

**A STUDY ON PARTIALLY HOMOMORPHIC ENCRYPTION
SCHEMES**

by

Shifat P. Mithila

A Thesis Submitted to the Faculty of
The Charles E. Schmidt College of Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

Florida Atlantic University

Boca Raton, FL

May 2017

Copyright 2017 by Shifat P. Mithila


A STUDY ON PARTIALLY HOMOMORPHIC ENCRYPTION SCHEMES

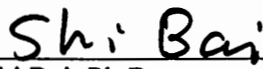
by

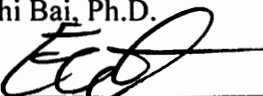
Shifat P. Mithila


This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Koray Karabina, Department of Mathematical Sciences, and has been approved by the members of her supervisory committee. It was submitted to the faculty of the Charles E. Schmidt College of Science and was accepted in partial fulfillment of the requirements for the degree of Master of Science.


SUPERVISORY COMMITTEE:


Koray Karabina, Ph.D.
Thesis Advisor

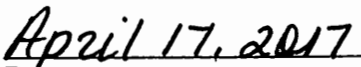

Shi Bai, Ph.D.


Edoardo Persichetti, Ph.D.


Rainer Steinwand, Ph.D.
Chair, Department of Mathematical Sciences


Ata Sarajedini, Ph.D.
Dean, the Charles E. Schmidt College of
Science


Deborah L. Floyd, Ed.D.
Dean, Graduate College


Date

ACKNOWLEDGEMENTS

The author would like to thank her thesis advisor Dr. Koray Karabina, Assistant Professor, Department of Mathematical Sciences, for his support, encouragement, time and dedication to this research. The author would also like to express her appreciation to Dr. Shi Bai, and Dr. Edoardo Persichetti for their valuable inputs, reviews, and suggestions in the research.

ABSTRACT

Author: Shifat P. Mithila
Title: A Study on Partially Homomorphic Encryption Schemes
Institution: Florida Atlantic University
Thesis Advisor: Dr. Koray Karabina
Degree: Master of Science
Year: 2017

High processing time and implementation complexity of the fully homomorphic encryption schemes intrigued cryptographers to extend partially homomorphic encryption schemes to allow homomorphic computation for larger classes of polynomials. In this thesis, we study several public key and partially homomorphic schemes and discuss a recent technique for boosting linearly homomorphic encryption schemes. Further, we implement this boosting technique on CGS linearly homomorphic encryption scheme to allow one single multiplication as well as arbitrary number of additions on encrypted plaintexts. We provide MAGMA source codes for the implementation of the CGS scheme along with the boosted CGS scheme.

A STUDY ON PARTIALLY HOMOMORPHIC ENCRYPTION SCHEMES

List of Tables	ix
1 Introduction	1
2 Public Key Encryption Schemes	5
2.1 Introduction	5
2.2 Description	6
2.3 RSA Encryption Scheme	8
2.3.1 Introduction	8
2.3.2 Description	8
2.3.3 Security of the RSA encryption scheme	11
2.3.4 Efficiency of the RSA encryption scheme	12
2.3.5 RSA scheme in practice	13
2.4 ElGamal Encryption Scheme	13
2.4.1 Introduction	13
2.4.2 Description	14
2.4.3 Security of the ElGamal encryption scheme	16
2.4.4 Efficiency of the ElGamal encryption scheme	19
2.4.5 ElGamal encryption scheme in practice	20
2.5 Remarks	20
3 CGS Encryption Scheme	21
3.1 Introduction	21

3.2	Description	22
3.2.1	Key generation function for the CGS scheme ($\text{KeyGen}_{\text{CGS}}$) . .	24
3.2.2	Encryption function for the CGS scheme (Enc_{CGS})	25
3.2.3	Evaluation functions for the CGS scheme (Eval_{CGS})	26
3.2.4	Decryption function for the CGS scheme (Dec_{CGS})	33
3.3	Security of the CGS encryption scheme	34
3.4	Efficiency of the CGS encryption scheme	35
3.5	Remarks	35
4	Boosting Linearly Homomorphic Encryption Scheme	37
4.1	Introduction	37
4.2	Description	37
4.2.1	Key generation function for the Boosted-LHE scheme (KeyGen_{B})	38
4.2.2	Encryption function for the Boosted-LHE scheme (Enc_{B}) . . .	38
4.2.3	Evaluation functions for the Boosted LHE scheme (Eval_{B}) . .	39
4.2.4	Decryption functions for the Boosted LHE scheme (Dec_{B}) . . .	49
4.3	Security of the Boosted LH encryption scheme	51
4.4	Efficiency of the Boosted LH encryption scheme	51
4.5	Remarks	52
5	Boosted-CGS Encryption Scheme	54
5.1	Introduction	54
5.2	Description	54
5.2.1	Key generation function for the Boosted-CGS scheme (KeyGen_{B})	55
5.2.2	Encryption function for the Boosted-CGS scheme (Enc_{B}) . . .	56
5.2.3	Evaluation functions for the Boosted-CGS scheme (Eval_{B}) . .	57
5.2.4	Decryption functions for the Boosted-CGS scheme (Dec_{B}) . . .	61
5.3	Security of the Boosted CGS encryption scheme	64
5.4	Efficiency of the Boosted CGS encryption scheme	64

5.5	Remarks	65
6	Conclusion	66
	Bibliography	67

LIST OF TABLES

2.1	Construction (The Public-key Encryption Scheme)	7
2.2	Construction (The Textbook RSA Scheme)	9
2.3	Efficiency of Textbook RSA Scheme	12
2.4	Construction (The ElGamal Encryption Scheme)	15
2.5	Efficiency of ElGamal Encryption Scheme	19
3.1	Efficiency of CGS Encryption Scheme	36
4.1	Efficiency of Boosted LHE Encryption Scheme	53

CHAPTER 1

INTRODUCTION

The purpose of this thesis is to boost additively homomorphic encryption schemes to allow single multiplication operation over the ciphertexts. This is an approach to enable a partially homomorphic encryption scheme to perform other homomorphic encryptions it usually doesn't follow.

In cryptography, one of the most intriguing questions is the problem of enabling computations over the encrypted data or ciphertexts. This has been attracting noteworthy attentions of the cryptographers and studied extensively by the researchers lately. In simple words, this problem is between two parties: the client and the server, where the client has the input m and the server has the hold of some function f . The client would try to learn about $f(m)$ while leaking no information about m and outsourcing the computation on a server. This is known as *semantic security*. In some applications, the server may require to maintain *circuit privacy*, i.e. the server will perform the computations without revealing any information about f (beyond $f(m)$) to the client. *Compactness* is also desirable in such computations which means the communication between the client and the server as well as the amount of work to be done by the client is supposed to be at the minimum level [4].

To address the above problem, cryptographers seem naturally to rely on some encryption mechanisms called homomorphic encryption schemes. These schemes have very useful features in modern communication systems and play a crucial role in various applications such as cloud computing, election scheme, multiparty computations, secret sharing schemes, threshold schemes, zero-knowledge proofs, protection of mobile agents, water marking and fingerprint schemes, oblivious transfer, mix-nets, com-

commitment schemes, lottery protocols, encrypted database search etc.[17]. Informally, homomorphic encryption schemes allow combining or comparing of messages without revealing the messages. A *homomorphic encryption* is an encryption mechanism where certain computations (e.g. additions and multiplications) can be performed on the encrypted plaintexts, generating a ciphertext such that when this ciphertext is decrypted, it matches the result of operations performed on the plaintexts. For simplicity, we assume that our message space $(\mathcal{M}, +, \cdot)$ is the set of integers where $+$ and \cdot are usual addition and multiplication operations on it. A homomorphic encryption is said to be *additive* if the encryption of the summation of the messages can be computed as a function of the encryption of the messages. This particular function can be regarded as the addition operation on the ciphertexts. We denote this by \boxplus . That is, for messages $m_1, m_2 \in \mathcal{M}$, this gives $\text{Enc}(m_1 + m_2) = \text{Enc}(m_1) \boxplus \text{Enc}(m_2)$. Here, Enc stands for the encryption function and \boxplus is the addition operation in ciphertext space C . Similarly, a homomorphic encryption is said to be *multiplicative* homomorphic encryption if the encryption of the multiplication of the messages can be computed as a function of the encryption of the messages. This function can be regarded as the multiplication operation on the ciphertexts which mathematically we denote by \boxtimes . That is, for messages $m_1, m_2 \in \mathcal{M}$, this gives $\text{Enc}(m_1 \cdot m_2) = \text{Enc}(m_1) \boxtimes \text{Enc}(m_2)$. Here, Enc stands for the encryption function and \boxtimes is the multiplication operation in ciphertext space C . *Scalar multiplication* property can be derived from the additive homomorphic encryption as for message $m \in \mathcal{M}$ and for some scalar s we can see, $\text{Enc}(s \cdot m) = \text{Enc}(m + m + \dots + m) = \text{Enc}(m) \boxplus \text{Enc}(m) \boxplus \dots \boxplus \text{Enc}(m) = s \boxtimes \text{Enc}(m)$.

Homomorphic encryption schemes that supports arbitrary functionalities are called *fully* homomorphic encryption schemes while schemes that allow one or somewhat algebraic operations (e.g. addition or multiplication but not both) are known as *partially* or *somewhat* homomorphic encryption schemes. The idea of homomorphic encryption was first proposed by Rivest et al. in 1978 [15] though a couple of years

later their *privacy* was broken by Brickell and Yacobi [3]. After that many researchers proposed several encryption schemes that supported not arbitrary operations but still allowed some meaningful functionalities. These schemes include Goldwasser-Micali cryptosystem [9], Paillier’s cryptosystem [13] etc. But all these schemes are *linearly homomorphic* i.e. they support only linear functions. A little progress was achieved in determining if there exists any encryption scheme which is fully homomorphic, secure and efficient until 2009 when Craig Gentry came up with a solution to this central problem of cryptography. His breakthrough work theoretically demonstrated a fully-fledged homomorphic encryption scheme which is secure and efficient [8].

However, the concept of fully homomorphic encryption scheme developed by Craig Gentry is theoretically possible but sophisticated to implement. Moreover, unlike most of the homomorphic cryptosystems Gentry’s scheme didn’t use simple modular arithmetic computations, rather this scheme is lattice-based which is computationally complicated and requires very large ciphertexts compared to plaintexts. To outperform these difficulties, there have been some effort in enabling linearly homomorphic schemes to support more complex functionalities like multiplications. Boneh et al. in their paper [2] discussed this problem in the context of composite-order bilinear groups where they used bilinear map to enable one single multiplication operation on ciphertexts. But, this is a construction limited to bilinear groups and the open problem still remains that whether it can be extended to any linearly homomorphic scheme like Paillier, Goldwasser-Micali etc. Dario Catalano and Dario Fiore presented a boosting technique to generalize linearly homomorphic encryption scheme to gain the possibility to perform one single multiplication over the encrypted plaintexts [4]. Theoretically, they showed that their proposed method is capable of such computations on any linearly homomorphic scheme. In this thesis, we implement the method by Catalano et al. [4] on a public key linearly homomorphic encryption scheme called

CGS [16] by R. Cramer, R. Gennaro and B. Schoenmakers. This allows performing homomorphic encryption for quadratic functions. We provide MAGMA source code for each of the functions.

The rest of the thesis is organized as follows. In Chapter 2, we present some basic and formal discussion about public key encryption schemes and provide two examples of this scheme. These examples include the RSA scheme for historical reasons and ElGamal scheme as this scheme is the base of the CGS scheme. Chapter 3 discusses CGS encryption scheme and its construction, security, and efficiency. We provide MAGMA codes for this scheme. In Chapter 4, we describe the general boosting technique on linearly homomorphic encryption schemes, it's evaluation functions, correctness, security, and efficiency. In Chapter 5, We apply the boosting technique discussed in Chapter 4 on the CGS scheme described in Chapter 3. We also provide MAGMA source code for this implementation. We finish the thesis in Chapter 6 with some concluding remarks.

CHAPTER 2

PUBLIC KEY ENCRYPTION SCHEMES

2.1 INTRODUCTION

Cryptography is the study or process of hiding information or reading and writing secret messages/codes. These messages can be texts, binary files or documents. Two basic techniques for encrypting information in cryptography are symmetric encryption and asymmetric encryption. *Symmetric* encryption is the oldest technique that uses only one secret shared key by both the sender and the receiver to encrypt and decrypt the messages. This secret key can be a number, a word or just a string of random letters. However, in *asymmetric* cryptography which is also known as public key cryptography, a pair of keys (pk, sk) is used where pk refers to public key and sk refers to secret/private key. The *public key encryption* was initiated on the purpose to avoid meetings or any priori communication between the sender and the receiver. The sender uses the public key pk to encrypt the message and the receiver uses sk to decrypt the ciphertext and recover the original message. This technique provides essentially two ways for the receiver to learn about the public key pk [10]. One way is, the receiver would generate the key pair (pk, sk) and send the public key pk to the sender via an authenticated channel (communication by an authenticated channel means that the adversary can not modify or replace the information). Another way is, the receiver would generate the key pair (pk, sk) , independent of any sender and disseminates the pk by publishing it on her web page or her business card, any newspaper or in any public directory. In that way, public key is freely available to anyone who wants to communicate with the receiver. But public key publicizing is

still necessary to be authentic so that adversary can not make any changes to the key. Multiple senders can send messages to the receiver using the same public key whereas the secret key sk is kept secret and only the receiver knows about it. If the message is encrypted by the public key, it can only be decrypted by the matching private key from the pair (pk, sk) in the public key encryption scheme.

In this chapter, we focus our discussion on the construction of public key encryption schemes, and two significant examples of public key encryption schemes. We include RSA public key encryption scheme for historical reasons and ElGamal encryption scheme which is the building block of some homomorphic encryption schemes which we describe later on.

2.2 DESCRIPTION

A public key encryption scheme has three phases: key generation, encryption and decryption. We assume a scenario where the sender Alice is trying to send an encrypted message to the receiver Bob. In the key generation step, Bob generates a public key and secret key pair (pk, sk) . Note that, it should be infeasible to compute or generate sk by the parties other than Bob for the security of the scheme. In the encryption step, Alice receives an “authentic” copy of Bob’s public key pk and she uses pk and the encryption function to encrypt the message m and generates the ciphertext $c = \text{Enc}(pk, m)$. Next in the decryption step, Bob receives the ciphertext c and decrypts c using the secret key sk that Bob already possesses. He gets back the message $m = \text{Dec}(sk, c)$. So, the construction of the public key encryption scheme is a tuple of a probabilistic or randomized polynomial-time algorithms (PPT) $(\text{KeyGen}, \text{Enc}, \text{Dec})$ shown in Table 2.1. [A *polynomial time algorithm* is an algorithm say, A for which there exists a polynomial $p(\cdot)$ with the input $x \in \{0, 1\}^*$ where running time of A is at most $p(|x|)$. A *probabilistic algorithm* has the ability to choose elements (or bits) at “random” and use the result in its computation. If the algorithm is in polynomial

time, it can choose randomly at most a polynomial number of bits.]

Table 2.1: Construction (The Public-key Encryption Scheme)

Functions	Procedure
KeyGen :	The key generation algorithm KeyGen takes as input the security parameter 1^n and outputs a pair of keys (pk, sk) where pk =public key and sk =secret/private key. We assume for convenience that pk and sk have length at least n and n can be determined from (pk, sk) .
Enc :	The inputs for encryption algorithm Enc are a public key pk and a message m from the underlying message space \mathcal{M} which might depend on the public key. It outputs a ciphertext c and this is written as $c = \text{Enc}(pk, m)$.
Dec :	The decryption algorithm Dec takes as input a secret key sk and a ciphertext c and outputs a message m or <i>error</i> indicating <i>failure</i> . We assume without loss of generality that Dec is deterministic and is denoted as $m := \text{Dec}(sk, c)$.

It is required that $\Pr[\text{Dec}(sk, (\text{Enc}(pk, m))) = m]$ except with possibly negligible probability over (sk, pk) output by **KeyGen**(1^n) and any randomness used by **Enc** [10].

A negligible error is allowed for the decryption, for example, if a random prime needs to be chosen, a negligible probability of choosing a composite is considered. For the purpose of practical use, generally it is expected that the message space \mathcal{M} would be independent of the public key, for example $\{0, 1\}^n$ or $\{0, 1\}^*$. But there are cases where the plaintext space \mathcal{M} depends on the public key or the message space does not contain all bit strings of some fixed length. In those cases, the messages are first encoded to bit string and then the bit string is encrypted to the ciphertext. The process of encoding bit strings must be specified. This encoding process must also be

both efficient and efficiently reversible so that the receiver is able to decode the bit string to the original message after decrypting the ciphertext [10].

Concrete examples of public key encryption schemes include the RSA public key encryption scheme and the ElGamal public key encryption scheme. The following two sections discuss these popular cryptosystems along with their security and efficiency.

2.3 RSA ENCRYPTION SCHEME

2.3.1 Introduction

RSA is a public key encryption scheme whose security is based on the difficulty of integer factorization problem. This asymmetric encryption scheme is the first public-key cryptosystem and is widely used in secure data transmission. This scheme is named after Ron Rivest, Adi Shamir and Leonard Adleman who introduced this cryptosystem in 1978 [14]. A similar system was developed by Clifford Cocks, an English Mathematician, but it was not declassified until 1997 [18].

2.3.2 Description

We first describe “Textbook RSA”, which is the base of the construction of RSA algorithm. Note that “Textbook RSA” is not secure for various reasons which we describe later in section 2.3.3. The input to the “Textbook RSA” is a security parameter n and output is a modulus N which is a product of two n -bit primes p and q . The integers e, d that satisfy $ed = 1 \pmod{\phi(N)}$, where $\phi(N)$ is the Euler’s totient function and $\phi(N) = (p - 1)(q - 1)$. The construction of this algorithm becomes easier with the help of another algorithm say call **KeyGenModulus** (algorithm 2) that gives a composite number N as output along with its factorization p and q . In **KeyGenModulus**, p and q are two random primes where usually $q = 2p + 1$. The primes are chosen this way to make the factorization harder which is necessary for the security of the scheme. The encryption function of “Textbook RSA” is designed by the encryption

of a message $m \in [0, N)$ by raising it to e and taking the mod value in N . The ciphertext $c \in (0, N)$ such that $c = m^e \pmod{N}$ is decrypted by raising c to d and taking the mod value in N . However, in spite of this scheme being deterministic and insecure, it certainly demonstrates the advantage of the RSA scheme in constructing secure public key encryption schemes. To do it, we assume that the key generation algorithm of RSA is a probabilistic or randomized polynomial time(PPT) algorithm that takes 1^n as input and outputs the private key and public key.

The construction of this scheme is described in Table 2.2:

Table 2.2: Construction (The Textbook RSA Scheme)

Functions	Procedure
KeyGen :	On input 1^n run $KeyGenRSA(1^n)$ to obtain N, e, d . The public key is $\langle N, e \rangle$ and the private key is $\langle N, d \rangle$.
Enc :	On input a public key $pk = \langle N, e \rangle$ and a random message $m \in_R \mathbb{Z}_N^*$, compute the ciphertext $c := m^e \pmod{N}$.
Dec :	On input a private key $sk = \langle N, d \rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute the message $m := c^d \pmod{N}$.

Algorithm 1 “Textbook” RSA key generation

Input: Security parameter 1^n

Output: N, e, d such that $ed = 1 \pmod{\phi(N)}$

$(N, p, q) \leftarrow \text{KeyGenModulus}(1^n)$

Compute $\phi(N) := (p - 1)(q - 1)$

Choose a random e such that $\gcd(e, \phi(N)) = 1$

Compute d such that $d := e^{-1} \pmod{\phi(N)}$

return N, e, d

We now see that the RSA scheme is correct by the following theorem.

Algorithm 2 “Textbook” RSA KeyGenModulus

Input: Security parameter 1^n

Output: N, p, q such that $N = pq$

Choose a random number p

Choose a random number q

Compute $N = pq$

return N, p, q

Algorithm 3 “Textbook” RSA Encryption

Input: Public key (N, e) , the message m

Output: The ciphertext c

Choose a m such that $m \in [0, N)$

Compute $c = m^e \bmod N$

return c

Theorem 1 Assume that, m is a message from the message space $\mathcal{M} = [0, N)$, e is the public key, d is the secret key and c is the ciphertext from m that is, $c = m^e \bmod N$ with $c \in (0, N)$ then $c^d \bmod N = m$.

Proof : We first consider the case $\gcd(m, N) = 1$, then $m \in \mathcal{Z}_N^*$. We have, $c^d \bmod N \equiv (m^e)^d \bmod N \equiv m^{1+k \cdot \phi(N)} \bmod N$, for some $k \equiv m \cdot (m^{\phi(N)})^k \bmod N \equiv m \bmod N$.

Now if $\gcd(m, N) \neq 1$ then either $\gcd(m, N) = p$ or $\gcd(m, N) = q$. It suffices to prove $m \equiv c^d \bmod p$ and $m \equiv c^d \bmod q$, because then they lead to $m \equiv c^d \bmod n$ by the Chinese Remainder Theorem.

First, we prove $m \equiv c^d \bmod p$. From $c \equiv m^e \bmod n$, we know $c \equiv m^e \bmod p$, and hence, $c^d \equiv m^{ed} \bmod p$. As $ed \equiv 1 \bmod (p-1)(q-1)$, we know that $ed = t(p-1)(q-1) + 1$ for some integer t . Therefore, $m^{ed} \equiv m \cdot m^{t(p-1)(q-1)} \bmod p \equiv m \cdot (m^{p-1})^{t(q-1)} \bmod p$ (by Fermat's Little Theorem) $\equiv m \cdot (1)^{t(q-1)} \bmod p \equiv m \bmod p$. By symmetry, we also have $m^{ed} \equiv m \bmod q$. ■

Algorithm 4 “Textbook” RSA Decryption

Input: Private key (d, p, q) , the ciphertext C

Output: The message M

Compute $m = c^d \bmod N$

return m

2.3.3 Security of the RSA encryption scheme

As we stated before, “Textbook RSA” is not secure for various reasons. First of all, it is malleable as if we get ciphertext c which is the encryption of m , we can easily compute $c' = c \cdot 2^e \bmod n$ and the private key holder gets $2m \bmod n$ by decrypting c' . So making predictable changes to the ciphertexts is possible. Again, “Textbook RSA” is deterministic since distinguishing between the ciphertexts of 0 and 1 is easy by simply calculating and comparing them. Thus, it is not “semantically secure” (by which we mean that given any probabilistic polynomial time algorithm, a ciphertext of message m and message’s length, it is not possible to get any partial information about the message). So in practice, a secure RSA should use some modifications such as padding of the messages [10].

If one can factor N and find p and q , he can easily compute $\phi(N)$ by taking the product of $(p - 1)$ and $(q - 1)$. He then uses $\phi(N)$ and the public key e to find the secret key d by computing $d \equiv e^{-1} \bmod \phi(N)$ and break RSA. Again, if one has knowledge about $\phi(N)$ or, the secret key d , breaking RSA is even easier by the above computations. However, these three problems (Computing d , Factoring N and Computing $\phi(N)$) are equivalent. For example, if one can factor N , he can use p and q to find $\phi(N)$ (as above). On the other way, if $\phi(N)$ can be computed, it is easy to factor N since $\phi(N) = pq - p - q + 1 = N + 1 - (p + q)$. For given N , one solves this equation for $p + q$ and $p + q$ is used to compute $p - q$ from $p - q = \sqrt{(p + q)^2 - 4pq}$. Finally, an easy computation leads to finding the values of p and q which are the factors of N . For other equivalency proofs we refer to the book [10]. To our knowledge,

Table 2.3: Efficiency of Textbook RSA Scheme

Functions	Operations (we denote multiplication by M, pseudo random number generation by PRNG, pseudo random prime number generation by PRPNG, inverse multiplication by MI and exponentiation by E)
RSA Key Generation	$2 M + 1 \text{ PRNG} + 2 \text{ PRPNG} + 1 \text{ gcd} + 1 \text{ MI}$
RSA Encryption	$1E$
RSA Decryption	$1E$

these are the only ways of breaking RSA generally. However, solving these problems are hard in general. For example, factoring N is not efficient as the best algorithm we know off runs in sub-exponential time (e.g. N is 1024 bits, factoring N takes 2^{80} steps). But there might be other ways of breaking RSA. We can say, breaking “RSA problem” may be easier than factoring N , computing d or computing $\phi(N)$.

2.3.4 Efficiency of the RSA encryption scheme

The key generation algorithm of RSA takes 2 pseudo random generation of primes and 1 multiplication for `KeyGenModulus`. This algorithm also takes 1 multiplication, 1 inverse modular multiplication and 1 pseudo random generation of e with the gcd computation. Each of RSA encryption and decryption takes 1 modular exponentiation that can be performed in polynomial time in the input size using square and multiplication type exponentiation algorithm. The Table 2.3 shows the efficiency of the Textbook RSA scheme:

2.3.5 RSA scheme in practice

As it is true for all the public key encryption schemes, RSA encryption scheme is also extensively utilized in many practical implementations specially in hybrid encryption schemes. They are mainly utilized in “key agreement signatures” and “digital signature”. As it is regarded as comparatively slow, RSA is mostly not used in encrypting the whole file, rather it’s directly used in the encryption of the key only. After key agreement, symmetric encryption schemes are deployed for encrypting the communication.

2.4 ELGAMAL ENCRYPTION SCHEME

2.4.1 Introduction

Another popular public key encryption scheme is the ElGamal encryption scheme, named after Taher ElGamal who introduced this technique in 1985 [7]. This is an asymmetric key encryption algorithm which is based on the DiffieHellman key exchange. ElGamal encryption is implemented in the recent versions of PGP [19] and free GNU Privacy Guard software [11]. We particularly emphasize on the ElGamal encryption scheme because this scheme provides the base of the construction of the CGS encryption scheme [16] we discuss in Chapter 3.

As we did for RSA scheme, we will first represent the “Textbook” version of ElGamal encryption scheme. This version is not secure [see section 2.4.3] but provides a good idea about the ElGamal scheme used in practice which is made secure using some modifications such as padding of messages. We refer to section 2.4.3 for further details on this topic.

The ElGamal encryption can be defined over any cyclic group \mathbb{G} . Its security depends on the difficulty of a certain problem in \mathbb{G} related to computing discrete logarithms. Discrete logarithm problem in \mathbb{G} is defined as follows: Given a multiplicative

cyclic group $\mathbb{G} = \langle g \rangle$ and $g, h \in \mathbb{G}$, to find (if exists) x such that $g^x = h$. Discrete logarithm problem(DLP) is not hard in general. The hardness of DLP depends on the group \mathbb{G} .

2.4.2 Description

The following lemma provides the motivation of the ElGamal encryption scheme:

Lemma 1 *Let \mathbb{G} be a finite group, and let $m \in \mathbb{G}$ be an arbitrary element. Then choosing random $g \leftarrow \mathbb{G}$ and setting $g' := m \cdot g$ gives the same distribution for g' as choosing random $g' \leftarrow \mathbb{G}$. I.e., for any $\hat{g} \in \mathbb{G}$, $Pr[m \cdot g = \hat{g}] = 1/|\mathbb{G}|$, where the probability is taken over random choice of g .*

Proof : Let \mathbb{G} be a finite group, and $m \in \mathbb{G}$ be an arbitrary element. Now let $\hat{g} \in \mathbb{G}$ be arbitrary. Then $Pr[m \cdot g = \hat{g}] = Pr[g = m^{-1} \cdot \hat{g}]$. Since g is chosen uniformly at random, the probability that g is equal to the fixed element $m^{-1} \cdot \hat{g}$ is exactly $1/|\mathbb{G}|$.

■

This lemma gives the construction of a perfectly secret/private key encryption scheme in which the sender and the receiver takes a random element $g \leftarrow \mathbb{G}$ as their secret and to encrypt the message $m \in \mathbb{G}$, the sender computes the ciphertext $g' := m \cdot g$. The receiver can recover the message from the ciphertext g' by computing $m := g'/g$.

The decryption of this scheme is only possible because g is already shared by the sender and the receiver in advance, though it is truly random. However, in the setting for the public key encryption scheme, the receiver would be required a different method to be allowed to decrypt the ciphertext. Using a “pseudo random” element g rather than a truly random one would serve the purpose. The idea is, g will be defined in such a way that the receiver will be able to compute g using her private key, yet g will “look random” for any eavesdropper.

Let \mathcal{G} be a polynomial-time algorithm that, that takes 1^n as input, outputs a description of a cyclic group \mathbb{G} , its order q (with $|q| = n$), and a generator g . The group operation in \mathbb{G} can be computed in time polynomial in n , and the possibility of \mathcal{G} to fail with negligible probability is also allowed.

Similar to the RSA scheme, the construction of the ElGamal encryption scheme as a public key encryption scheme also has three phases: Key generation, encryption and decryption. In the key generation step, the input is the message space \mathbb{G} , an element $g \in \mathbb{G}$ and q such that q is the order of \mathbb{G} . As output, a random integer a is chosen from the group $[1, q - 1]$ as the secret key and the public key is computed as g^a . In the encryption step, the input is a message $m \in \mathbb{G}$ and as output, a ciphertext is computed which is a pair (c_1, c_2) where $c_1 = g^r$ for $r \in \mathbb{Z}_q$ chosen at random and $c_2 = (g^a)^r \cdot m$. The decryption step takes the private key a and the ciphertext (c_1, c_2) as input. The private key a is used to recover the message m by computing $m = c_2/c_1^a$. The construction of ElGamal scheme is presented in Table 2.4:

Table 2.4: Construction (The ElGamal Encryption Scheme)

Functions	Procedure
KeyGen :	Input is (\mathbb{G}, q, g) . Then choose a random $a \leftarrow \mathbb{Z}_q$ and compute g^a . The public key is $\langle \mathbb{G}, q, g, g^a \rangle$ and the private key is $\langle \mathbb{G}, q, g, a \rangle$.
Enc :	On input a public key $pk = \langle \mathbb{G}, q, g, g^a \rangle$ and a message $m \in \mathbb{G}$, choose a random $r \leftarrow \mathbb{Z}_q$ and output the ciphertext $(c_1, c_2) := (g^r, (g^a)^r \cdot m)$.
Dec :	On input a private key $sk = \langle \mathbb{G}, q, g, a \rangle$ and a ciphertext (c_1, c_2) , compute the message $m := c_2/c_1^a$.

The following theorem proves the correctness of the ElGamal encryption scheme:

Theorem 2 *If for the message $m \in \mathbb{G}$, the ciphertext is $c = (c_1, c_2)$ such that*

Algorithm 5 ElGamal Key Generation

Input: Security parameter 1^n , group \mathbb{G} and element $g \in \mathbb{G}$

Output: Private key $sk = a$, public key $pk = g^a$

Choose a random $a \leftarrow [1, q - 1]$

Compute g^a

return a, g^a

Algorithm 6 ElGamal Encryption

Input: A public key, $pk = \langle \mathbb{G}, q, g, g^a \rangle$ and a message $m \in \mathbb{G}$

Output: Ciphertext $(c_1, c_2) := (g^r, (g^a)^r \cdot m)$

Choose a random $r \leftarrow [1, q]$

Compute $c_1 = g^r$

Compute $(g^a)^r$

Compute $c_2 = (g^a)^r \cdot m$

return $g^r, (g^a)^r \cdot m$

$(c_1, c_2) := (g^r, (g^a)^r \cdot m)$, where g, a, r are as in Table 2.4. Then, $c_2/c_1^a = m$.

Proof : The decryption of ElGamal works as follows:

We have $C = (c_1, c_2) = (g^r, (g^a)^r \cdot m)$, then $c_2/c_1^a = (g^a)^r \cdot m / (g^r)^a = (g^a)^r \cdot m / (g^a)^r = m$.

■

2.4.3 Security of the ElGamal encryption scheme

The security of ElGamal encryption scheme depends on the difficulty of the computational Diffie-Hellman(CDH) problem. Computational Diffie-Hellman(CDH) problem is a problem that if we pick a random generator g from a cyclic group G of order q ; draw g^r and g^s randomly from that group where $r, s \in \{0, 1, \dots, q - 1\}$, it is very hard for the challenger to compute g^{rs} without having any knowledge about r and s . The following theorem proves that solving CDH and decryption of ElGamal are

Algorithm 7 ElGamal Decryption

Input: A private key $sk = \langle \mathbb{G}, q, g, a \rangle$ and a ciphertext (c_1, c_2)

Output: Message m

 Compute c_1^a

 Compute c_2/c_1^a

return m

equivalent.

Theorem 3 *An algorithm that solves computational Diffie-Hellman(CDH) problem can be converted (in polynomial time) to another algorithm that decrypts ElGamal ciphertexts. Similarly any algorithm that can decrypt ElGamal ciphertexts can be converted (in polynomial time) to an algorithm that solves CDH.*

Proof : We first prove that if CDH problem can be solved then it is easy to decrypt the ElGamal problem. To do this, let's first suppose there exists an algorithm that solves CDH problem. Suppose that the instance to the decryption of the ElGamal problem is (g, c_1, c_2) where (c_1, c_2) is ciphertext and $\mathbb{G} = \langle g \rangle$, $|\mathbb{G}| = q$, q is prime. Then we can query the CDH solver with the input (g, g^a, c_1) where g^a is the public key, $c_1 = g^b$ for some b and receive output c_1^a by $g^{ab} = (g^a)^b$. Then we output (as the output of decryption of ElGamal) c_2/c_1^a .

Now we prove the other direction, i.e. if decryption of ElGamal encrypted message is possible, then it is also possible to solve CDH problem. Let's suppose now that there is an algorithm that decrypts the ElGamal problem. Suppose that the instance to the CDH problem is g, g^a, g^b . We create an instance to the decryption of ElGamal algorithm as follows: Public key:= g^a , (generator:= g , $c_1 := g^b$ and $c_2 := c \in \mathbb{G}$ chosen at random). Then the decryption of ElGamal scheme will output $x \in \mathbb{G}$ and we will output (as the output of CDH solver) $c_2 \cdot x^{-1} = g^{ab}$ because $x \cdot c_1^a = c_2$ implies $x \cdot g^{ab} = c_2$ by the definition and by the correctness of the ElGamal decryption. ■

The theorem shows that if solving CDH is infeasible, then decrypting ElGamal is also infeasible i.e. ElGamal encryption is one-way. It does not say much about the semantic security of the ElGamal encryption. In the paper [12], it is shown that the semantic security is not implied by the computational DiffieHellman assumption alone. The semantic security of the ElGamal encryption scheme requires the intractability of the decisional Diffie-Hellman(DDH) problem which is the problem of distinguishing g^{ab} from a random element in \mathbb{G} ; given g, g^a and g^b for some a and b . We summarize this in the following theorem.

Theorem 4 [10] *If the DDH problem is hard relative to g , then the ElGamal encryption scheme has indistinguishable encryptions under a chosen plaintext attack. (By indistinguishable encryptions, we mean that given encryption of two different messages, it is indistinguishable to find out which message the ciphertext is from).*

As mentioned in section 2.4.1, the “Textbook” version of ElGamal encryption scheme is completely insecure under chosen ciphertext attack. For example, given an encryption (c_1, c_2) of some unknown (possibly) message m , it is very easy to construct a valid encryption (c_1, c_2) of the message $2m$. The scheme must be changed or, further modified, or an appropriate padding scheme must be implemented to achieve chosen-ciphertext security. The necessity of the decisional Diffie-Hellman(DDH) assumption [1] depends on this modification of the scheme.

The factors that the security of the ElGamal scheme depends on are the properties of the underlying group \mathbb{G} as well as any padding scheme used on the messages.

If the computational Diffie-Hellman(CDH) assumption holds in the underlying cyclic group \mathbb{G} , then the encryption function is one-way [5]. If the decisional Diffie-Hellman(DDH) assumption [1] holds in \mathbb{G} , then ElGamal achieves semantic security [5]. For the scheme to be specified fully, showing the process of encoding binary strings as elements of \mathbb{G} is also required.

Table 2.5: Efficiency of ElGamal Encryption Scheme

Functions	Operations (we denote multiplication by M, pseudo random number generation by PRNG, scalar multiplication by SM, division by D, subtraction by S, addition by A and exponentiation by E)
ElGamal Key Generation	1PRNG + 1E
ElGamal Encryption	1PRNG + 2E + 1M
ElGamal Decryption	1E + 1D

Some other schemes related to ElGamal which are secure against chosen ciphertext attacks have also been proposed. Based on the assumption that DDH holds for G , the CramerShoup cryptosystem is regarded to be secure under chosen ciphertext attack. Its proof does not use the random oracle model. Another proposed scheme is DHAES [12], but this scheme is proved assuming that it is weaker than the DDH assumption.

2.4.4 Efficiency of the ElGamal encryption scheme

In general, ElGamal encryption scheme has a size expansion of 2:1 ratio from plaintext to ciphertext since it is a probabilistic scheme. By a probabilistic scheme, we mean that there are possibilities of construction of many ciphertexts from a single plaintext. The encryption function under ElGamal requires two exponentiations and a single multiplication; however, none of these exponentiations are dependent on the message and can be computed ahead of time if needed to be. The decryption function under ElGamal requires only one exponentiation and a single division. The table 2.5 shows the efficiency of this scheme:

2.4.5 ElGamal encryption scheme in practice

ElGamal cryptosystem is an asymmetric cryptosystem which is usually slower as compared to the symmetric cryptosystems. This case is in general true for all the asymmetric cryptosystem considering the same level of security. Symmetric key is mostly quite small in size comparing with the message size. So using ElGamal scheme to encrypt the symmetric key and other symmetric crypto-schemes to encrypt the message, which can be arbitrarily large, is a relatively a faster way of encryption. This technique of implementing both asymmetric and symmetric cryptosystem in encryption is known as *hybrid cryptosystem*. Another significant use of ElGamal encryption is with \mathbb{G} being an elliptic curve group and have the advantage of enabling the use of much shorter keys.

2.5 REMARKS

The public key encryption concept was introduced first in the open literature by Diffie and Hellman [6]. Later on, they proposed Diffie-Hellman key exchange protocol in 1976. The ElGamal encryption [7] scheme which is regarded as the direct transformation of this Diffie-Hellman key exchange protocol, yet was not proposed until 1984. The RSA assumption was introduced by Rivest, Shamir and Adleman [14] and based on this assumption, they proposed the well-known RSA public key encryption scheme. In the next chapter, we discuss another public key cryptographic scheme known as CGS Scheme [16]. This scheme is a linearly homomorphic encryption scheme which is based on the ElGamal cryptosystem. We also describe linearly homomorphic functions of this scheme.

CHAPTER 3

CGS ENCRYPTION SCHEME

3.1 INTRODUCTION

While electronic voting protocols are some key applications of the secure multiparty computations in the cryptographic literature, these protocols still require refinements in terms of privacy, verifiability and robustness. To address these issues, R. Cramer, R. Gennaro and Berry Schoenmakers jointly introduced a new multi-authority secret ballot election scheme known as CGS scheme [16]. This scheme basically uses the ElGamal encryption scheme discussed in the last chapter, but replaces the message m by G^m where $G \in G_q$, G public, and q is a prime. The advantages of this scheme over the other proposed techniques is that CGS encryption scheme provides computer privacy protection even though the voter's effort is independent of the number of authorities.

A *homomorphic encryption* scheme is an encryption mechanism where certain computations (addition and multiplication) can be performed on the encrypted plaintexts, generating a ciphertext such that when it is decrypted, this is the same result from the similar operations performed on the plaintexts. It is a very useful feature in modern communication systems and cryptography as this scheme allows combining or comparing of messages without revealing the messages. For simplicity, we assume that our message space $(\mathcal{M}, +, \cdot)$ is the set of integers where $+$ and \cdot are usual addition and multiplication operations on it. A homomorphic encryption is said to be *additive* if the encryption of the summation of the messages can be computed as a function of the encryption of the messages. This function is regarded as the ad-

dition operation on the ciphertexts. Mathematically, we denote it by \boxplus . That is, for messages $m_1, m_2 \in \mathcal{M}$, this gives $\text{Enc}(m_1 + m_2) = \text{Enc}(m_1) \boxplus \text{Enc}(m_2)$. Here, Enc stands for the encryption function and \boxplus is the addition operation in ciphertext space C . Similarly, a homomorphic encryption is said to be *multiplicative* if the encryption of the multiplication of the messages can be computed as a function of the encryption of the messages. This function is regarded as the multiplication operation on the ciphertexts which mathematically we denote by \boxtimes . That is, for messages $m_1, m_2 \in \mathcal{M}$, this gives $\text{Enc}(m_1 \cdot m_2) = \text{Enc}(m_1) \boxtimes \text{Enc}(m_2)$. Here, Enc stands for the encryption function and \boxtimes is the multiplication operation in ciphertext space C . *Scalar multiplication* function can be derived from the additive homomorphic encryption as for message $m \in \mathcal{M}$ and for some scalar s we can see, $\text{Enc}(s \cdot m) = \text{Enc}(m + m + \dots + m) = \text{Enc}(m) \boxplus \text{Enc}(m) \boxplus \dots \boxplus \text{Enc}(m) = s \boxtimes \text{Enc}(m)$.

CGS encryption scheme is a linearly homomorphic encryption scheme. That is, this scheme allows homomorphic addition and scalar multiplication operations, thus linear combination properties on the ciphertexts. For these homomorphic properties, CGS encryption scheme turned out to be very useful in the context of cryptography [16]. However, CGS scheme does not allow homomorphic multiplication on encrypted messages. In the next chapter, we discuss about a well-known boosting technique on linearly homomorphic schemes, which we apply on the CGS scheme in the later chapter so that homomorphic multiplication operation can be performed on ciphertexts.

In this chapter, we focus our discussion on the CGS encryption scheme [16] and the evaluation functions for its homomorphic properties.

3.2 DESCRIPTION

As mentioned earlier in this chapter, Cramer, Genarro and Schoenmakers presented CGS-97 encryption scheme as a variant on the ElGamal [7] scheme that takes a public key $G \in G_q$ as input, computes G^m and use it instead of the message m in the

encryption of the ElGamal scheme.

The construction of the CGS scheme consists of four faces: key generation, encryption, evaluation functions and decryption. The CGS scheme randomly chooses two large n -bit public primes p and q such that $q|(p-1)$ where n is the security parameter for the key generation function. The message or plaintext space for this scheme is \mathbb{Z}_p and the ciphertext space is $\tilde{C} = \mathbb{Z}_p \times \mathbb{Z}_p$. A random number a from \mathbb{Z}_q^* provides as the sender Alice's secret key and her public key is $G = g^a \mod p$. In the encryption step, the receiver Bob chooses a random number r from \mathbb{Z}_q^* and encrypts the message m to get the ciphertext $c = [x, y] = (g^r \mod p, G^{r+m} \mod p) \in \tilde{C}$. Evaluation functions step includes the linearly homomorphic properties of CGS scheme: addition, scalar multiplication and thus linear combination. In the decryption step, Alice, to decrypt the ciphertext into the original message, finds $G^m = \frac{y}{x^a} \mod p$ and then computes m by solving discrete logarithm problem of $(g^a, (g^a)^m)$. By solving discrete logarithmic problem, we mean that it tries all possible $m' \in \mathcal{M} = [-B, B]$ until it satisfies $G^{m'} = (g^a)^m$, here B is the number that determines the size of the message space. The decryption scheme requires an exhaustive search in the plaintext space which is only possible if the message space is small. Note that, decryption function (to be discussed later) is a linear function of B . As B gets larger, decryption slows down. So, the number B that is the size of the message space needs to be reasonable so that the runtime is efficient as well as the message space is not too small. In our implementation of this scheme, we used $B=10,000$. [see MAGMA code for $\text{KeyGen}_{\text{CGS}}(n)$]. We denote the addition and multiplication operation on CGS ciphertexts by \boxplus and \boxtimes respectively.

We now discuss the partially linearly homomorphic scheme [16] CGS, represented by the notation $\text{CGS} = (\text{KeyGen}_{\text{CGS}}, \text{Enc}_{\text{CGS}}, \text{Dec}_{\text{CGS}}, \text{Eval}_{\text{CGS}})$.

3.2.1 Key generation function for the CGS scheme ($\text{KeyGen}_{\text{CGS}}$)

The key generation function of CGS scheme $\text{KeyGen}_{\text{CGS}}$ has a security parameter n as input (the security parameter n determines the size of the primes and keys) and output of this function is (p, q, g, a, G) . The secret key sk of the scheme is a random number a in \mathbb{Z}_q^* . The parameters g , p and q are the domain(system) parameters which are common in all the functions. The public key of the scheme pk is G . The algorithm 8 refers to the key generation function for the CGS scheme.

Algorithm 8 CGS Key Generation

Input: Security parameter 1^n , group \mathbb{G} and element $g \in \mathbb{G}$

Output: Private key $sk = a$, public key $pk = g^a$

Choose a random $a \leftarrow [1, q - 1]$

Compute g^a

return a, g^a

MAGMA source code ($\text{Keygen}_{\text{CGS}}$): We state the MAGMA code we used for the key generation function for CGS scheme.

```
SetSeed(1);

KeyGen_CGS:= function(n)
    while true do
        i:=Random(2^(n-1),2^n);
        if IsPrime(i: Proof:=false) then
            q:=i;    //q is a randomly picked n-bit prime;
            if IsPrime(2*q+1: Proof:=false) then
                p:=2*q+1;    // p is a prime
                a:=Random(1,q-1); // a is a randomly picked number in Zq^*
                break;
            end if;
        end if;
    end while;
end function;
```

```

        end if;
    end if;
end while;

Zp:=GF(p);
// g is the generator of the subgroup  $G_q \leq Z_p^*$ ;
//if  $g^a \neq 1$ , g is replaced by  $g^2$ 
for i in Zp do
    if i  $\neq$  0 then
        if  $i^2 \neq 1$  then
            if i  $\neq$  1 then
                if  $i^a \neq 1$  then //a not equal to 0
                     $g:=i^2$ ;
                     $G:=g^a$ ;
                    break i;
                end if;
            end if;
        end if;
    end if;
end for;

return p,q,g,a,G;
end function;

```

3.2.2 Encryption function for the CGS scheme (Enc_{CGS})

The encryption function of the CGS scheme Enc_{CGS} takes a pair (pk, m) as input where $pk = G$ is the public key and the message $m \in \mathcal{M}$, \mathcal{M} is the message space $[-B, B]$ where B is some number that determines the size of the message space. The

domain(system) parameters are g and q which are common parameters in all the functions. Output of this function is a cipher text $c \in \tilde{C}$; where \tilde{C} is a cipher text space (i.e. CGS cipher text space). The ciphertext c is a sequence of two group elements i.e. $c = [x, y]$. Algorithm 9 gives the encryption function for the CGS scheme.

Algorithm 9 CGS Encryption

Input: Public key G , message m

Output: Ciphertext $c = [x, y]$

Choose a random number $r \in_R [1, q - 1]$

if r is prime **then**

 Compute $x := g^r$

 Compute $y := G^r * G^m$

end if

return $[x, y]$

MAGMA source code (Enc_{CGS}): Following is the MAGMA code for the CGS encryption function. The message space we considered is the interval $[-B, B]$.

```
Enc_CGS:= function(m,g,q,G)
    r:=Random(1,q);    //r is a randomly picked number.
    x:=g^r;
    y:=(G^r)*(G^m);
    return [x,y];
end function;
```

3.2.3 Evaluation functions for the CGS scheme (Eval_{CGS})

In this section, we focus on different evaluation functions Eval_{CGS} using CGS cryptosystem. It allows addition and scalar multiplication of messages unbounded times,

so the CGS scheme efficiently performs on the linear combination function of encrypted messages. We show the evaluation algorithm for different functions with their correctness in the following:

Addition function in CGS: The CGS encryption scheme allows the addition of two ciphertexts and returns the encryption of the summation of the plaintexts. So, input to this function is a pair (c_1, c_2) where $c_1, c_2 \in \tilde{C}$; \tilde{C} is the cipher text space (i.e. CGS cipher text space). The ciphertexts are pairs $c_1 = \text{Enc}_{\text{CGS}}(m_1) = [x_1, y_1]$, $c_2 = \text{Enc}_{\text{CGS}}(m_2) = [x_2, y_2]$. The common domain(system) parameters are g, p, G . Output of this function is a cipher text $c_1 \boxplus c_2$ where $c_1 \boxplus c_2 \in \tilde{C}$; \tilde{C} is the CGS cipher text space. The ciphertext $c_1 \boxplus c_2$ is a sequence of two group elements, we write $c = [x, y] = \text{Enc}_{\text{CGS}}(m_1 + m_2)$. Algorithm 10 shows the addition function Add_{CGS} of CGS scheme.

Algorithm 10 CGS Addition

Input: The ciphertext pair $c_1 = \text{Enc}_{\text{CGS}}(m_1) = [x_1, y_1]$ and $c_2 = \text{Enc}_{\text{CGS}}(m_2) = [x_2, y_2]$

Output: The ciphertext $c = c_1 \boxplus c_2 = [x, y] = \text{Enc}_{\text{CGS}}(m_1 + m_2)$

 Compute $x := x_1 \cdot x_2$

 Compute $y := y_1 \cdot y_2$

return $[x, y]$

The following theorem proves the correctness of addition function of CGS scheme.

Theorem 5 *Assume that m_1 and m_2 are two messages from the message space \mathcal{M} . If $c_1 = \text{Enc}_{\text{CGS}}(m_1)$, $c_2 = \text{Enc}_{\text{CGS}}(m_2)$ and $c = [x, y]$ is the output of Algorithm 10, then $c = \text{Enc}_{\text{CGS}}(m_1 + m_2)$.*

Proof : We start with the summation of the encrypted messages and show to the

other end. This proof works both ways.

$$c = \text{Enc}_{\text{CGS}}(m_1) \boxplus \text{Enc}_{\text{CGS}}(m_2) = [x_1, y_1] \boxplus [x_2, y_2] \quad (3.1)$$

We use CGS homomorphic addition (Algorithm 10) on the right hand side (r.h.s.) to get $[x_1.x_2, y_1.y_2]$ and follow Algorithm 9 to substitute values of x_i 's and y_i 's, so the r.h.s. of (3.1) becomes,

$$[g^{r_1}.g^{r_2}, G^{r_1}.G^{m_1}.G^{r_2}.G^{m_2}] \quad (3.2)$$

computations follows to

$$[g^{r_1}.g^{r_2}, G^{r_1}.G^{r_2}.G^{m_1}.G^{m_2}]$$

then

$$[g^{r_1+r_2}, G^{r_1+r_2}.G^{m_1+m_2}]$$

and

$$[g^{r'}, G^{r'} \cdot G^{m_1+m_2}] \quad (3.3)$$

where $r' = r_1 + r_2$, which finally yields,

$$\text{Enc}_{\text{CGS}}(m_1 + m_2)$$

■

MAGMA source code (Add_{CGS}): We represent below the MAGMA code for the addition function of the CGS scheme:

```
Add_CGS:= function(C1,C2,g,p,G);
  x1:=C1[1];y1:=C1[2];
  x2:=C2[1];y2:=C2[2];
  x:=x1*x2;
  y:=y1*y2;
  return [x,y];
end function;
```

Scalar multiplication in CGS: Input to this function is a pair (s, c_1) where s is a scalar from the message space \mathcal{M} . The ciphertext $c_1 \in \tilde{C}$ where \tilde{C} is a cipher text space (i.e. CGS cipher text space) and $c_1 = \text{Enc}_{\text{CGS}}(m_1) = [x_1, y_1]$. The common domain(system) parameters are g, p and G . Output of this function is a cipher text $s \boxtimes c_1$ where $s \boxtimes c_1 \in \tilde{C}$; \tilde{C} is the CGS cipher text space and $s \boxtimes c_1$ is a sequence of two group elements $s \boxtimes c_1 = [x, y] = \text{Enc}_{\text{CGS}}(s \cdot m_1)$.

Algorithm 11 CGS Scalar Multiplication

Input: The ciphertext $c_1 = [x_1, y_1]$ and a scalar $s \in \mathcal{M}$

Output: The ciphertext $s \boxtimes c_1 = [x, y] = \text{Enc}_{\text{CGS}}(s \cdot m_1)$

Compute $x := x_1^s$

Compute $y := y_1^s$

return $[x, y]$

The following theorem proves the correctness of scalar multiplication function of CGS scheme.

Theorem 6 Assume that m_1 is a message from the message space \mathcal{M} and s is any scalar in \mathcal{M} . If $c_1 = \text{Enc}_{\text{CGS}}(m_1)$ and $c = [x, y]$ is the output of Algorithm 11, then $c = \text{Enc}_{\text{CGS}}(s \cdot m_1)$.

Proof : We start with the scalar multiplication of the encrypted message and show that this is equal to the encryption of the scalar multiplication of the message. We use algorithm 11 for CGS scalar multiplication on ciphertext.

$$c = s \boxtimes c_1 = s \boxtimes \text{Enc}_{\text{CGS}}(m_1) = s \boxtimes [x_1, y_1] = [x_1^s, y_1^s] \quad (3.4)$$

Substitute the values of x_i 's and y_i 's following Algorithm 9 and get right hand side of (3.4) equal to

$$[(g^{r_1})^s, (G^{r_1} \cdot G^{m_1})^s] \quad (3.5)$$

then

$$[g^{r_1 s}, G^{r_1 s} . G^{m_1 s}] \quad (3.6)$$

we can write,

$$[g^{r'}, G^{r'} . G^{sm_1}]$$

where $r' = r_1 s$ which finally yields,

$$\text{Enc}_{\text{CGS}}(s.m_1)$$

■

MAGMA source code (SMult_{CGS}): Below is the MAGMA code we implemented for the scalar multiplication function of the CGS scheme:

```
Smult_CGS:= function(s,C1,g,p,G);
  x1:=C1[1];y1:=C1[2];
  x:=x1^s;
  y:=y1^s;
  return [x,y];
end function;
```

Linear combination function (addition and scalar multiplication) in CGS: The addition and scalar multiplication functions over unbounded number of encrypted messages allows the linear combination of the ciphertexts. Input to this function is a pair of sets (s, c) where $s = [s_1, s_2, \dots]$; s_i 's are a scalars in the message space \mathcal{M} and $c = [c_1, c_2, \dots]$ where c_i 's are ciphertexts and each $c_i \in \tilde{C}$; the set \tilde{C} is a cipher text space (i.e. CGS cipher text space). The ciphertext $c_i = \text{Enc}_{\text{CGS}}(m_i) = [x_i, y_i]$. The parameters g, p, G are the domain (system) parameters, common in all the functions of CGS. Output of this function is a ciphertext $c = (s_1 \boxdot c_1) \boxplus (s_2 \boxdot c_2) \boxplus \dots \boxplus (s_k \boxdot c_k)$

where $c \in \tilde{C}$; \tilde{C} is a cipher text space (i.e. CGS cipher text space). The ciphertext $c = \text{Enc}_{\text{CGS}}(\sum_i s_i \cdot m_i)$ is a sequence of two group elements, $c = [x, y]$.

Algorithm 12 CGS Linear Combination function

Input: A a pair of sets (s, c) where $s = [s_1, s_2, \dots]$ and $c = [c_1, c_2, \dots]$ where each

$$c_i = [x_i, y_i]$$

Output: The ciphertext $c = [x, y] = \text{Enc}_{\text{CGS}}(\sum_i s_i \cdot m_i)$

Define $k := \#s$

Choose $x := 1$

Choose $y := 1$

for $i=1$ to k **do**

 Compute $x := x.x_i^{s_i}$

 Compute $y := y.y_i^{s_i}$

end for

return $[x, y]$

MAGMA source code (LinComb_{CGS}): We state the MAGMA code for the linear combination function for the CGS scheme below.

```
LinComb_CGS:= function(S,C,g,p,G);
  k:=#S;
  x:=1;
  y:=1;
  for i:=1 to k do
    x[i]:=C[i][1];  y[i]:=C[i][2];
    x:=x*(x[i])^S[i];
    y:=y*(y[i])^S[i];
  end for;
  return [x,y];
```


end function;

The following theorem proves the correctness of linear combination function function of the CGS scheme.

Theorem 7 *Assume that m_1, m_2, \dots, m_k are messages from the message space \mathcal{M} and s_1, s_2, \dots, s_k are scalars in \mathcal{M} . If $c_i = \text{Enc}_{\text{CGS}}(m_i)$ for all i such that $1 \leq i \leq k$ and $c = [x, y]$ is the output of Algorithm 12, then $c = \text{Enc}_{\text{CGS}}(\sum_i s_i \cdot m_i)$.*

Proof : We start with the linear combination of the encrypted messages and show that this is equal to the to the encryption of the linear combination of the message. This proof is true in both ways.

$$\begin{aligned} c = [x, y] &= (s_1 \boxdot c_1) \boxplus (s_2 \boxdot c_2) \boxplus \dots \boxplus (s_k \boxdot c_k) \\ &= (s_1 \boxdot \text{Enc}_{\text{CGS}}(m_1)) \boxplus (s_2 \boxdot \text{Enc}_{\text{CGS}}(m_2)) \boxplus \dots \boxplus (s_k \boxdot \text{Enc}_{\text{CGS}}(m_k)) \\ &= (s_1 \boxdot [x_1, y_1]) \boxplus (s_2 \boxdot [x_2, y_2]) \boxplus \dots \boxplus (s_k \boxdot [x_k, y_k]) \end{aligned} \quad (3.7)$$

Use algorithm 11 to get the right hand side of this equation as,

$$[x_1^{s_1}, y_1^{s_1}] \boxplus [x_2^{s_2}, y_2^{s_2}] \boxplus \dots \boxplus [x_k^{s_k}, y_k^{s_k}] \quad (3.8)$$

Substitute the values of x_i 's and y_i 's following algorithm 9 and using linear combination from algorithm 12, we get right hand side of (3.8) equal to

$$[g^{r_1 s_1} + g^{r_2 s_2} + \dots + g^{r_k s_k}, (G^{r_1} \cdot G^{m_1})^{s_1} + (G^{r_2} \cdot G^{m_2})^{s_2} + \dots + (G^{r_k} \cdot G^{m_k})^{s_k}]$$

then,

$$[g^{r_1 s_1} + g^{r_2 s_2} + \dots + g^{r_k s_k}, G^{r_1 s_1} \cdot G^{m_1 s_1} + G^{r_2 s_2} \cdot G^{m_2 s_2} + \dots + G^{r_k s_k} \cdot G^{m_k s_k}] \quad (3.9)$$

Computations follow to,

$$[g^{r_1 s_1 + r_2 s_2 + \dots + r_k s_k}, G^{r_1 s_1 + r_2 s_2 + \dots + r_k s_k} \cdot G^{m_1 s_1 + m_2 s_2 + \dots + m_k s_k}]$$

and then

$$[g^{r'}, G^{r'} \cdot G^{m_1 s_1 + m_2 s_2 + \dots + m_k s_k}]$$

where $r' = r_1 s_1 + r_2 s_2 + \dots + r_k s_k$, which is equal to,

$$\text{Enc}_{\text{CGS}}(s_1.m_1 + s_2.m_2 + \dots + s_k.m_k).$$

■

3.2.4 Decryption function for the CGS scheme (Dec_{CGS})

Input to decryption function of the CGS scheme Dec_{CGS} is a pair (sk, c) where sk is the secret key and $c \in \tilde{C}$ is the ciphertext; \tilde{C} is a cipher text space (i.e. CGS cipher text space). c is a sequence of two group elements $[x, y]$ and $x, y \in Z_p$. The secret key sk is a . The parameter p is the domain (system) parameter, common in all the functions and the public key pk is G . Output of this function is the message m . Algorithm 13 shows the decryption function of the CGS scheme.

Algorithm 13 CGS Decryption

Input: Secret key a , the ciphertext $c = [x, y]$

Output: Message m

 Compute $k_1 := x^a$

 Compute $k_2 := y/k_1$

for $i \in [-B, B]$

if $G^i == k_2$ **then**

$m = i$

return m

end if

end for

return "error"

MAGMA source code (Dec_{CGS}): We now state the MAGMA code for the decryption function of the CGS scheme.

```

B:=10000;
Dec_CGS:=function(C,G,a,p)
  Zp:=GF(p);
  x:=C[1];    // x in Z_p
  y:=C[2];    // y in Z_p
  K1:=(Zp!x)^a;
  K2:=(Zp!y)/K1;
  // Check if for any i in M, G^i=K2 then i=m
  for i in [-B..B] do
    // printf "i= %o\n",i;
    if G^i eq K2 then
      return i;
    end if;
  end for;
  return "error";
end function;

```

3.3 SECURITY OF THE CGS ENCRYPTION SCHEME

For the security of the CGS encryption scheme, it is required that discrete logarithm problem is intractable [16]. Since the construction of the CGS scheme depends on the ElGamal scheme. Computational Diffie-Hellman problem has to be intractable to ensure the security of the CGS scheme since solving computational Diffie-Hellman problem and decryption of ElGamal scheme are equivalent problems (we proved in section 2.4.3). With a similar argument, for the semantic security of CGS scheme, decisional Diffie-Hellman problem has to be intractable. However, to our knowledge

it's still not known whether the security of ElGamal and CGS schemes are equivalent or not. For further security notions, we refer to section 6 of the paper [16].

3.4 EFFICIENCY OF THE CGS ENCRYPTION SCHEME

As the invention of the CGS scheme was aimed for improving the performance of electric voting techniques [16], it came out to be an efficient scheme for the purpose as compared to the other mix-based scheme available at that time [16]. A stronger sense of robustness is gained due to the threshold cryptosystem. We refer to the paper [16] for further information. The key generation of the CGS scheme requires 1 pseudo random number generation and 1 exponentiation. Encryption takes 1 pseudo random prime number generation, 3 exponentiation and 1 multiplication. The decryption step needs 2 exponentiations with 1 division. One CGS addition needs 2 multiplications while CGS scalar multiplication requires 2 exponentiations. Finally, the general linear combination function requires $2k$ exponentiation and $2k$ multiplications where k is the number of scalars. Table 3.1 shows the efficiency of different functions on CGS scheme.

3.5 REMARKS

CGS encryption system is very efficient as a linearly homomorphic encryption scheme as it allows unbounded number of additions with scalar multiplications. However, it fails for the computations where it requires multiplication of plaintext inputs. In the next chapter, we discuss a new encryption scheme that boosts the linearly homomorphic encryption schemes to evaluate degree-2 functions on encrypted data [4].

Table 3.1: Efficiency of CGS Encryption Scheme

Functions	Operations (we denote multiplication by M, pseudo random number generation by PRNG, pseudo random prime number generation by PRPNG, scalar multiplication by SM, division by D, subtraction by S, addition by A and exponentiation by E)
CGS Key generation	1 PRNG + 1E
CGS Encryption	1 PRPNG + 3E + 1M
CGS Decryption	2E + 1D
CGS Addition	2M
CGS Scalar Multiplication	2E
CGS Linear Combination ($k = \#s$)	2k E + 2k M

CHAPTER 4

BOOSTING LINEARLY HOMOMORPHIC ENCRYPTION SCHEME

4.1 INTRODUCTION

Although linearly homomorphic schemes are very useful, these have limitations of being capable of evaluating computations involving 2nd degree polynomials on ciphertexts. To resolve this issue, several researchers came up with different schemes that would allow higher degree computations on ciphertexts. Dario Catalano and Dario Fiore introduced a technique in their paper [4] to boost a linearly homomorphic encryption scheme to enable degree 2 polynomial computations on encrypted messages provided that the message space is a public ring so that sample elements can be picked uniformly at random. This simple but effective boosting scheme surprisingly works on virtually all the existing number theoretic linearly-homomorphic schemes such as Paillier, ElGamal [7] or Goldwasser-Micali [9]. This scheme satisfies leveled circuit privacy and can evaluate polynomials in class $\mathcal{F}_\epsilon^* = f(m) \subset F_2$ where F_d =class of (multivariate) polynomials of total degree d over the ring \mathcal{M} . In this chapter, we discuss this boosting linearly homomorphic scheme by Catalano [4] which later on would be implemented to the CGS scheme (we discussed in chapter 3) in the next chapter.

4.2 DESCRIPTION

The construction of the boosted linearly homomorphic scheme consists of four faces: key generation, encryption, evaluation functions and decryption. This methodology is capable of converting a public-space linearly-homomorphic encryption scheme $\widehat{\text{HE}} =$

$(\widehat{\text{KeyGen}}, \widehat{\text{Enc}}, \widehat{\text{Eval}}, \widehat{\text{Dec}})$ with message space $\mathcal{M}(\widehat{HE})$, and ciphertext space $\widehat{\mathcal{C}}(\widehat{HE})$ to a homomorphic encryption scheme supporting one multiplication, we denote this scheme by $HE_B = (\text{KeyGen}_B, \text{Enc}_B, \text{Eval}_B, \text{Dec}_B)$ with message space $\mathcal{M}(HE_B)$ and ciphertext space $\mathcal{C}(HE_B)$. That is, the converted scheme would be able to evaluate arithmetic circuits of degree 2 over the message space \mathcal{M} . More generally, HE_B can compactly evaluate arithmetic circuits in which the number of additions of degree-2 terms is bounded by some constant whereas the number of additions of degree-1 terms is unbounded. We denote homomorphic addition operation by \boxplus and scalar multiplication operation by \boxtimes in the ciphertext space $\widehat{\mathcal{C}}(\widehat{HE})$ of underlying LHE scheme. The message spaces are the same for both the homomorphic encryption schemes (non-boosted and boosted), that is, $\mathcal{M}(\widehat{HE}) = \mathcal{M}(HE_B)$. The addition operation and the multiplication operation in the plaintext/message space $\mathcal{M}(\widehat{HE})$ or $\mathcal{M}(HE_B)$ are denoted by $+$ and \cdot respectively.

4.2.1 Key generation function for the Boosted-LHE scheme (KeyGen_B)

The key generation function for a boosted linearly homomorphic scheme is same as the key generation of any linearly homomorphic encryption scheme i.e. $\widehat{\text{KeyGen}} = \text{KeyGen}_B$. Input to this function is 1^n where n is the security parameter that determines the size of the message space. Output of this function is a pair (pk, sk) where pk =public key $\mathbb{G} = \langle g \rangle$ and sk = secret key a . The public key implicitly contains description of $\mathcal{M}(\widehat{HE})$ and $\widehat{\mathcal{C}}(\widehat{HE})$. The domain (system) parameter that is common in all functions is g . Message spaces are same for both the encryption scheme, i. e. $\mathcal{M}(\widehat{HE}) = \mathcal{M}(HE_B)$.

4.2.2 Encryption function for the Boosted-LHE scheme (Enc_B)

The encryption function for the boosted linearly homomorphic scheme has a pair (pk, m) as input where pk is the public key (g, G) and m is the message to be en-

encrypted. Output of this function is a cipher text c where $c \in \mathcal{M} \times \widehat{\mathcal{C}}$ where \mathcal{M} is a message space and $\widehat{\mathcal{C}}$ is a cipher text space (i.e. LHE cipher text space). So, a cipher-text is a sequence of two elements u and β where $u \in \mathcal{M}$ and $\beta \in \widehat{\mathcal{C}}$. i.e. $c = [u, \beta]$. The Algorithm 14 gives the encryption function of the Boosted-LHE scheme.

Algorithm 14 Boosted-LHE Encryption (Enc_B)

Input: Public key $pk = G$, message m

Output: Ciphertext $c = [u, \beta]$

Choose a random number $b \in_R \mathcal{M}$

Compute $u = m - b$

Compute $\beta = \widehat{\text{Enc}}(b)$

return u, β

4.2.3 Evaluation functions for the Boosted LHE scheme (Eval_B)

The evaluation function algorithms of this scheme implement the homomorphic operations, that is, addition, multiplication and multiplication by known constants. Ciphertexts are of two levels: level 1 and level 2. Level 1 ciphertexts are ciphertexts that encode either “fresh” messages or messages that are the linear combinations of “fresh” messages whereas level 2 ciphertexts are ciphertexts that are “multiplied” level 1 ciphertexts. The evaluation function algorithms are described in five different procedures we denote by, Add_1 , Mult , Add_2 , SMult_1 , SMult_2 where Add_1 and Mult are the addition and multiplication operation between two level 1 ciphertexts. Add_2 operates over a pair of level 2 ciphertexts, that is, addition operation between two level 2 ciphertexts. SMult_1 is the scalar multiplication that operates over a single level 1 ciphertext and SMult_2 is the scalar multiplication that operates over a single level 2 ciphertext.

Addition in the Boosted LHE, level 1 (Add₁): This is a function that maps an element from $C \times C$ to an element in C where $C = \mathcal{M}X\widehat{C}$. Input to this function is a pair (c_1, c_2) where c_1 and c_2 are level-1 ciphertexts. i.e. $c_1, c_2 \in \mathcal{M}X\widehat{C}$ where \mathcal{M} is a message space and \widehat{C} is a cipher text space (i.e. LHE cipher text space). The ciphertexts c_1 and c_2 are sequences of two elements, $c_1 = [u_1, \beta_1]$, $c_2 = [u_2, \beta_2]$ where $u_1, u_2 \in \mathcal{M}$ and $\beta_1, \beta_2 \in \widehat{C}$. We write the pairs β_i 's as $\beta_1 = [\beta_{11}, \beta_{12}]$ and $\beta_2 = [\beta_{21}, \beta_{22}]$. Output of this function is a level-1 cipher text $c = [u, \beta]$ where $u \in \mathcal{M}$ and $\beta \in \widehat{C}$.

Algorithm 15 gives the addition function on level 1 ciphertexts of the Boosted-LHE scheme.

Algorithm 15 Boosted-LHE Addition function, level 1 (Add₁)

Input: Ciphertexts $c_1 = [u_1, \beta_1]$; $c_2 = [u_2, \beta_2]$ where $u_1, u_2 \in \mathcal{M}$ and $\beta_1, \beta_2 \in \widehat{C}$

Output: Ciphertext $c = [u, \beta] = \text{Enc}_B(m_1 + m_2)$

 Compute $u = u_1 + u_2$

 Compute $\beta = \beta_1 \boxplus \beta_2$

return u, β

The following theorem proves the correctness of addition function on level 1 ciphertexts of boosted linearly homomorphic scheme.

Theorem 8 *Assume that m_1, m_2 are messages from the message space \mathcal{M} and b_1, b_2 are randomly picked numbers from \mathcal{M} . If $c_1 = [u_1, \beta_1] = \text{Enc}_B(m_1)$, $c_2 = [u_2, \beta_2] = \text{Enc}_B(m_2)$ and c is the output of Algorithm 15 (i.e. $c = \text{Add}_1(c_1, c_2)$), then $c = \text{Enc}_B(m_1 + m_2)$.*

Proof : We assume for the ciphertexts $c_1 = [u_1, \beta_1] \in C$ and $c_2 = [u_2, \beta_2] \in C$ where $u_1 = m_1 - b_1 \in \mathcal{M}$ and $u_2 = m_2 - b_2 \in \mathcal{M}$ for some random numbers $b_1, b_2 \in \mathcal{M}$. Also $\beta_1 = \widehat{\text{Enc}}(b_1) \in \widehat{C}$ and $\beta_2 = \widehat{\text{Enc}}(b_2) \in \widehat{C}$. We have then,

$$c = \text{Add}_1(c_1, c_2) = [u, \beta]$$

where

$$u = u_1 + u_2 = m_1 - b_1 + m_2 - b_2 = (m_1 + m_2) - (b_1 + b_2) \quad (4.1)$$

$$\beta = \beta_1 \boxplus \beta_2 = \widehat{\text{Enc}}(b_1) \boxplus \widehat{\text{Enc}}(b_2) = \widehat{\text{Enc}}(b_1 + b_2) \quad (4.2)$$

by using Algorithm 10. We can write

$$u = (m_1 + m_2) - b \quad (4.3)$$

$$\beta = \widehat{\text{Enc}}(b) \quad (4.4)$$

where $b = b_1 + b_2$. Hence, the sequence $c = [u, \beta]$ gives the encryption of the message $m_1 + m_2$ i.e. $c = \text{Add}_1(\text{Enc}_B(m_1), \text{Enc}_B(m_2)) = \text{Enc}_B(m_1 + m_2)$. \blacksquare

Multiplication in the Boosted LHE, level 1 (Mult₁): This function defines the multiplication between two level 1 ciphertexts in C and outputs a level 2 ciphertext. That is, **Mult₁** maps an element from $C \times C$ to an element in $\widehat{C} \times \widehat{C}^{2l}$ where $l = 1$. Input to this function is a pair of ciphertexts (c_1, c_2) where c_1 and c_2 are level-1 ciphertexts. i.e. $c_1, c_2 \in C = \mathcal{M} \times \widehat{C}$ where \mathcal{M} is a message space and \widehat{C} is a cipher text space (i.e. LHE cipher text space). Each of the ciphertexts c_1 and c_2 is a sequences of two elements, we write, $c_1 = [u_1, \beta_1]$ and $c_2 = [u_2, \beta_2]$ where $u_1, u_2 \in \mathcal{M}$ and $\beta_1, \beta_2 \in \widehat{C}$. We express β_1 and β_2 as $\beta_1 = [\beta_{11}, \beta_{12}]$ and $\beta_2 = [\beta_{21}, \beta_{22}]$ respectively since $\beta_1 = \widehat{\text{Enc}}(b_1)$ and $\beta_2 = \widehat{\text{Enc}}(b_2)$ for some $b_1, b_2 \in \mathcal{M}$ (by Algorithm 14). Output of this function is a level-2 cipher text $c = [\alpha, \beta]$ where $\alpha \in \widehat{C}$ and $\beta \in \widehat{C}^2$.

Algorithm 16 gives the multiplication function on level 1 ciphertexts of the Boosted-LHE scheme.

The following theorem proves the correctness of multiplication function on level 1 ciphertexts of boosted linearly homomorphic scheme.

Theorem 9 *Assume that m_1, m_2 are messages from the message space \mathcal{M} and b_1, b_2 are randomly picked numbers from \mathcal{M} . If $c_1 = [u_1, \beta_1] = \text{Enc}_B(m_1)$, $c_2 = [u_2, \beta_2] =$*

Algorithm 16 Boosted-LHE multiplication function, level 1 (**Mult₁**)

Input: Ciphertexts $c_1 = [u_1, \beta_1]$ and $c_2 = [u_2, \beta_2]$ where $u_1, u_2 \in \mathcal{M}$ and $\beta_1, \beta_2 \in \widehat{C}$

Output: Ciphertext $c = [\alpha, \beta] = \text{Enc}_B(m_1 \cdot m_2)$

Compute $\alpha := \widehat{\text{Enc}}(u_1 \cdot u_2) \boxplus (u_1 \boxminus \beta_2) \boxplus (u_2 \boxminus \beta_1)$

Compute $\beta := \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} \beta_{11} & \beta_{12} \\ \beta_{21} & \beta_{22} \end{pmatrix}$

return α, β

$\text{Enc}_B(m_2)$ and c is the output of Algorithm 16 (i.e. $c = \text{Mult}_1(c_1, c_2)$), then one can decrypt c and recover $m_1 \cdot m_2$.

Proof : We assume for the ciphertexts $c_1 = [u_1, \beta_1] \in C$ and $c_2 = [u_2, \beta_2] \in C$ where $u_1 = m_1 - b_1 \in \mathcal{M}$ and $u_2 = m_2 - b_2 \in \mathcal{M}$ for some random numbers $b_1, b_2 \in \mathcal{M}$. Also $\beta_1 = \widehat{\text{Enc}}(b_1) \in \widehat{C}$ and $\beta_2 = \widehat{\text{Enc}}(b_2) \in \widehat{C}$. We have then,

$$c = \text{Mult}_1(c_1, c_2) = [\alpha, \beta]$$

where

$$\alpha := \widehat{\text{Enc}}(u_1 \cdot u_2) \boxplus (u_1 \boxminus \beta_2) \boxplus (u_2 \boxminus \beta_1) \quad (4.5)$$

$$\beta := \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} \beta_{11} & \beta_{12} \\ \beta_{21} & \beta_{22} \end{pmatrix} \quad (4.6)$$

We substitute the values of u_i 's and β_i 's from Algorithm 14 and observe that,

$$\begin{aligned} \alpha &= \widehat{\text{Enc}}((m_1 - b_1) \cdot (m_2 - b_2)) \boxplus ((m_1 - b_1) \boxminus \widehat{\text{Enc}}(b_2)) \boxplus ((m_2 - b_2) \boxminus \widehat{\text{Enc}}(b_1)) \\ &= \widehat{\text{Enc}}((m_1 - b_1) \cdot (m_2 - b_2)) \boxplus \widehat{\text{Enc}}((m_1 - b_1) \cdot b_2) \boxplus \widehat{\text{Enc}}((m_2 - b_2) \cdot b_1)) \\ &= \widehat{\text{Enc}}((m_1 - b_1) \cdot (m_2 - b_2) + ((m_1 - b_1) \cdot b_2) + ((m_2 - b_2) \cdot b_1)) \\ &= \widehat{\text{Enc}}(m_1 m_2 - b_1 m_2 - b_2 m_1 + b_1 b_2 + m_1 b_2 - b_1 b_2 + m_2 b_1 - b_1 b_2) \end{aligned}$$

which is actually,

$$\widehat{\text{Enc}}(m_1 m_2 - b_1 b_2)$$

and

$$\beta = (\widehat{\text{Enc}}(b_1), \widehat{\text{Enc}}(b_2))^T \quad (4.7)$$

Hence, one can recover $m_1 m_2$ as follows:

$$\begin{aligned} \widehat{\text{Dec}}(\alpha) + \widehat{\text{Dec}}(\beta_1) \cdot \widehat{\text{Dec}}(\beta_2) &= \widehat{\text{Dec}}(\alpha) + \widehat{\text{Dec}}(\widehat{\text{Enc}}(b_1)) \cdot \widehat{\text{Dec}}(\widehat{\text{Enc}}(b_2)) \\ &= (m_1 m_2 - b_1 b_2) + (b_1 \cdot b_2) \\ &= m_1 m_2 - b_1 b_2 + b_1 b_2 \\ &= m_1 m_2 \end{aligned} \quad (4.8)$$

The decryption function in the left hand side of equation (4.8) we define as decryption function on level 2 ciphertexts (for $l_1 = l_2 = 1$) later. \blacksquare

Addition in the Boosted LHE, level 2 (Add₂): This function defines the addition between two level 2 ciphertexts $c_1, c_2 \in C$ and outputs another level 2 ciphertext c . That is, **Add₂** maps an element from $(\widehat{C} \times \widehat{C}^{2l_1}) \times (\widehat{C} \times \widehat{C}^{2l_2})$ to an element in $(\widehat{C} \times \widehat{C}^{2(l_1+l_2)})$ where l_1 and l_2 are the number of degree 2 terms added together in c_1 and c_2 respectively. So, input to the addition function in the boosted linearly homomorphic scheme for level 2 is a pair of ciphertexts (c_1, c_2) where c_1 and c_2 are a level-2 ciphertexts, that is $c_1 \in \widehat{C} \times \widehat{C}^{2l_1}$, $c_2 \in \widehat{C} \times \widehat{C}^{2l_2}$ where \widehat{C} , \widehat{C}^{2l_1} and \widehat{C}^{2l_2} are ciphertext spaces (i.e. LHE cipher text space). For messages $m_1, m_2 \in \mathcal{M}$, the ciphertexts c_1 and c_2 are sequences of two elements we denoted by $c_1 = [\alpha_1, \beta_1] = \text{Enc}_B(m_1)$ and $c_2 = [\alpha_2, \beta_2] = \text{Enc}_B(m_2)$ where $\alpha_1, \alpha_2 \in \widehat{C}$, $\beta_1 \in \widehat{C}^{2l_1}$ and $\beta_2 \in \widehat{C}^{2l_2}$. By Algorithm 14, each $c_i = [\alpha_i, \beta_i]$ such that $\alpha_i = \widehat{\text{Enc}}(m_i - b_i)$ for some $b_i \in \mathcal{M}$ and $\beta_i := \begin{pmatrix} \beta_{11}^{(i)} & \beta_{12}^{(i)} & \cdots & \beta_{1l_i}^{(i)} \\ \beta_{21}^{(i)} & \beta_{22}^{(i)} & \cdots & \beta_{2l_i}^{(i)} \end{pmatrix}$ and $\beta_{1k}^{(i)} = \widehat{\text{Enc}}(b_{1k}^{(i)})$ and $\beta_{2k}^{(i)} = \widehat{\text{Enc}}(b_{2k}^{(i)})$ for some $b_{1k}^{(i)}, b_{2k}^{(i)} \in \mathcal{M}$ where $1 \leq k \leq l_i$. Output of this function is a level-2 cipher text $c = [\alpha, \beta]$ where $\alpha \in \widehat{C}$ and $\beta \in \widehat{C}^{2(l_1+l_2)}$. Algorithm 17 gives the addition function for boosted linearly homomorphic encryption scheme for level 2 ciphertexts.

Algorithm 17 Boosted-LHE addition function, level 2 (Add_2)

Input: Ciphertexts $c_1 = [\alpha_1, \beta_1] = \text{Enc}_B(m_1)$ and $c_2 = [\alpha_2, \beta_2] = \text{Enc}_B(m_2)$ where

$$m_1, m_2 \in \mathcal{M}, \alpha_1, \alpha_2 \in \widehat{C}, \beta_1 \in \widehat{C}^{2l_1} \text{ and } \beta_2 \in \widehat{C}^{2l_2}$$

Output: Ciphertext $c = [\alpha, \beta] = \text{Enc}_B(m_1 + m_2)$

Compute $\alpha := \alpha_1 \boxplus \alpha_2$

$$\text{Compute } \beta := (\beta_1 || \beta_2) = \begin{pmatrix} \beta_{11}^{(1)} & \beta_{12}^{(1)} & \cdots & \beta_{1l_1}^{(1)} & \beta_{11}^{(2)} & \beta_{12}^{(2)} & \cdots & \beta_{1l_2}^{(2)} \\ \beta_{21}^{(1)} & \beta_{22}^{(1)} & \cdots & \beta_{2l_1}^{(1)} & \beta_{21}^{(2)} & \beta_{22}^{(2)} & \cdots & \beta_{2l_2}^{(2)} \end{pmatrix}$$

return α, β

The following theorem proves the correctness of addition function on level 2 ciphertexts of boosted linearly homomorphic scheme.

Theorem 10 *If $c_i = [\alpha_i, \beta_i]$ such that $\alpha_i = \widehat{\text{Enc}}(m_i - b_i)$ for some $b_i \in \mathcal{M}$, $\beta_i := \begin{pmatrix} \beta_{11}^{(i)} & \beta_{12}^{(i)} & \cdots & \beta_{1l_i}^{(i)} \\ \beta_{21}^{(i)} & \beta_{22}^{(i)} & \cdots & \beta_{2l_i}^{(i)} \end{pmatrix}$ where $\beta_{1k}^{(i)} = \widehat{\text{Enc}}(b_{1k}^{(i)})$, also $\beta_{2k}^{(i)} = \widehat{\text{Enc}}(b_{2k}^{(i)})$ for some $b_{1k}^{(i)}, b_{2k}^{(i)} \in \mathcal{M}$ with $1 \leq k \leq l_i$ and $\sum_{k=1}^{l_i} [b_{1,k}^{(i)} \cdot b_{2,k}^{(i)}] = b_i$, then c can be computed (knowing pk) and given c , one can decrypt c and recover $m_1 + m_2$ (knowing sk).*

Proof : We assume that the ciphertext $c_i = [\alpha_i, \beta_i]$ such that $\alpha_i = \widehat{\text{Enc}}(m_i - b_i)$ for some $b_i \in \mathcal{M}$, $\beta_i := \begin{pmatrix} \beta_{11}^{(i)} & \beta_{12}^{(i)} & \cdots & \beta_{1l_i}^{(i)} \\ \beta_{21}^{(i)} & \beta_{22}^{(i)} & \cdots & \beta_{2l_i}^{(i)} \end{pmatrix}$ where $\beta_{1k}^{(i)} = \widehat{\text{Enc}}(b_{1k}^{(i)})$, also $\beta_{2k}^{(i)} = \widehat{\text{Enc}}(b_{2k}^{(i)})$ for some $b_{1k}^{(i)}, b_{2k}^{(i)} \in \mathcal{M}$ with $1 \leq k \leq l_i$ and $\sum_{k=1}^{l_i} [b_{1,k}^{(i)} \cdot b_{2,k}^{(i)}] = b_i$, then

$$c = \text{Add}_2(c_1, c_2) = [\alpha, \beta]$$

where

$$\alpha := \alpha_1 \boxplus \alpha_2 \tag{4.9}$$

$$\beta := (\beta_1 || \beta_2) = \begin{pmatrix} \beta_{11}^{(1)} & \beta_{12}^{(1)} & \cdots & \beta_{1l_1}^{(1)} & \beta_{11}^{(2)} & \beta_{12}^{(2)} & \cdots & \beta_{1l_2}^{(2)} \\ \beta_{21}^{(1)} & \beta_{22}^{(1)} & \cdots & \beta_{2l_1}^{(1)} & \beta_{21}^{(2)} & \beta_{22}^{(2)} & \cdots & \beta_{2l_2}^{(2)} \end{pmatrix} \tag{4.10}$$

Observe that

$$\begin{aligned}
\alpha &= \alpha_1 \boxplus \alpha_2 \\
&= \widehat{\text{Enc}}((m_1 - b_1) + (m_2 - b_2)) \\
&= \widehat{\text{Enc}}((m_1 + m_2) - (b_1 + b_2))
\end{aligned}$$

where $b_1 + b_2 = \sum_{k=1}^{l_1} [b_{1,k}^{(1)} \cdot b_{2,k}^{(1)}] + \sum_{k=1}^{l_2} [b_{1,k}^{(2)} \cdot b_{2,k}^{(2)}]$. Hence, one can recover $m_1 + m_2$ as follows:

$$\begin{aligned}
\widehat{\text{Dec}}(\alpha) + \sum_{k=1}^{l_1+l_2} [\widehat{\text{Dec}}(\beta_{1k}) \cdot \widehat{\text{Dec}}(\beta_{2k})] &= \widehat{\text{Dec}}(\alpha) + \sum_{k=1}^{l_1} [\widehat{\text{Dec}}(\beta_{1k}^{(1)}) \cdot \widehat{\text{Dec}}(\beta_{2k}^{(1)})] \\
&\quad + \sum_{k=1}^{l_2} [\widehat{\text{Dec}}(\beta_{1k}^{(2)}) \cdot \widehat{\text{Dec}}(\beta_{2k}^{(2)})] \tag{4.11} \\
&= ((m_1 + m_2) - (b_1 + b_2)) + \sum_{k=1}^{l_1} [b_{1,k}^{(1)} \cdot b_{2,k}^{(1)}] \\
&\quad + \sum_{k=1}^{l_2} [b_{1,k}^{(2)} \cdot b_{2,k}^{(2)}] \\
&= ((m_1 + m_2) - (b_1 + b_2)) + b_1 + b_2 \\
&= m_1 + m_2
\end{aligned}$$

The decryption function in the left hand side of equation (4.11) we define as decryption function on level 2 ciphertexts (for l_1 and l_2) later. \blacksquare

Scalar multiplication in the Boosted LHE, level 1 (SMult₁): Scalar Multiplication function in the boosted linearly homomorphic scheme operates over level 1 ciphertexts and is a function that maps an element from $\mathcal{M} \times (\mathcal{M} \times \widehat{C})$ to an element in $\mathcal{M} \times \widehat{C}$. So, input to this function is a pair (s, c) where $s \in \mathcal{M}$ is a scalar/constant, \mathcal{M} is a message space and c is a level-1 ciphertext. Here, $c \in \mathcal{M} \times \widehat{C}$ where \widehat{C} is a cipher text space (i.e. LHE cipher text space). The level-1 ciphertext c is a sequence of two elements, $c = [u, \beta] = \text{Enc}_{\mathbf{B}}(m)$ where $u \in \mathcal{M}$ and $\beta \in \widehat{C}$ for message $m \in \mathcal{M}$. We express β as $\beta = [\beta_1, \beta_2]$ since $\beta = \widehat{\text{Enc}}(b)$ for some $b \in \mathcal{M}$ (by Algorithm 14). Output

of this function is also a level-1 cipher text $c' = [u', \beta']$ where $u' \in \mathcal{M}$ and $\beta' \in \widehat{C}$ such that $c' = \text{Enc}_B(s \cdot m)$. Algorithm 18 gives the scalar multiplication function for boosted linearly homomorphic scheme on level 1 ciphertexts.

Algorithm 18 Boosted-LHE scalar multiplication function, level 1 (SMult_1)

Input: Ciphertext $c = [u, \beta] = \text{Enc}_B(m)$ where $u \in \mathcal{M}$ and $\beta = [\beta_1, \beta_2] \in \widehat{C}$, scalar $s \in \mathcal{M}$

Output: Ciphertext $c' = [u', \beta'] = \text{Enc}_B(s \cdot m)$

 Compute $u' := s \cdot u$

 Compute $\beta' := s \boxdot \beta$

return u', β'

The following theorem proves the correctness of scalar multiplication function on level 1 ciphertexts of boosted linearly homomorphic scheme.

Theorem 11 *Assume that the level 1 ciphertext $c = [u, \beta] = \text{Enc}_B(m) \in C$ and a scalar $s \in \mathcal{M}$ where $u = m - b \in \mathcal{M}$ for some random number $b \in \mathcal{M}$, $\beta = \widehat{\text{Enc}}(b) \in \widehat{C}$ and c is the output of Algorithm 18 (i.e. $c' = \text{SMult}_1(s, c)$), then $c' = \text{Enc}_B(s \cdot m)$.*

Proof : We assume for the ciphertext $c = [u, \beta] \in C$ and a scalar $s \in \mathcal{M}$ where $u = m - b \in \mathcal{M}$ for some random number $b \in \mathcal{M}$ and $\beta = \widehat{\text{Enc}}(b) \in \widehat{C}$. We have then,

$$c' = \text{SMult}_1(s, c) = [u', \beta'] \quad (4.12)$$

where

$$u' = s \cdot u = s \cdot (m - b) = (sm - sb) \quad (4.13)$$

$$\beta' := s \boxdot [\beta_1, \beta_2] = s \boxdot \widehat{\text{Enc}}(b) = \widehat{\text{Enc}}(s \cdot b) \quad (4.14)$$

From equation (4.12),

$$c' = [u', \beta'] = [sm - sb, \widehat{\text{Enc}}(sb)] \quad (4.15)$$

which finally yields,

$$c' = \text{Enc}_B(sm) \quad (4.16)$$

Hence, the sequence $c' = [u', \beta']$ gives the encryption of the message sm i.e. $c = \text{SMult}_1(s, \text{Enc}_B(m)) = \text{Enc}_B(sm)$. \blacksquare

Scalar multiplication in the Boosted LHE, level 2 (SMult₂): Scalar Multiplication function in the boosted linearly homomorphic scheme operates over level 2 ciphertexts and is a function that maps an element from $\mathcal{M} \times (\widehat{C} \times \widehat{C}^{2l})$ to an element in $\widehat{C} \times \widehat{C}^{2l}$ where \mathcal{M} is the message space, \widehat{C} is a cipher text space (i.e. LHE cipher text space), l is number of degree 2 terms added together in c for an element $c \in \widehat{C} \times \widehat{C}^{2l}$. So, input to this function is a pair (s, c) where $s \in \mathcal{M}$ is a scalar and c is a level-2 ciphertext such that $c \in \widehat{C} \times \widehat{C}^{2l}$. The ciphertext c is a sequence of two elements such that $c = [\alpha, \beta] = \text{Enc}_B(m)$ where $m \in \mathcal{M}$, $\alpha \in \widehat{C}$ and $\beta \in \widehat{C}^{2l}$. We express β as $\beta := \begin{pmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1l} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2l} \end{pmatrix}$ for each $\beta_{ik} = \widehat{\text{Enc}}(b_{ik})$ (by Algorithm 14). Output of this function is also a level-2 cipher text $c' = [\alpha', \beta'] = \text{Enc}_B(s \cdot m)$ where $\alpha' \in \widehat{C}$ and $\beta' \in \widehat{C}^{2l}$. Algorithm 19 gives the scalar multiplication function for boosted linearly homomorphic scheme on level 1 ciphertexts.

Algorithm 19 Boosted-LHE scalar multiplication function, level 2 (SMult₂)

Input: Ciphertext $c = [\alpha, \beta] = \text{Enc}_B(m)$ and scalar s where $s, m \in \mathcal{M}, \alpha \in \widehat{C}$ and $\beta \in \widehat{C}^{2l}$

Output: Ciphertext $c' = [\alpha', \beta'] = \text{Enc}_B(s \cdot m)$

Compute $\alpha' := s \boxtimes \alpha$

Compute $\beta' := \begin{pmatrix} s \boxtimes \beta_{11} & s \boxtimes \beta_{12} & \cdots & s \boxtimes \beta_{1l} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2l} \end{pmatrix}$

return α', β'

The following theorem proves the correctness of scalar multiplication function on

level 2 ciphertexts of boosted linearly homomorphic scheme.

Theorem 12 *Assume that the level 2 ciphertext $c = [\alpha, \beta] = \text{Enc}_B(m) \in C$ and a scalar $s \in \mathcal{M}$ such that $\alpha = \widehat{\text{Enc}}(m - b) \in \widehat{C}$ for some random number $b \in \mathcal{M}$, $\beta = \widehat{\text{Enc}}(b) \in \widehat{C}^{2l}$ for some number l and c is the output of Algorithm 19 (i.e. $c' = \text{SMult}_2(s, c)$), then one can decrypt c' and recover $s \cdot m$.*

Proof : We assume that the ciphertext $c = [\alpha, \beta] = \text{Enc}_B(m) \in C$ and a scalar $s \in \mathcal{M}$ such that $\alpha = \widehat{\text{Enc}}(m - b) \in \widehat{C}$ for some random number $b \in \mathcal{M}$, $\beta = \widehat{\text{Enc}}(b) \in \widehat{C}^{2l}$ for some number l . We have then,

$$c' = \text{SMult}_2(s, c) = [\alpha', \beta'] \quad (4.17)$$

where

$$\alpha' := s \boxdot \alpha \quad (4.18)$$

$$\beta' := \begin{pmatrix} \beta_{11}' & \beta_{12}' & \cdots & \beta_{1l}' \\ \beta_{21}' & \beta_{22}' & \cdots & \beta_{2l}' \end{pmatrix} \quad (4.19)$$

where $\beta_{1k}' = s \boxdot \beta_{1k}$ and $\beta_{2k}' = \beta_{2k}$. Observe that,

$$\begin{aligned} \alpha' &= s \boxdot \alpha \\ &= s \boxdot \widehat{\text{Enc}}(m - b) \\ &= \widehat{\text{Enc}}(s \cdot (m - b)) \\ &= \widehat{\text{Enc}}(sm - sb) \end{aligned} \quad (4.20)$$

where $sb = s \boxdot \sum_{k=1}^l [b_{1,k} \cdot b_{2,k}] = \sum_{k=1}^l [sb_{1,k} \cdot b_{2,k}]$. Hence, one can recover $s \cdot m$ as follows:

$$\begin{aligned} \widehat{\text{Dec}}(\alpha') + \sum_{k=1}^l [\widehat{\text{Dec}}(\beta_{1k}') \cdot \widehat{\text{Dec}}(\beta_{2k}')] &= (sm - sb) + \sum_{k=1}^l [sb_{1,k} \cdot b_{2,k}] \\ &= (sm - sb) + sb \\ &= sm \end{aligned}$$

The decryption function given above we define as decryption function on level 2 ciphertexts (for l) later. ■

4.2.4 Decryption functions for the Boosted LHE scheme (Dec_B)

Decryption functions for boosted linearly homomorphic encryption scheme are designed in two levels of ciphertexts. Level 1 ciphertexts are ciphertexts that encode either “fresh” messages or messages that are the linear combinations of “fresh” messages whereas level 2 ciphertexts are ciphertexts that are “multiplied” level 1 ciphertexts.

Level 1 decryption function (Dec1): Input to the level 1 decryption function is a pair (sk, c) where sk is the secret key and c is the ciphertext. The ciphertext $c \in \mathcal{M} \times \widehat{C}$ where \mathcal{M} is a message space and \widehat{C} is a cipher text space (i.e. LHE ciphertext space). A ciphertext is a sequence of two elements u and β where $u \in \mathcal{M}$ and $\beta \in \widehat{C}$. i.e. $c = [u, \beta]$. Output of this function is the message $m \in \mathcal{M}$. Algorithm 20 gives the level 1 decryption function of the Boosted-LHE scheme.

Algorithm 20 Boosted-LHE Decryption Level 1(Dec1)

Input: Ciphertext c , secret key $sk = a$

Output: Message m

 Compute $m := u + \widehat{\text{Dec}}(\beta)$

return m

The following theorem shows the correctness of the level 1 decryption function that maps C to the message space \mathcal{M} .

Theorem 13 *If $c = [u, \beta] \in C$ is an encryption of $m \in \mathcal{M}$, then $\text{Dec1}(c) = m$.*

Proof : We start with $\text{Dec1}(c)$ which is $\text{Dec1}([u, \beta])$. We substitute the values of u and β from Algorithm 14 and get,

$$\text{Dec1}([m - b, \widehat{\text{Enc}}(b)])$$

We now apply Algorithm 20, this becomes,

$$m - b + \widehat{\text{Dec}}(\widehat{\text{Enc}}(b))$$

which finally yields $m - b + b$ and thus m . Hence we have, $\text{Dec1}(c) = m$. \blacksquare

Level 2 decryption function (Dec2): Decryption function for level 2 ciphertexts has a pair (sk, c) as input where sk is the secret key and $c \in \widehat{C} \times \widehat{C}^{2l}$. Here, \widehat{C} is the cipher text space (i.e. LHE ciphertext space). The ciphertext c is a sequence of elements α and β ; where $\alpha \in \widehat{C}$ and $\beta \in \widehat{C}^{2l}$. We represent β by a $2 \times l$ matrix $((\beta_{11}, \beta_{21})^T (\beta_{12}, \beta_{22})^T \cdots (\beta_{1l}, \beta_{2l})^T)$. Output of this function is the message $m \in \mathcal{M}$. Algorithm 21 gives the level 2 decryption function of the Boosted-LHE scheme.

Algorithm 21 Boosted-LHE Decryption Level 2 (Dec2)

Input: Ciphertext c , secret key $sk = a$

Output: Message m

Compute $m := \widehat{\text{Dec}}(\alpha) + \sum_{i=1}^l (\widehat{\text{Dec}}(\beta_{1i}).\widehat{\text{Dec}}(\beta_{2i}))$

return m

The following theorem shows the correctness of the level 2 decryption function that maps C to the message space \mathcal{M} .

Theorem 14 *If a level 2 ciphertext $c = [\alpha, \beta] \in C$ is an encryption of $m \in \mathcal{M}$, then $\text{Dec2}(c) = m$.*

Proof : We start with $\text{Dec2}(c)$ which is $\text{Dec2}([\alpha, \beta])$. We substitute the values of α and β from Algorithm 14 and get,

$$\text{Dec2}([\widehat{\text{Enc}}(m - b), \begin{pmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1l} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2l} \end{pmatrix}])$$

where for each $\beta_{ik} = \widehat{\text{Enc}}(b_{ik})$. We now apply Algorithm 21, this becomes,

$$\begin{aligned} \widehat{\text{Dec}}(\alpha) + \sum_{i=1}^l [\widehat{\text{Dec}}(\beta_{1i}) \cdot \widehat{\text{Dec}}(\beta_{2i})] &= m - b + \sum_{i=1}^l [\widehat{\text{Dec}}(\widehat{\text{Enc}}(b_{1i})) \cdot \widehat{\text{Dec}}(\widehat{\text{Enc}}(b_{2i}))] \\ &= m - b + \sum_{i=1}^l [b_{1i} \cdot b_{2i}] \end{aligned}$$

which finally yields $m - b + b$ and thus m . Hence we have, $\text{Dec2}(c) = m$. \blacksquare

4.3 SECURITY OF THE BOOSTED LH ENCRYPTION SCHEME

The semantic security of boosted LHE scheme HE_B depends on the semantic security of the underlying homomorphic encryption scheme $\widehat{\text{HE}}$ since $(m_0 - b, \widehat{\text{Enc}}(b)) \approx (m_0 - b, \widehat{\text{Enc}}(0)) \equiv (m_1 - b, \widehat{\text{Enc}}(0)) \approx (m_1 - b, \widehat{\text{Enc}}(b))$ where “ \approx ” means the computational indistinguishability of the distributions (which follows assuming the semantic security of $\widehat{\text{HE}}$) and “ \equiv ” means that the distributions are identical [4] (since $m_0 - b = m_1 - (-m_0 + m_1 + b)$ that is, $m_0 - b = m_1 - b'$ for $b' = -m_0 + m_1 + b$). It is stated and also proved in the paper [4] that if $\widehat{\text{HE}}$ is circuit private, then HE_B is also a leveled circuit private homomorphic encryption (see Theorem 2 of the paper). There are some other theorems that are proved for the security of the boosted LHE, we refer to the paper by Catalano and Fiore [4] for details.

4.4 EFFICIENCY OF THE BOOSTED LH ENCRYPTION SCHEME

While describing this boosting procedure of linearly homomorphic encryption scheme, Catalano and Fiore emphasized about implementing a re-randomization of the ciphertexts in their paper [4]. This re-randomization is crucial for achieving circuit privacy and operated over both level-1 and level-2 ciphertexts. The input of this procedure is a ciphertext that encrypts a message m using a random $padb$ whereas it outputs a ciphertext which is a new encryption of the same message m but using a *fresh* random $padb^*$ (for the procedure of this re-randomizations and their correctness, we

refer to the paper [4] by Catalano and Fiore). The resulting scheme achieves circuit privacy and compactness when considering 2nd degree polynomial sub-classes where the number of additions of degree-2 terms is bounded by a constant. The efficiency of the boosted linearly homomorphic scheme is given in the Table 4.1, where the cost of each function can easily be derived from our arguments in previous sections.

4.5 REMARKS

The simple, easy to implement boosted linearly homomorphic scheme is efficient and provides desirable security. We tried to implement this scheme on the CGS linearly homomorphic scheme [16] to enable CGS to allow one single multiplication and unbounded number of additions. We refer to Chapter 5 for details.

Table 4.1: Efficiency of Boosted LHE Encryption Scheme

Functions	Operations (we denote multiplication by M, pseudo random number generation by PRNG, scalar multiplication by SM, division by D, subtraction by S, addition by A and exponentiation by E)
B-LHE Key generation	same as underlying LHE
B-LHE Encryption	1PRNG + 1S+ 1 LHE encryption
B-LHE Decryption (l is # of column in β)	1A + 1 LHE decryption (for Dec1) and $(2l + 1)$ LHE decryption + l M+ l A (for Dec2)
B-LHE Addition, level-1	1A + 1 LHE A
B-LHE Multiplication, level 1	1M+ 2 LHE SM + 2 LHE A + 1 LHE encryption
B-LHE Addition, level-2	1 LHE A
B-LHE Scalar multiplication, level-1	1M+ 2 LHE SM
B-LHE Scalar multiplication, level-2 (l is # of column in β)	$(l + 1)$ LHE SM

CHAPTER 5

BOOSTED-CGS ENCRYPTION SCHEME

5.1 INTRODUCTION

As discussed in Chapter 4, boosted linearly homomorphic scheme can be implemented to mostly all the well-known linearly homomorphic schemes. There are other partially homomorphic schemes we discussed in Chapter 2, RSA scheme and ElGamal scheme. Both of these schemes are multiplicatively homomorphic. Boosting scheme we presented in Chapter 4 is restricted to only linearly homomorphic encryption schemes. In chapter 3, we discussed about CGS linearly homomorphic scheme which was designed by modifying ElGamal encryption scheme but this scheme is linearly homomorphic, thus not capable of boosting RSA or ElGamal schemes. In this chapter, we implement boosted linearly homomorphic encryption scheme based on the CGS linearly homomorphic encryption scheme. This implementation computations of degree 2 polynomials over the ciphertexts. i.e. it would allow at most one product with unbounded number of additions and scalar multiplications over the encrypted plaintexts. We also provide MAGMA source codes for this boosted CGS scheme.

5.2 DESCRIPTION

The construction of the boosted CGS linearly homomorphic scheme would follow from the construction of boosted LHE in Chapter 4, thus consists of four faces: key generation, encryption, evaluation functions and decryption. As described in chapter 4, the boosting LH encryption methodology is converting CGS public-key linearly homomorphic encryption scheme (i.e. $\widehat{\text{HE}} = (\widehat{\text{KeyGen}}, \widehat{\text{Enc}}, \widehat{\text{Eval}}, \widehat{\text{Dec}})$) is this

case $\text{CGS} = (\text{KeyGen}_{\text{CGS}}, \text{Enc}_{\text{CGS}}, \text{Eval}_{\text{CGS}}, \text{Dec}_{\text{CGS}})$ (with message space $\mathcal{M}(\text{CGS}) = \mathcal{M}$, and ciphertext space $C(\text{CGS}) = \tilde{C}$) to a homomorphic encryption scheme supporting one multiplication (i.e. $\text{HE}_B = (\text{KeyGen}_B, \text{Enc}_B, \text{Eval}_B, \text{Dec}_B)$ is in this case $\text{BCGS} = (\text{KeyGen}_B, \text{Enc}_B, \text{Eval}_B, \text{Dec}_B)$ with ciphertext space $C(\text{BCGS}) = C$). The addition and multiplication operation on the message space \mathcal{M} are denoted by $+$ and \cdot respectively. Also the addition and multiplication operation on the CGS ciphertext space \tilde{C} are denoted by \boxplus and \boxtimes respectively. Ciphertexts in BCGS scheme are of two levels: level 1 and level 2. Level 1 ciphertexts are ciphertexts that encode either “fresh” messages or messages that are the linear combinations of “fresh” messages whereas level 2 ciphertexts are ciphertexts that are “multiplied” level 1 ciphertexts. The evaluation function algorithms in BCGS scheme are also described by following boosted LHE scheme, in five different procedures we denote by, Add_1 , Mult , Add_2 , SMult_1 , SMult_2 where Add_1 and Mult are the addition and multiplication operation between two level 1 ciphertexts. Add_2 operates over a pair of level 2 ciphertexts, that is, multiplication operation between two level 2 ciphertexts. SMult_1 is the scalar multiplication that operates over a single level 1 ciphertext and SMult_2 is the scalar multiplication that operates over a single level 2 ciphertext.

5.2.1 Key generation function for the Boosted-CGS scheme (KeyGen_B)

The key generation function for the boosted CGS scheme is same as CGS key generation i.e. $\text{KeyGen}_B = \text{KeyGen}_{\text{CGS}}$. Input to this function is 1^n where n is the security parameter that determines the size of the input. Output of this function is a pair (pk, sk) where pk is the public key G and sk is the secret key a . The parameters g , p and q are Domain(system) parameters which are common in all the functions. The message spaces are same for both the encryption scheme, i. e. $\mathcal{M}(\text{CGS}) = \mathcal{M}(\text{BCGS}) = \mathcal{M}$. We refer to the same MAGMA source code we presented for $\text{KeyGen}_{\text{CGS}}$ in Chapter 3.

5.2.2 Encryption function for the Boosted-CGS scheme (Enc_B)

Input to the encryption function for the boosted CGS scheme is a pair (pk, m) where pk is the public key (g, q, G) and m is the message to be encrypted from message space \mathcal{M} . Output of this function is a cipher text c where $c \in \mathcal{M} \times \tilde{C}$ where \tilde{C} is a cipher text space (i.e. CGS cipher text space). The ciphertext c is a sequence of two elements i.e. $c = [u, \beta]$ where $u \in \mathcal{M}$ and $\beta \in \tilde{C}$. Algorithm 14 gives the this procedure of encryption.

Note that, decryption function of the CGS scheme (Section 3.2.4) is a linear function of B for message space $\mathcal{M} = [-B, B]$. As B gets larger, decryption slows down. So, the number B that is the size of the message space needs to be reasonable so that the run-time is efficient as well as the message space is not too small. The encryption algorithm (Algorithm 14) requires a random number b from the message space \mathcal{M} to compute the ciphertext. This b should be chosen from the interval $[m - B, m + B]$. This restriction is due to the fact that when the encryption of the linear combination of messages is computed, the linear combination of messages also should lie in \mathcal{M} .

MAGMA source code for encryption of Boosted-CGS (Enc_B): The following is the MAGMA source code we used for the implementation of this encryption function. We name this function to be Enc_B . We chose B to be 50 for our implementation. The number b is picked from the interval $[m - B, m + B]$ at random for message m . u is computed as $m - b$ and the CGS encryption function Enc_{CGS} is used to compute the encryption of b which is β . The function then returns the sequence $[u, \beta]$ as the ciphertext $c = \text{Enc}_B(m)$.

```
B:=50
Enc_B:= function(m,g,q,G)
b:=Random(m-B,m+B);
```

```

//b is randomly chosen by randomized encryption algorithm
// from message space M
printf "m=%o, b=%o \n", m, b;
u:=m-b;
beta:=Enc_CGS(b,g,q,G);
return u,beta;
end function;

```

5.2.3 Evaluation functions for the Boosted-CGS scheme (Eval_B)

We implement the evaluation functions of the Boosted-LHE scheme where the CGS scheme serves as the underlying linearly homomorphic scheme. Details below:

Addition in the Boosted-CGS scheme, level 1 (Add_1): Input to this function is a pair (c_1, c_2) where c_1 and c_2 are level-1 ciphertexts. The ciphertexts $c_1, c_2 \in \mathcal{M} \times \tilde{C}$ where \mathcal{M} is a message space and \tilde{C} is a cipher text space (i.e. CGS cipher text space). The ciphertexts c_1 and c_2 are sequences of two elements $c_1 = [u_1, \beta_1]$ and $c_2 = [u_2, \beta_2]$ where $u_1, u_2 \in \mathcal{M}$ and $\beta_1, \beta_2 \in \tilde{C}$. We express β_1 and β_2 as $\beta_1 = [\beta_{11}, \beta_{12}]$ and $\beta_2 = [\beta_{21}, \beta_{22}]$. Output of this function is also a level-1 cipher text $c = [u, \beta]$ where $u \in \mathcal{M}$ and $\beta \in \tilde{C}$. We refer Algorithm 15 for the procedure of this function.

MAGMA source code for addition function on level 1 ciphertexts of Boosted-CGS (Add_1): Following is the MAGMA code for the addition function for the level 1 ciphertexts with boosted CGS encryption scheme. u is computed as the usual addition of u_1 and u_2 since this is an addition in the message space \mathcal{M} . To compute β , we used the addition operation \boxplus of the CGS ciphertext space to add β_1 and β_2 . This addition \boxplus is actually the componentwise multiplication. Finally, the function Add_1 returns the ordered sequence $[u, \beta]$ as the addition of c_1 and c_2 .

```

Add_1:= function(u1,beta1,u2,beta2,g,p,G);

```

```

u:=u1+u2;

beta:=[beta1[1]*beta2[1],beta1[2]*beta2[2]];

return u,beta;

end function;

```

Multiplication in the Boosted CGS scheme, level 1 (Mult₁): Input to this function is a pair (c_1, c_2) where c_1 and c_2 is a level-1 ciphertexts. The ciphertexts $c_1, c_2 \in \mathcal{M} \times \tilde{C}$ where \mathcal{M} is a message space and \tilde{C} is a cipher text space (i.e. CGS cipher text space). The ciphertexts c_1 and c_2 are sequences of two elements $c_1 = [u_1, \beta_1]$ and $c_2 = [u_2, \beta_2]$ where $u_1, u_2 \in \mathcal{M}$ and $\beta_1, \beta_2 \in \tilde{C}$. Here, β_1 and β_2 are expressed as $\beta_1 = [\beta_{11}, \beta_{12}]$ and $\beta_2 = [\beta_{21}, \beta_{22}]$. Output of this function is a level-2 cipher text c which is a sequence of two elements, that is, $c = [\alpha, \beta] = \text{Mult}_1(c_1, c_2)$ where $\alpha \in \tilde{C}$ and $\beta \in \tilde{C}^2$. Algorithm 16 gives the construction of multiplication function for level 1 ciphertexts with CGS as underlying linearly homomorphic scheme.

MAGMA source code for multiplication on level 1 ciphertexts of Boosted-CGS (Mult₁): We represent the MAGMA source code for the multiplication function for level 1 ciphertexts in the following. To compute α , we first encrypt the product of u_1 and u_2 using the encryption function of CGS scheme Enc_{CGS} and denote it by \mathbf{u} . Next, we use the scalar multiplication function of CGS scheme SMult to compute the scalar multiplications $u_1 \boxtimes \beta_2$ and $u_2 \boxtimes \beta_1$ and denote by \mathbf{b} and \mathbf{c} respectively. Then, we first add the pair (\mathbf{u}, \mathbf{b}) using Add_{CGS} and take the result to add with \mathbf{u} again using Add_{CGS} . The result of this addition is α . β is computed by taking the column vector as the transpose of the vector $[\beta_1, \beta_2]$. Finally, the function returns the pair $[\alpha, \beta]$ as the output of $\text{Mult}_1(c_1, c_2)$.

```

Mult_1:= function(u1,beta1,u2,beta2,g,p,G);

u:=Enc_CGS(u1*u2,g,q,G);

b:=[beta2[1]^u1,beta2[2]^u1];

```

```

c:=[beta1[1]^u2,beta1[2]^u2];
alpha:=Add_CGS(Add_CGS(u,b,g,p,G),c,g,p,G);
beta:=Matrix(IntegerRing(),2,2,[beta1[1],beta1[2],beta2[1],beta2[2]]);
return alpha,beta;
end function;

```

Addition in the Boosted CGS scheme, level 2 (Add₂): Input to the addition function in the boosted linearly homomorphic scheme for level 2 is a pair of ciphertexts (c_1, c_2) where c_1 and c_2 are a level-2 ciphertexts, that is $c_1 \in \tilde{C} \times \tilde{C}^{2l_1}$, $c_2 \in \tilde{C} \times \tilde{C}^{2l_2}$ where \tilde{C} , \tilde{C}^{2l_1} and \tilde{C}^{2l_2} are ciphertext spaces (i.e. CGS cipher text space). The ciphertexts c_1 and c_2 are sequences of two elements we denoted by $c_1 = [\alpha_1, \beta_1]$ and $c_2 = [\alpha_2, \beta_2]$ where $\alpha_1, \alpha_2 \in \tilde{C}$, $\beta_1 \in \tilde{C}^{2l_1}$ and $\beta_2 \in \tilde{C}^{2l_2}$. We express β_1 and β_2 as $\beta_1 := \begin{pmatrix} \beta_{11}^{(1)} & \beta_{12}^{(1)} & \cdots & \beta_{1l_1}^{(1)} \\ \beta_{21}^{(1)} & \beta_{22}^{(1)} & \cdots & \beta_{2l_1}^{(1)} \end{pmatrix}$ and $\beta_2 := \begin{pmatrix} \beta_{11}^{(2)} & \beta_{12}^{(2)} & \cdots & \beta_{1l_2}^{(2)} \\ \beta_{21}^{(2)} & \beta_{22}^{(2)} & \cdots & \beta_{2l_2}^{(2)} \end{pmatrix}$. Output of this function is a level-2 cipher text $c = [\alpha, \beta]$ where $\alpha \in \tilde{C}$ and $\beta \in \tilde{C}^{2(l_1+l_2)}$. The algorithm 17 gives the procedure for the addition function for Boosted-CGS scheme for level 2 ciphertexts.

MAGMA source code for addition function on level 2 ciphertexts of Boosted-CGS (Add₂): Following is the MAGMA source code we used for adding level 2 ciphertexts with this scheme. Here is an example 2×2 matrix for each β_1 and β_2 , i.e. $l_1 = l_2 = 2$. We used addition operation of CGS scheme Add_{CGS} to add α_1 and α_2 to get α . We adjoin the matrices for β_1 and β_2 to get β . Finally, the function **Add₂** returns the sequence $[\alpha, \beta]$ as the output of **Add₂**(c_1, c_2).

```

Add_2:= function(alpha1,beta1,alpha2,beta2,g,p,G);
alpha:=Add_CGS(alpha1,alpha2,g,p,G);
beta:=Matrix(IntegerRing(),2,4,[beta1[1][1],beta1[1][2],beta2[1][1],
beta2[1][2],beta1[2][1],beta1[2][2],beta2[2][1],beta2[2][2]]);

```

```

    return alpha,beta;
end function;

```

Scalar multiplication in the Boosted CGS scheme, level 1 (SMult₁): Scalar Multiplication function in the Boosted-CGS scheme has input a pair (s, c) where $s \in \mathcal{M}$ is a scalar/constant, \mathcal{M} is a message space and c is a level-1 ciphertext. $c \in \mathcal{M} \times \tilde{C}$ where \tilde{C} is a cipher text space (i.e. CGS cipher text space). The level-1 ciphertext c is a sequence of two elements, $c = [u, \beta]$ where $u \in \mathcal{M}$ and $\beta \in \tilde{C}$. We express β as $\beta = [\beta_1, \beta_2]$. Output of this function is also a level-1 ciphertext $c' = [u', \beta']$ where $u' \in \mathcal{M}$ and $\beta' \in \tilde{C}$. Algorithm 18 gives the scalar multiplication function for boosted CGS scheme on level 1 ciphertexts.

MAGMA source code for scalar multiplication function on level 2 ciphertexts of Boosted-CGS (SMult₁): We represent the MAGMA source code for scalar multiplication on level 1 ciphertexts for Boosted-CGS scheme. u' (u1 in the code) is computed as the product of the scalar s and u . For β' (beta1 in the code), we use the scalar multiplication function for CGS scheme $\text{Smult}_{\text{CGS}}$ to compute the scalar product of s and β . Finally, the function SMult_1 returns the ordered sequence $[u', \beta']$ ([u1,beta1] in the code) as the output of $\text{SMult}_1(s, c)$.

```

SMult_1:= function(s,u,beta,g,p,G);
    u1:=u*s;
    beta1:=Smult_CGS(s,beta,g,p,G);  //[beta[1]^s,beta[2]^s];
    return u1,beta1;
end function;

```

Scalar multiplication in the Boosted CGS scheme, level 2 (SMult₂): Input to this function is a pair (s, c) where $s \in \mathcal{M}$ is a scalar and \mathcal{M} is the message space. The ciphertext c is a level-2 ciphertext such that $c \in \tilde{C} \times \tilde{C}^{2l}$, where \tilde{C} and \tilde{C}^{2l} is

a cipher text space (i.e. CGS cipher text space) of dimension 1 and $2 \times l$ (l is a number that determines the number of column in β). The ciphertext c is a sequence of two elements such that $c = [\alpha, \beta]$ where $\alpha \in \tilde{C}$ and $\beta \in \tilde{C}^{2l}$. We express β as $\beta := \begin{pmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1l} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2l} \end{pmatrix}$. Output of this function is also a level-2 cipher text $c' = [\alpha', \beta']$ where $\alpha' \in \tilde{C}$ and $\beta' \in \tilde{C}^{2l}$. Algorithm 19 gives the procedure for the scalar multiplication function for Boosted-CGS scheme on level 2 ciphertexts.

MAGMA source code for scalar multiplication function on level 2 ciphertexts of

Boosted-CGS (SMult₂): Following is the MAGMA code we implemented for scalar multiplication function on level 2 ciphertexts with Boosted-CGS scheme. α' (**alpha1** in the code) is computed using the scalar multiplication function for the CGS scheme **SMult_{CGS}** with input scalar s and α . β' (**beta1** in the code) is computed taking the scalar product **SMult_{CGS}**(s, β_{1i}) for each i in 1st row of β and computing the matrix of β . Then, the function returns the ordered sequence $[\alpha', \beta']$ as the output of **SMult₂**(s, c).

```
SMult_2:= function(s,alpha,beta,g,p,G);
    alpha1:=Smult_CGS(s,alpha,g,p,G);
    beta1:=Matrix(IntegerRing(),2,2,[(beta[1][1])^s,(beta[1][2])^s,
    beta[2][1],beta[2][2]]);
    return alpha1,beta1;
end function;
```

5.2.4 Decryption functions for the Boosted-CGS scheme (Dec_B)

As it is in the original boosting scheme [4], decryption function for the boosted CGS encryption scheme operates for two levels of ciphertexts. We describe below:

Level 1 decryption function (Dec_{1B}) Input to this function is a pair (sk, c) where sk is the secret key which is a for the CGS scheme and the ciphertext $c \in \mathcal{M} \times \tilde{C}$

where \mathcal{M} is a message space and \tilde{C} is a cipher text space (i.e. CGS ciphertext space). The ciphertext c is a sequence of two elements i.e. $c = [u, \beta]$ where $u \in \mathcal{M}$ and $\beta \in \tilde{C}$. Output of this function is the message $m \in \mathcal{M}$.

Algorithm 20 gives the decryption procedure of level 1 ciphertext that works for Add1 and Smult1.

MAGMA source code for decryption function on level 1 ciphertexts of Boosted-CGS (Dec1_B): Corresponding MAGMA source code we used for the decryption of Add1 and Smult1 is given below. The message is recovered by decrypting β with the CGS decryption function Dec_{CGS} and adding with u .

```
Dec1_B:=function(u,beta,G,a,p)
    m:=u+Dec_CGS(beta,G,a,p);
    return m;
end function;
```

Level 2 decryption function (Dec2_B) Input to this function is a pair (sk, c) where sk is the secret key and $c \in \tilde{C} \times \tilde{C}^{2l}$ where \tilde{C} and \tilde{C}^{2l} are ciphertext spaces (i.e. CGS cipher text spaces) where l is the number of degree 2 terms in plaintexts polynomial corresponding to the ciphertext c . The ciphertext c is a sequence of two elements $c = [\alpha, \beta]$ where $\alpha \in \tilde{C}$ and $\beta \in \tilde{C}^{2l}$. We express $\beta = ((\beta_{11}, \beta_{21})^T, (\beta_{12}, \beta_{22})^T, \dots, (\beta_{1l}, \beta_{2l})^T)$. Output of this function is the message $m \in \mathcal{M}$. We refer to Algorithm 21 for the procedure of this decryption function.

MAGMA source code for decryption function on level 2 ciphertexts of Boosted-CGS (Dec2_B): The followings are the MAGMA code we used to implement for the decryption algorithm for functions Mult₁, Add₂ and SMult₂. We implemented the decryption function on three particular problems for the above three operation. We name them as follows: decryption function for Mult₁ is denoted by DEC₂, decryption

function for **Add₂** is denoted by **DEC₃** and decryption function for **SMult₂** is denoted by **DEC₄**.

- Decryption function for **Mult₁** : This code is for a level 2 ciphertext in $\widehat{C} \times \widehat{C}^{2l}$ where $l = 1$. To recover the message, CGS decryption function **Dec_{CGS}** is used for decrypting α , β_1 and β_2 . Then, plaintexts corresponding to β_1 and β_2 are multiplied and added with α to decrypt to m .

```
DEC_2:= function(alpha,beta,G,a,p)
x:=beta[1];
y:=beta[2];
m:=Dec_CGS(alpha,G,a,p)+(Dec_CGS(x,G,a,p)*Dec_CGS(y,G,a,p));
return m;
end function;
```

- Decryption function for **Add₂** : This code is for a level 2 ciphertext in $\widehat{C} \times \widehat{C}^{2(l_1+l_2)}$ where $l_1 = 1$ and $l_2 = 1$. Plaintexts corresponding to column vectors are multiplied for each column and added together with the decryption of α to recover the message m .

```
DEC_3:= function(alpha,beta,G,a,p)
x1:=beta[1][1];x2:=beta[1][2];x3:=beta[1][3];x4:=beta[1][4];
y1:=beta[2][1];y2:=beta[2][2];y3:=beta[2][3];y4:=beta[2][4];
x:=[x1,x2];
y:=[x3,x4];
z:=[y1,y2];
w:=[y3,y4];
m:=Dec_CGS(alpha,G,a,p)+(Dec_CGS(x,G,a,p)*Dec_CGS(z,G,a,p)
+Dec_CGS(y,G,a,p)*Dec_CGS(w,G,a,p));
return m;
end function;
```


- Decryption function for SMult_2 : Again in our example, we implemented the function on a level 2 ciphertext in $\hat{C} \times \hat{C}^{2l}$ where $l = 1$. To recover the message, similar to the case for DEC_2 , CGS decryption function Dec_{CGS} is used for decrypting α , β_1 and β_2 . Here, the ciphertext $c = [\alpha, \beta]$ with $\beta = [\beta_1, \beta_2]$. Then, plaintexts corresponding to β_1 and β_2 are multiplied and added with α to decrypt to m .

```

DEC_4:= function(alpha,beta,G,a,p)
  x1:=beta[1][1];x2:=beta[1][2];
  y1:=beta[2][1];y2:=beta[2][2];
  x:=[x1,x2];
  y:=[y1,y2];
  m:=Dec_CGS(alpha,G,a,p)+(Dec_CGS(x,G,a,p)*Dec_CGS(y,G,a,p));
  return m;
end function;

```

5.3 SECURITY OF THE BOOSTED CGS ENCRYPTION SCHEME

As we mentioned in section 4.3 and described in the paper [4] by Catalano and Fiore, the semantic security as well as the circuit privacy of the boosted linearly homomorphic scheme depends on the underlying linearly homomorphic scheme, thus the semantic security and circuit privacy of boosted CGS scheme depends on the related features of CGS scheme.

5.4 EFFICIENCY OF THE BOOSTED CGS ENCRYPTION SCHEME

The efficiency of the boosted CGS scheme directly follows from the efficiency of boosted linearly homomorphic scheme we discuss in Section 4.4 and present in Table 4.1.

5.5 REMARKS

In this chapter, We implemented the boosted linearly homomorphic scheme based on the CGS linearly homomorphic scheme. As a result, the boosted CGS scheme allows performing one multiplication and any number of addition operations on ciphertexts. From Section 5.3 and 5.4, we see that this scheme is efficient and seems secure to be implemented.

CHAPTER 6

CONCLUSION

In this thesis, we studied partially homomorphic encryption schemes and an approach to enable the scheme for more complex computations. In particular, we have studied RSA scheme, ElGamal scheme, and CGS scheme. We have presented detailed discussion about these schemes in Chapter 2 and 3. We also discussed some applications of these schemes such as voting, banking, cloud computing etc. Craig Gentry introduced the idea of fully homomorphic encryption scheme in 2009 [8] that is theoretically perfect but not very practical for processing time and implementation complexity. These drawbacks intrigued the cryptographers to extend partially homomorphic encryption schemes to empower the schemes to be able to perform other homomorphic computations. We studied one efficient and secure such boosting technique by Catalano et al. [4] and implemented this technique based on the CGS scheme. This implementation enables linearly homomorphic CGS scheme to efficiently perform one multiplication as well as arbitrary number of additions over the ciphertexts. We verified the efficiency of the scheme with our MAGMA implementation. This boosted CGS scheme seems secure since the security of this scheme relies on the security of the underlying CGS scheme, which is believed to be secure.

Although the applications of homomorphic schemes are theoretically rewarding, these schemes have not yet reached the maturity to be called practical. There are scopes of great amount of research in this area and further installments in this sector can focus on developing more practical cryptosystems.

BIBLIOGRAPHY

- [1] Dan Boneh. The decision diffiehellman problem. *Lecture Notes in Computer Science*, 1423:48–63, 1998.
- [2] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Proceedings of the Second International Conference on Theory of Cryptography*, TCC’05, pages 325–341, Berlin, Heidelberg, 2005. Springer-Verlag.
- [3] Ernest F. Brickell and Yacov Yacobi. *On Privacy Homomorphisms (Extended Abstract)*, pages 117–125. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988.
- [4] Dario Catalano and Dario Fiore. Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS ’15, pages 1518–1529, New York, NY, USA, 2015. ACM.
- [5] CRYPTUTOR. Elgamal encryption scheme.
- [6] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
- [7] Taher ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31 (4):469472, 1985.
- [8] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.
- [9] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270 – 299, 1984.
- [10] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall & CRC, Taylor & Francis Group, LLC, Boca Raton, FL, 2008.
- [11] Werner Koch. Gnupg’s elgamal signing keys compromised, November 2003.
- [12] P. Rogaway M. Abdalla, M. Bellare. Dhaes, an encryption scheme based on the diffiehellman problem. Appendix A.
- [13] Pascal Paillier. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*, pages 223–238. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

- [14] A. Shamir R. Rivest and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):12Q–126, 1978.
- [15] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, Academia Press, pages 169–179, 1978.
- [16] Rosario Gennaro Ronald Cramer and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. *EUROCRYPT, Lecture notes in Computer Science*, Springer-Verlag, 1233:103–118, 1997.
- [17] Jaydip Sen. Homomorphic encryption: Theory and applications.
- [18] Smart and Nigel. Dr. clifford cocks cb, February 2008. [Retrived on August 14, 2011].
- [19] Karen t. Coe. Pgp: How it works and the mathematics behind it. *GIAC directory of certified professionals*, SANS Institute, GSEC Security Essentials version 1.4 b, 2002.