors [31] modify the function $F$ defined by (4.4.2) using a mix of classical and more sophisticated techniques to reduce the depth of the decryption circuit such that it may be evaluated by their scheme. They first explain how to reduce the number of bitwise additions performed by using the classical "three-for-two trick", as described in [21]. They then explain that with just this technique, the depth of their circuit is still too large, so they follow work by Gentry [14], [16], who uses the fact that all but $\theta$ of the $\mathtt{s}_i z_i$ are in fact 0 to his advantage. He writes each $\mathtt{s}_i z_i$ as a row vector of a matrix, creating a $\Theta$ by $n + 1$ matrix. Gentry's method then involves using the Hamming weight of each column to create a new $(n + 1)$-bit number such that the sum of its bits is equal to the sum of the $\mathtt{s}_i z_i$. This reduces the addition of $\Theta$ elements to the addition of $n + 1$ elements, which, by (4.3.6) and taking $\theta = \lambda$, can be chosen to be as small as $\lceil \log \theta \rceil + 4 = \lceil \log \lambda \rceil + 4$.

These methods enable van Dijk et al. to reduce the depth of their decryption circuit to

$$d \leq 64\lambda \log^2 \lambda. \tag{4.5.6}$$

| Equation | Reference |
|---|---|
| $\lambda \leq \rho < \rho' < \eta < \gamma < \tau$ | (2.4.13) |
| $\frac{\eta}{2} - 5 \geq \rho' \geq \rho + \log(\tau + 1)$ | (3.3.2) and (2.4.12) |
| $\gamma \geq \lambda\eta^2$ | (2.4.11) |
| $\tau \geq \gamma + \lambda$ | (2.4.7) |
| $d < \frac{\eta-3}{\rho'+3} - 1$ | (3.4.3) |
| $n > \log\theta + 3 \geq \log\lambda + 3$ | (4.3.6) and (4.3.8) |
| $\Theta \geq \kappa\lambda \geq (\gamma + 2)\lambda$ | (4.3.7) and (4.3.5) |

Table 4.2: Summary of all our parameter constraints when realizing the DGHV scheme

Most importantly, their modifications to the decryption circuit are such that the depth is dependent on $n$ and therefore $\theta$, rather than $\eta$. This allows the DGHV authors to choose their parameters a little more freely without the issue of circular constraints that we encountered in Section 3.4.

We will see in the next section that this depth, combined with their choice of parameters, is in fact small enough to make their scheme bootstrappable, thus completing their fully homomorphic scheme over the integers.

## 4.6 Comparing Parameters

Throughout this thesis, we have provided an independent analysis of the constraints to be imposed on the various parameters of the DGHV scheme in order to respect both functionality and security. In some cases, our assumptions and therefore our results differ from those proposed by the authors of the DGHV scheme. We summarize in Table 4.2 all parameter constraints we have used in this thesis. We compare this to Table 4.3, which summarizes all parameter constraints listed in [31] to realize the DGHV scheme. We note that they use standard complexity notation $(\boldsymbol{\Theta}, \omega)$ to describe the order of growth of their parameters (see [25, Ch.1, Sec.2.1]). We have used a bold $\boldsymbol{\Theta}$ to differentiate the notation from the variable $\Theta$.

In addition to detailing their constraints, the authors of the DGHV scheme propose a set of "ideal" parameters that satisfy their constraints. We describe each of their proposed parameters for the scheme with respect to the base security parameter $\lambda$ in Table 4.4. We note that $\eta = \tilde{O}(\lambda^2)$ (read as "soft-O") means that there exists some $k \in \mathbb{N}$ such that $\eta = O(\lambda^2 \log^k(\lambda^2))$ (see [9, Sec.3.2, Pr.3.5]). In practice, we will say $\eta = c\lambda^2 \log^k \lambda$ for some positive integers $c$ and $k$.

We see immediately that the "ideal" values for $\rho, \tau, \theta, n, \kappa$, and $\Theta$ satisfy our conditions. Setting $\eta = c\lambda^2 \log^k \lambda$, we get

$$\gamma \geq \eta^2\lambda = c^2\lambda^5 \log^{2k} \lambda \in \tilde{O}(\lambda^5)$$

| Constraint |
|---|
| $\rho = \omega(\log \lambda)$ |
| $\eta \geq \rho \cdot \mathbf{\Theta}(\lambda \log^2 \lambda)$ |
| $\gamma = \omega(\eta^2 \log \lambda)$ |
| $\tau \geq \gamma + \omega(\log \lambda)$ |
| $\rho' = \rho + \omega(\log \lambda)$ |
| $d \leq \frac{\eta - 4 - \log a}{\rho' + 2}$ |
| $\kappa = \frac{\gamma\eta}{\rho'}$; if optimizing, $\kappa = \gamma + 2$ |
| $\theta = \lambda$ |
| $\Theta = \omega(\kappa \cdot \log \lambda)$ |
| $n = \lceil \log \theta \rceil + 3$ |

Table 4.3: Parameter constraints of the DGHV scheme by its authors

| | | |
|---|---|---|
| $\rho = \lambda;$ | $\rho' = 2\lambda;$ | $\eta = \tilde{O}(\lambda^2);$ |
| $\gamma = \tilde{O}(\lambda^5);$ | $\tau = \tilde{O}(\lambda^5) + \lambda;$ | $\theta = \lambda;$ |
| $n = \lceil \log \lambda \rceil + 3;$ | $\kappa = \tilde{O}(\lambda^5) + 2;$ | $\Theta = \lambda(\tilde{O}(\lambda^5) + 2).$ |

Table 4.4: The "ideal" parameters suggested by the DGHV authors, where $\tilde{O}$ notation is a variation of big O notation that ignores logarithmic factors.

as required. If $\lambda > 6$, the we have $2\rho' + 10 = 4\lambda + 10 < \lambda^2 < \eta$ and (3.3.2) is satisfied. Finally, if $\tau = \gamma + \lambda = c^2\lambda^5 \log^{2k} \lambda + \lambda$ then the condition $\rho' \geq \rho + \log(\tau + 1)$ from (2.4.12) is equivalent to

$$\lambda \geq \log(c^2\lambda^5 \log^{2k} \lambda + \lambda + 1),$$

which will certainly be satisfied for $\lambda$ large enough. For example, we see, using graphing software, that if $c = 256$ and $k = 2$, we need $\lambda > 55$; if $c = 4$ and $k = 4$, we require $\lambda > 52$. Therefore, for sufficiently large $\lambda$ and correct choices of $\eta, \gamma$, and $\tau$, the "ideal" parameters satisfy all the constraints we have found, aside from depth.

Finally, we examine van Dijk et al.'s suggested parameters with respect to their allowed upper bound on the depth of a circuit and what they stated to be their final depth for the decryption circuit. Recall from Section 3.4 the DGHV authors' formula for the allowed depth of a circuit:

$$d \leq \frac{\eta - 4 - \log a}{\rho' + 2},$$

where $a$ represents the number of additions performed. Using their "ideal" parameters and stated depth of circuit, we therefore need

$$(64\lambda \log^2 \lambda)(2\lambda + 2) \leq c\lambda^2 \log^k \lambda - 4 - \log a.$$

We see that, since $\log a$ is not significantly large compared to the other values, we are able to choose $c$ and $k$ such that this inequality holds for reasonably-sized $\lambda$. For analysis purposes, we assume $\log a < 6$. If we take $c = 256$ and $k = 2$, for example, this inequality is true for $\lambda > 1$; if $c = 4$ and $k = 4$ then the above is true for $\lambda > 52$ [32]. We therefore see, by taking the second scenario, that if $\lambda > 52$, all the constraints are satisfied.

It is this ability to choose $c$ and $k$ and modify $\eta$ that enables the DGHV scheme to be fully homomorphic. We have seen in some literature (such as [8]) a misinterpretation of the $\tilde{O}$ notation; some authors have misquoted van Dijk et al.'s "ideal" parameter $\eta$ as being equal to $\lambda^2$. We see an immediate contradiction when using this value to compare the depth of the decryption circuit from (4.5.6) to the allowed upper bound; we must have

$$64\lambda \log^2 \lambda \leq \frac{\lambda^2 - 4 - \log a}{2\lambda + 2},$$

which is simply not possible.

### Alice chooses her final parameters

Alice has now realized a security parameter of $\lambda = 10 < 52$ is too small to satisfy all her constraints, but she is still curious to see the sizes of all van Dijk et al.'s "ideal" parameters from [31] with $\lambda = 10$. Ignoring the constraint (2.4.12), she chooses her parameters as follows: $\rho$ and $\theta$ first; then $n, \eta$; then $\gamma$; then $\kappa, \tau$; and finally $\rho'$ and $\Theta$. Using Table 4.4 and $c = 4$ and $k = 4$ from the previous analysis, she obtains the following parameters:

$$\begin{array}{lll} \rho = 10; & \rho' = 20; & \eta = 48711; \\ \gamma \approx 2.37 \times 10^{10}; & \tau \approx 2.37 \times 10^{10}; & \theta = 10; \\ n = 7; & \kappa \approx 2.37 \times 10^{10}; & \Theta \approx 2.37 \times 10^{11}. \end{array}$$

She observes that, even for such a small security parameter as $\lambda = 10$, her resulting public key, comprising over 284 billion elements in total, each one almost 3GB in length, is massive (over 84 million TB in size). She resolves to research methods of making the DGHV scheme more efficient, while still being secure and preserving its novel and potentially transformative homomorphic properties.

# Appendix A

# Matlab Code

In order to understand the inner workings of the homomorphic encryption scheme presented in [31], we used Matlab to code a small toy example of the scheme. The Matlab code is divided into multiple scripts and functions to represent each piece of the scheme. Below are the workings of each script and function and how they are related to each other. To realize this example, we used Matlab (versions 2015a and 2015b) with the addition of the Variable Precision Integer Arithmetic (vpi) toolbox. The vpi package allowed us to work with much larger integers without truncating their precision. Anyone interested in viewing the actual Matlab code may contact the author.

## A.1 KeyGen

The KeyGen script sets up the necessary parameters for the scheme and then creates the private key as well as the public key to be used for the encryption. One sets the base parameters $\lambda, \rho, \rho', \eta, \gamma$, and $\tau$. The script then generates a uniformly random $p \in (2^{\eta-1}, 2^{\eta}) \cap (2\mathbb{Z}+1)$, which becomes the private key.

To create the public key, the KeyGen script initializes $x, q$, and $r$, all $1 \times (\tau+1)$ matrices. Then, for each $i = 0, \ldots \tau$, the matrices are assigned the following values:

- The $i^{th}$ entry of $q$ is assigned a uniformly random integer from $[0, 2^{\gamma}/p]$;
- The $i^{th}$ entry of $r$ is assigned a uniformly random integer from $(-2^{\rho}, 2^{\rho})$;
- The $i^{th}$ entry of $x$ becomes $pq_i + r_i$.

The next step is to find the largest value of the matrix $x$ and its associated entry in $r$. Let $I$ be the index of the maximum value of $x$. For the encryption and decryption to work, as described in Section 2.1.3, we must have $x_I$ odd and $r_I$ even. If this is not true, the loop to create $x$ is restarted to maintain the randomness of

creating such a matrix. For testing purposes, a count variable has been added to see how many times the script needs to recreate $x$ until the conditions are met.

Once $x$ and $r$ meet the required constraints, $x_I$ and $x_0$ are swapped, as are $r_I$ and $r_0$. This new matrix $x$ is the public key to be used by all other scripts and functions.

This script, when run, outputs the parameters for encryption, the private key $p$, and the public key $x$, including how it was built, i.e. the $q_i$'s and $r_i$'s used. This is for testing purposes only; in the true encryption scheme, the $q_i$'s, $r_i$'s, and $p$ are private information.

## A.2 Encrypt

This is a function called by other scripts in order to encrypt a bit and return a ciphertext. It receives as inputs the public key $x$, the message $\mathtt{m}$, and $\rho'$ the secondary noise parameter in order to generate noise on the encryption.

The encrypt function first creates a random subset of the public key (not including the first, and largest, element $x_0$) by generating a random bit string of length $\tau$, called $s$. The ones in the bit string correspond to the elements of the public key to use in the subset. We call the subset $S = \{i \mid s_i = 1\}$. We also choose the secondary noise $r'$ uniformly randomly from $\mathbb{Z} \cap (-2^{\rho'}, 2^{\rho'})$. Finally, the function encrypts the message $\mathtt{m}$. The ciphertext is

$$\mathrm{c} = \mathtt{m} + 2r' + 2\sum_{i \in S} x_i \mod x_0.$$

The encryption then outputs $\mathrm{c}, r'$, and $s$ to its calling script. It only needs to output $\mathrm{c}$; the other two variables are given for testing.

## A.3 Decrypt

This function is incorporated into the Hom_Enc_Z script in Section A.5. To decrypt a ciphertext $\mathrm{c}$, first reduce it modulo $p$ the private key, using the cmod function from Section A.4, then reduce the result modulo 2 to obtain $\mathtt{m}' \in \{0, 1\}$. Decryption is valid when $\mathtt{m}' = \mathtt{m}$.

## A.4 Cmod

This function ensures that the modular reduction necessary in the decryption process works properly. The cmod function takes as input the ciphertext $\mathrm{c}$ and the private key $p$. It then evaluates $\mathrm{c} \bmod p$ and verifies that the result is less than $p/2$. If it is, the function outputs the result directly. If the result is greater than $p/2$, cmod subtracts $p$ from it, outputting a result less than $p/2$ in absolute value.

## A.5 Hom_Enc_Z

This script is the main script that one would use to demonstrate the DGHV scheme. It first calls KeyGen from Section A.1 in order to obtain the parameters and public key required for it to encrypt and decrypt a bit. Next, we generate a uniformly distributed pseudorandom number from the interval $(0, 1)$ and then round it to the nearest integer, thus generating a pseudorandom $\mathtt{m} \in \{0, 1\}$ for the message.

The script then sends $\mathtt{m}, x$, and $\rho'$ (which is from the KeyGen script) to the encrypt function from Section A.2 and receives a ciphertext $\mathrm{C}$, as well as (for testing) the noise $r'$ on the encryption and $s$ the random bit string generated to create a random subset of the public key. Finally, to ensure that the decryption is valid, the script decrypts the ciphertext to recover a 0 or 1, which should match $\mathtt{m}$.

For testing purposes, the script evaluates the noise on the ciphertext $\mathrm{C}$, defined as in (2.1.4):

$$N(\mathrm{C}) = 2r' + 2\sum_{i \in S} r_i - z \cdot r_0.$$

If the absolute value of $N(\mathrm{C})$ is greater than $p/2$, the decryption process risks returning a different value than that of the original message $\mathtt{m}$.

At the end of this script, the user is given the message, its corresponding ciphertext, and the subset used to encrypt it.

## A.6 Error_calc

Error_calc is a script that was created to be able to test the validity of the decryption scheme (whether the decrypted bit matched the original message) over multiple runs. The main purpose of this script is to test the various parameters used in the encryption scheme.

This script runs the Hom_Enc_Z script a predetermined number of times (we chose 300) and for each run, it verifies whether the message and the decrypted ciphertext match. If they don't, a message is printed, containing the values of the message $\mathtt{m}$, the decrypted ciphertext $\mathtt{m}'$, $p$, $N(\mathrm{C})$, and the size of the subset $S$ used to encrypt $\mathtt{m}$. A counting variable called $err$ is then incremented to be able to see how many times the decryption failed over the prescribed number of runs.

A second check is done during each of the runs: if $|N(\mathrm{C})| > p/2$, a message is printed to that regard and a separate variable $errc$ is incremented. At the end of the predetermined number of runs, the user can quickly see how many times the decryption failed, as well as how many times $|N(\mathrm{C})|$ went over $p/2$.

## A.7 Evaluate

This script was designed to be able to encrypt two messages and then compute their sum and product, then check if the corresponding ciphertexts are valid. It begins by calling KeyGen in order to obtain the necessary parameters and a public key. It then creates a first message in the same manner as did Hom_Enc_Z, that is, by rounding a pseudorandom number between 0 and 1 to the nearest integer. It sends the required information to the encrypt function as before and receives a first ciphertext. The process of creating a message and encrypting it is then repeated, thus giving two messages and their associated ciphertexts. The messages are then added (or multiplied) modulo 2, and their ciphertexts are added (multiplied) as integers. Optimization is performed if necessary (see Section A.8). The ciphertexts designating the sum/product are then decrypted and are checked versus the sum/product of the original messages to see if they match.

## A.8 Optimize

The optimize function is called when the ciphertext in another script becomes larger than $2^\gamma$. The function receives as inputs $\gamma, \rho$, and $p$, in order to create new elements of the form $x'_i = 2\left(q'_i \cdot p + r'_i\right)$, where $r'_i \in \mathbb{Z} \cap (-2^\rho, 2^\rho)$ (as before) and $q'_i \in \mathbb{Z} \cap \left[\frac{2^{\gamma+i-1}}{p}, \frac{2^{\gamma+i}}{p}\right)$, for $i = 0, \ldots, \gamma$, as described in Section 3.3.1. This gives a matrix $x'$ with $\gamma + 1$ elements for which $x'_i \in [2^{\gamma+i}, 2^{\gamma+i+1})$.

The function also takes as input the ciphertext c, which it compares to each element of $x'$ until it finds an element $M$ such that $x'_{M+1} >$ c. Next, it reduces the ciphertext by computing c modulo $x'_M$, then modulo $x'_{M-1}$, and so on for all remaining elements of $x'$. After all reductions, the optimized ciphertext c′ is returned to the calling script.

## A.9 Other Scripts and Functions

We created a number of other scripts and functions to test various portions of the encryption scheme, validate parameter sets, analyze noise distribution, and attempt to recreate some other parts of the DGHV paper [31]. Some notable examples include:

- Find_bad: A script created to run Hom_Enc_Z over and over, checking the message m versus the decrypted ciphertext m′ each time and only stopping when the two did not match. This was designed so that it would run an infinite loop if the parameter set was valid. The script also kept track of how many times Hom_Enc_Z was iterated and how much time it had been running. We later modified it to perform the same checks on the evaluate script.

- mod_coefficients: We designed a modification to the optimize function to run it as a script in order to perform further testing. This script runs optimize as a script and for each modular reduction computed, it keeps track of the coefficient. The sum of all coefficients is also calculated and stored. Analysis of the coefficients and their sums generated over multiple iterations of the optimize script allowed us to validate our noise accumulation analysis in Section 3.3.4.

- Brute_force_factors: We explored how one would go about attacking the DGHV scheme, based on known attacks described by van Dijk et al. [31]. This script took a number as input and stored the set of its prime factors. It then generated all possible subsets of the factors and computed the product of all elements in each subset. The products that fell within the proper range (that is, those of the same bit-length as the private key) were collected for further analysis. Using this script over all possible noise values of a ciphertext gave us an idea of how a brute-force attack would work, and how complex it would be.

- Dist_noise: This script was created to analyze the noise distribution of $r$ and $r'$ within the DGHV scheme. We input $\rho, \rho'$, and $\tau$ and the script generated $\tau$ values $r \in (-2^{\rho}, 2^{\rho})$. Then, over multiple runs (we chose 1000), the script generated a random bit string to take a random subset of the $r$'s and compute its sum. It also generated an $r' \in (-2^{\rho'}, 2^{\rho'})$ for each iteration. We then compared the distribution of the subset sums with the distribution of the subset sums with an $r'$ added to each. This script generated the histograms found in Section 2.4.4.

# Bibliography


[1] Aggarwal, D., Maurer, U.: *Breaking RSA Generically is Equivalent to Factoring*, Joux, A. (Ed.): EUROCRYPT 2009, LNCS 5479, pp. 36–53, 2009.

[2] Ajtai, M.: *The Shortest Vector Problem in $L_2$ is NP-hard for Randomized Reductions*, Proceedings of the 30th annual ACM Symposium on Theory of Computing, pp. 10–19, 1998.

[3] Barker, E., Roginsky, A.: *Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths*, NIST Special Publication 800–131A Revision 1, http://dx.doi.org/10.6028/NIST.SP.800-131Ar1, last accessed January 2017.

[4] Beaver, D.: *Minimal-Latency Secure Function Evaluation*, Preneel, B. (Ed.): EUROCRYPT 2000, LNCS 1807, pp. 335–350, 2000.

[5] Black, N.: *Homomorphic Encryption and the Approximate GCD Problem*, PhD thesis, Clemson University, 2014.

[6] Chen, H., Laine, K., Player, R.: *Simple Encrypted Arithmetic Library – SEAL v2.1*, Microsoft Research TechReport MSR-TR-2016-68, https://www.microsoft.com/en-us/research/publication/simple-encrypted-arithmetic-library-seal-v2-1/, last accessed January 2017.

[7] Chen, Y., Nguyen, P.: *Faster Algorithms for Approximate Common Divisors: Breaking Fully-Homomorphic-Encryption Challenges over the Integers*, Pointcheval, D., Johansson, T. (Eds.): EUROCRYPT 2012, LNCS 7237, pp. 502–519, 2012.

[8] Cohn, H., Heninger, N.: *Approximate Common Divisors via Lattices*, Proceedings of the Tenth Algorithmic Number Theory Symposium, pp. 271–293, 2013.

[9] Cormen, T., Leiserson, C., Rivest, R., Stein, C.: *Introduction to Algorithms*, MIT Press, Cambridge, MA, 2009.

[10] Coster, M., Joux, A., LaMacchia, B., Odlyzko, A., Schnorr, C.-P., Stern, J.: *Improved Low-Density Subset Sum Algorithms*, Computational Complexity 2, pp. 111–128, 1992.

[11] Diffie, W., Hellman, M.: *New Directions in Cryptography*, IEEE Transactions on Information Theory 22(6), pp. 644–654, 1976.

[12] Dowlin, N., Gilad-Bachrach, R., Laine, K. Lauter, K. Naehrig, M., Wernsing, J.: *Manual for Using Homomorphic Encryption for Bioinformatics*, Microsoft Research TechReport MSR-TR-2015-87, https://www.microsoft.com/en-us/research/publication/manual-for-using-homomorphic-encryption-for-bioinformatics/, last accessed January 2017.

[13] Elkadi, M., Galligo, A., Ba, T.L.: *Approximate GCD of Several Univariate Polynomials with Small Degree Perturbations*, Journal of Symbolic Computation 47, pp. 410–421, 2012.

[14] Gentry, C.: *A Fully Homomorphic Encryption Scheme*, PhD thesis, Stanford University, 2009.

[15] Gentry, C.: *Computing Arbitrary Functions of Encrypted Data*, Communications of the ACM 53(3), pp. 97–105, 2010.

[16] Gentry, C.: *Fully Homomorphic Encryption Using Ideal Lattices*, Proceedings of the 2009 ACM International Symposium on Theory of Computing, pp. 169–178, 2009.

[17] Goldwasser, S., Micali, S.: *Probabilistic Encryption*, Journal of Computer and System Sciences 28(2), pp. 270–299, 1984.

[18] Halevi, S.: *HElib – An Implementation of homomorphic encryption*, GitHub Inc., https://github.com/shaih/HElib, last accessed January 2017.

[19] Hoffstein, J., Pipher, J., Silverman, J.: *An Introduction to Mathematical Cryptography*, Springer, New York, NY, 2008.

[20] Howgrave-Graham, N.: *Approximate Integer Common Divisors*, Silverman, J.H. (Ed.): CaLC 2001, LNCS 2146, pp. 51–66, 2001.

[21] Karp, R.M., Ramachandran, V.: *A Survey of Parallel Algorithms for Shared-Memory Machines*, Technical Report CSD-88-408, UC Berkeley, 1988.

[22] Katz, J., Lindell, Y.: *Introduction to Modern Cryptography*, Chapman Hall/CRC, Boca Raton, FL, 2008.

[23] Knuth, D.: *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1969.

[24] Mano, M.: *Digital Logic and Computer Design*, Prentice-Hall, Englewood Cliffs, NJ, 1979.

[25] Micciancio, D., Goldwasser, S.: *Complexity of Lattice Problems: A Cryptographic Perspective*, Kluwer Academic Publishers, Boston, MA, 2002.

[26] Nagasaka, K.: *Approximate Polynomial GCD over Integers*, Journal of Symbolicc Computation 46, pp. 1306–1317, 2011.

[27] Nguyen, P., Stern, J.: *Adapting Density Attacks to Low-Weight Knapsacks*, Roy, B. (Ed.): ASIACRYPT 2005, LNCS 2388, pp. 41–58, 2005.

[28] Rivest, R., Adleman, L., Dertouzos, M.: *On Data Banks and Privacy Homomorphisms*, Foundations of Secure Computation, pp. 169–177, 1978.

[29] Sander, T., Young, A., Yung, M.: *Non-Interactive CryptoComputing for $NC^1$*, Proceedings of the 40th annual IEEE Symposium on Foundations of Computer Science, pp. 554–567, 1999.

[30] Siegrist, K.: *Random: Probability, Mathematical Satistics, Stochastic Processes*, http://www.math.uah.edu/stat/index.html, online textbook hosted by the University of Alabama in Huntsville, 2015.

[31] van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: *Fully Homomorphic Encryption over the Integers*, H. Gilbert (Ed.): EUROCRYPT 2010, LNCS 6110, pp. 24–43, 2010.

[32] Wolfram Alpha LLC, Wolfram—Alpha, http://www.wolframalpha.com, 2017.