DEGREE PROJECT IN TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2023

Homomorphic Encryption

KTH Thesis Report

Daniel Sanaee

KTH ROYAL INSTITUTE OF TECHNOLOGY
MATHEMATICS

Authors

Daniel Sanaee <dsanaee@kth.se>
Mathematics
KTH Royal Institute of Technology

Place for Project

Stockholm, Sweden

Examiner and supervisor

Johan Håstad
KTH Royal Institute of Technology

Abstract

The problem of constructing a secure encryption scheme that allows for computation on encrypted data was an open problem for more than 30 years. In 2009, Craig Gentry solved the problem, constructing the first fully homomorphic encryption (FHE) scheme. The challenge of constructing a homomorphic encryption scheme can be divided into three main components: 1) Constructing a decryption algorithm that is a ring homomorphism. 2) Proving CPA security. 3) Managing noise growth of evaluated ciphertexts. This thesis presents a formal mathematical background to FHE and discusses these three components. To compute on ciphertexts, the decryption algorithm needs to be homomorpic with respect to multiplication and addition operation. This means, theoretically, computing and then decrypting is equivalent to decrypting and then computing. Security is proved by reductions and hardness assumptions. The standard hardness assumptions used today is the LWE and RLWE assumptions. All existing encryption schemes are based on introducing noise when encrypting. This noise grows with each ciphertext operation and without controlling noise, decryption fails given sufficiently many operations. Gentry showed that a scheme that can evaluate its own decryption circuit can be used to reduce the noise and bootstrapped into a fully homomorphic encryption scheme.

Abstrakt

Problemet med att konstruera ett säkert krypteringssystem som tillåter beräkning på krypterad data var ett öppet problem i över 30 år. År 2009 löste Craig Gentry problemet genom att konstruera det första fullt homomorfa krypteringssystemet (FHE). Utmaningen med att konstruera ett homomorft krypteringssystem kan delas upp i tre huvudsakliga komponenter: 1) Konstruktion av en dekrypteringsalgoritm som är en ringhomomorfism. 2) Bevisning av CPA-säkerhet. 3) Hantering av brusökning i utvärderade chiffertexter. Denna uppsats presenterar en formell matematisk bakgrund till FHE och diskuterar dessa tre komponenter. För att göra beräkningar på chiffertexter måste dekrypteringsalgoritmen vara homomorf med avseende på multiplikation och addition. Detta betyder att det, teoretiskt sett, är ekvivalent att beräkna och sedan dekryptera som med att dekryptera och sedan beräkna. Säkerheten bevisas genom reduktioner och svårhetsantaganden. De standardmässiga svårhetsantaganden som används idag är LWE och RLWE-antagandena. Alla befintliga krypteringssystem är baserade på att införa brus vid kryptering. Detta brus ökar vid varje chiffertext operation och i avsaknad av kontroll över bruset kommer dekryptering misslyckas efter tillräckligt många operationer. Gentry visade att ett krypteringssystem som kan utvärdera sin egen dekrypteringskrets kan användas för att minska bruset och omvandlas till ett fullt homomorft krypteringssystem.

Acknowledgements

I want to thank my supervisor Johan Håstad for his guidance and support throughout this project. I also want to thank my family and friends for their support and encouragement.

Contents

1. Introduction

In a recent press release, Gartner estimated cloud computing to reach a market size of about 600 billion dollars in the year of 2023 [Gar23]. It is clear that the need for delegated storage and delegated computing has grown steadily as data has become more valuable. From a security perspective, delegated computation is a challenging problem. Normally, the client encrypts the data on their secure local machine and sends it to the server. The server decrypts and performs computations on the raw, unencrypted data, then re-encrypts the result and sends it back to the client, who decrypts it. For this protocol to work, the server needs access to the secret key for decrypting the data, meaning having access to the raw data. However, this raises privacy concerns. What if the server is malicious and leaks the data for profit? What if the server is compromised? To protect the client, the computation should ideally not require any information about the raw data. More generally, is there a safe way to allow for any third party to process sensitive data? How does one compute on encrypted data without first decrypting it?

Homomorphic encryption is an encryption method that allows for computation on encrypted data. To compute on ciphertexts, homomorphic encryption schemes need to satisfy what is called a homomorphic property; an operation on ciphertexts in the encrypted domain corresponds to an operation on the plaintexts in the unencrypted domain. Some encryption schemes have one natural homomorphic property, such as the multiplication operation in the RSA cryptosystem. However, to allow for general computation on ciphertexts, the scheme in question need to satisfy two types of operations; multiplication and addition of ciphertexts. It turns out that any computable function can be represented as a combination of these two operations. The question of secure, arbitrary computation on encrypted data therefore boils down to the following quesiton; does there exist a secure encryption scheme that satisfies an arbitrary (possibly infinite) combination of addition and multiplication operations on ciphertexts?

A scheme that satisfies the above is called a fully homomorphic encryption scheme. The problem of finding one has been dubbed as the ”holy grail of cryptography” and remained an open problem for more than 30 years. In 2009, Craig Gentry [Gen09], solved this problem. Gentry showed the existence of a secure encryption scheme that satisfies arbitrary multiplication and addition of ciphertext. His construction was complicated and inefficient, but it launched the development of more efficient schemes. Many research grants have been provided towards the development of more efficient schemes. The applications areas of homomorphic encryption essentially includes everything that processes sensitive information. Some examples of areas where computation on encrypted data is particularly relevant are healthcare records, machine learning, genome sequencing, navigation, electronic voting and financial records.

2. Computation theory

In this section, it is assumed that the reader has prior knowledge of deterministic- and probabilistic Turing machines as a model of computation (an excellent introduction can be found in [Gol01]). We introduce an alternative model of computation based on sets of circuits for the purpose of protection against stronger adversaries. We include the basics needed to understand cryptography and homomorphic encryption in particular.

2.1 Digital logic

Definition 1 (Circuit)**.** For $n, m \in \mathbb{N}$ and any field $(\mathbb{F}, +, \times)$, an arithmetic circuit is a vector-valued polynomial function $C \colon \mathbb{F}^n \to \mathbb{F}^m$.

A circuit C is represented by a finite directed acyclic graph with n source nodes (the n inputs) and m sink nodes (the m outputs). The internal nodes of the circuit are called *gates* and are stacked in layers. For more details about the structure of a circuit, see [Gol08] or [MF21]. The number of nodes in C is called its *size* and is denoted $|C|$. The longest path in C is called its *depth.* A circuit is called *Boolean* when the field is $\mathbb{F}_2$ and each gate takes at most 2 inputs. Boolean circuits and arithmetic circuits are equivalent in the sense that the set of functions that can be computed by an arithmetic circuit is equal to the set of functions that can be computed by a Boolean circuit.[1] If the input is a string of bits, we assume the circuit is Boolean.

We consider circuits as algorithms and use them as an alternative approach to the traditional Turing machine model of computation.[2] Notice that any given circuit, C, can only compute on inputs of the same length whereas a Turing machine M takes inputs of any size n. However, a circuit always halts on a given input whereas a Turing machine may not. For the purpose of our discussion relating to cryptography, we assume every Turing machines halts unless otherwise stated. To allow circuits to handle arbitrary length inputs we consider families of circuits.

[1]A Boolean circuit is an arithmetic circuit. Conversely, any arithmetic circuit can be simulated by representing the inputs and outputs as a bitstring and utilizing that XOR and AND is a complete set of gates.

[2]The reason for this alternative model is to assume adversaries are computationally ”stronger”. See Theorem 1.

Definition 2 (Circuit family [MF21]). A circuit family $C = \{C_n\}_{n\in\mathbb{N}}$ is an indexed set of circuits $C_n \colon \mathbb{F}^{n+r} \to \mathbb{F}^m$ where $r, m = \text{poly}(n)$.

For any input x with length n, $C(x) \overset{\text{def}}{=} C_n(x)$. For each circuit $C_n \in C$, r represents the random coins used. If $r = 0$ for all n then C is a deterministic circuit family. A circuit family is said to have polynomial-size if there exists a polynomial p such that $|C_n| < p(n)$ for all n.

2.2 Complexity classes

Definition 3 (Complexity Class P). P is the set of languages $\mathscr{L}$ such that there exists a deterministic polynomial-time Turing machine M satisfying $M(x) = 1 \iff x \in \mathscr{L}$

Definition 4 (Complexity Class BPP). BPP is the set of languages $\mathscr{L}$ such that there exists a probabilistic polynomial-time (PPT) Turing machine M satisfying

$$\Pr[M(x) = 1] \geq 2/3 \text{ if } x \in L$$
$$\Pr[M(x) = 1] \leq 1/3 \text{ if } x \notin L$$

Definition 5 (Complexity Class P/poly). P/poly is the set of languages $\mathscr{L}$ such that there exists a polynomial-size circuit family C satisfying $C(x) = 1 \iff x \in \mathscr{L}$

Informally speaking, circuit families is a stronger model of computation than the PPT Turing machine model in the sense that if there exists a PPT Turing machine for deciding a problem, then there also exists a circuit family that can decide the same problem. The formal statement is as follows:

Theorem 1. P $\subseteq$ BPP $\subsetneq$ P/poly

The first inclusion follows from the fact that if there exists a deterministic Turing machine that decides a language, then that same machine can be seen as a PPT machine that ignores a given input sequence of coin tosses. For the second inclusion, consider a language $\mathscr{L} \in$ BPP and a corresponding PPT machine M for $\mathscr{L}$. Then, for any given input x_n with length n, at least 2/3 of the set of all possible coin toss sequences are good (good r means $M(x_n; r) = 1 \iff x_n \in \mathscr{L}$). This means that there exists at least 1 sequence of coin tosses that yields the correct result for 2/3 of the possible inputs of length n. Consider machine M' that on input x_n runs $M(x_n)$ many (still $\text{poly}(n)$) times and outputs the majority result. Then, the error probability vanishes exponentially, meaning there are exponentially few coin toss sequences that are bad for M' (see [AB09; Gol01] for more detail). Therefore there exist a coin toss sequence, r_n, that yields the correct result for all inputs of length n. Consider circuit $C_n \colon \{0,1\}^{n+|r_n|} \to \{0,1\}$ with r_n hardcoded as inputs where C_n simulates M' using r_n, i.e., $C_n(x_n) = M'(x;r)$. Therefore $C_n(x) = 1 \iff x \in \mathscr{L}$ and C decides $\mathscr{L}$.

Interestingly the first inclusion is speculated to be set equivalence [AB09, pp. 126], meaning that a deterministic machine could decide the same languages as a probabilistic one. The second inclusion is proper since every unary language is in P/poly whereas undecidable ones are not in BPP (see [AB09, pp. 110] for details). In this sense, circuit families are a stronger model of computation than PPT Turing machines. We capture this notion with uniformity.

Definition 6 (Uniform circuit family)**.** A circuit family $\{C_n\}_{n\in\mathbb{N}}$ is uniform if there exists a polynomial-time Turing machine M such that $M(1^n)$ outputs the description of C_n for all $n \in \mathbb{N}$.

A uniform circuit family is polynomial size. The converse is not necessarily true. A family that is not uniform is said to be a non-uniform circuit family. Note that Turing machines are at least as strong as uniform circuit families. More formally, if a uniform circuit family decides $\mathscr{L}$ then there exists a polynomial-time Turing machine that decides $\mathscr{L}$.[3] Simply construct the polynomial-time Turing machine that given any input $x \in \mathscr{L}$, first generates a description of $C_{|x|}$ and then simulates $C_{|x|}(x)$. In other words, the non-uniform circuit families are stronger than the polynomial-time Turing machines.

[3]The converse is also true, meaning deterministic polynomial-time Turing machines are exactly as powerful as uniform circuit families. See [AB09, pp. 111] for details

3. Mathematical foundations

3.1 Probability theory

Homomorphic encryption schemes are based on noise. As we will see in Subsection 5.1, noise can be sampled from discrete spaces in accordance with a distribution. Since distributions are central in cryptography, it is important they are understood. A distribution is a probability measure on a measurable space $(S, \mathcal{S})$. Typically, probability distributions are associated with random variables; however, in the absence of random variables, distributions are understood as a specified measure function on the given measurable space (See [Kal21, pp. 83]).

Definition 7 (Discrete distribution measure of a random variable)**.** Consider a probability space $(\Omega, \mathcal{F}, \Pr)$ and a discrete measurable space $(S, \mathcal{S})$. Let $X\colon \Omega \mapsto S$ be a random variable. The discrete distribution measure of X, or simply *distribution* of X, χ is defined as follows

$$\begin{aligned}\chi\colon \mathcal{S} &\to [0,1]\\ \{x\} &\mapsto \Pr[X = x]\end{aligned}$$

We say that χ is the distribution or *law* of X, denoted $\mathcal{L}(X)$.

Remark. Since χ is a measure on a discrete space, the description of the singletons are sufficient. More explicitly, $\chi(A) = \sum_{x\in A} \Pr[X = x]$ for $A \in \mathcal{S}$

A distribution measure for a random variable is a probability measure on the sample space $(S, \mathcal{S})$, as opposed to on the outcome space $(\Omega, \mathcal{F})$. For a given probability space, any random variable X defined on it has a distribution associated with it. We write $x \leftarrow X$ or $x \leftarrow \chi$ for sampling the outcome x from X assuming χ is its distribution. When X is a space we mean that x is uniformly sampled from X.

Definition 8 (Ensembles)**.** Let I be a countable index set. A *probability ensemble* indexed by I (or just ensemble) is a sequence of random variables $(X_i)_{i\in I}$. A *distribution ensemble* indexed by I is a sequence of distributions $(\chi_i)_{i\in I}$

In this paper we will exclusively use $\mathbb{N}$ as the index set. For example, the encryption function is a random variable (PPT algorithm), meaning that a single message m corresponds to many valid ciphertexts. By varying the security parameter of the scheme we construct the probability ensemble $\{Enc(pk, m)\}_{pk\in\mathbb{N}}$

Definition 9 (Statistical distance)**.** Let S_n be finite set for all $n \in \mathbb{N}$ and let $X = (X_n \colon \Omega_n \to S_n)_{n\in\mathbb{N}}$, $Y = (Y_n \colon \Omega_n \to S_n)_{n\in\mathbb{N}}$ be two ensembles. The *statistical distance* is defined as

$$\Delta_{X,Y}(n) \overset{\text{def}}{=} \frac{1}{2} \sum_{\alpha\in S_n} |\Pr[X_n = \alpha] - \Pr[Y_n = \alpha]|.$$

Let $\chi = \{\chi_n\}_{n\in\mathbb{N}}$, $\psi = \{\psi_n\}_{n\in\mathbb{N}}$ be two discrete distribution ensembles on the same measurable spaces $\{(S_n, \mathcal{S}_n)\}_{n\in\mathbb{N}}$. The statistical distance is defined as

$$\Delta_{\chi,\psi}(n) \overset{\text{def}}{=} \frac{1}{2} \sum_{\alpha\in S_n} |\chi_n(\{\alpha\}) - \psi_n(\{\alpha\})|$$

Remark. This definition for distribution ensembles assumes every singleton is measurable. A more general definition is $\Delta(\chi, \psi) \overset{\text{def}}{=} \sup_{A\in\mathcal{S}} |\chi(A) - \psi(A)|$

Definition 10 (Negligible function)**.** Let $f \colon \mathbb{N} \to [0, 1]$. f is negligible if for all positive polynomials $p(\cdot)$, there exists an N such that for all $n > N$, $f(n) < \frac{1}{p(n)}$. We say $f = \text{negl}(n)$.

Definition 11 (Perfectly indistinguishable)**.** Two ensembles (probability or distribution) X and Y are *perfectly indistinguishable* if $\Delta_{X,Y}(n) = 0$ for all n.

Definition 12 (Statistically indistinguishable)**.** Two ensembles (probability or distribution) X and Y are *statistically indistinguishable* if $\Delta_{X,Y}(n) = \text{negl}(n)$.

A desirable property of encryption schemes is to make it unfeasible for adversaries to distinguish encryptions. For two random variables (or distributions) X and Y, we want every adversary C to not be able to distinguish samples from X and Y. More precisely, an adversary tasked with identifying whether given samples are from X (C outputs 0) or from Y (C outputs 1) should have roughly the same probability of success irrespective of which the samples are generated from. The formal definition is as follows:

Definition 13 (Computationally indistinguishable [Gol01])**.** Two ensembles (or distribution ensembles) X and Y are computationally indistinguishable if, for every polynomial-size circuit family $C = \{C_n\}_{n\in\mathbb{N}}$,

$$\mathrm{Adv}_{X,Y}(n) \overset{\text{def}}{=} |\Pr[C(X_n) = 1] - \Pr[C(Y_n) = 1]| = \mathrm{negl}(n).$$

$C(X_n), C(Y_n)$ means that the adversary has access to random oracle returning a sample from X_n and Y_n respectively. The point is that, for every adversary C guessing the samples are from one of the random variables or distributions (1 in this case represents from Y_n), the probability of being correct is, up to negligible error, equal to the probability of being incorrect. For example, an adversary always guessing 1 has 100% probability of being incorrect when given samples from X_n and 100% probability of being correct when given samples from Y_n. The intuition is that the sample does not make a noticable difference on the guess and that the advantage in distinguishing samples goes to zero fast (faster than the inverse of all polynomials).

Perfect distinguishability implies statistical indistinguishability as 0 is negligible. Statistical indistinguishability implies computational indistinguishability. To see why, assume $\Delta_{X,Y}(n) = \mathrm{negl}(n)$ and consider any circuit family C. Since X_n and Y_n are defined on the same space we have

$$\begin{aligned}\mathrm{Adv}_{X,Y}(n) &= |\Pr[C(X_n) = 1] - \Pr[C(Y_n) = 1]| \\ &\leq \sum_{\alpha\in S_n} |\Pr[C(X_n = \alpha) = 1 \mid X_n = \alpha]\Pr[X_n = \alpha] \\ &\qquad - \Pr[C(Y_n = \alpha) = 1 \mid Y_n = \alpha]\Pr[Y_n = \alpha]| \\ &\leq \sum_{\alpha\in S_n} |\Pr[X_n = \alpha] - \Pr[Y_n = \alpha]| = 2\Delta_{X,Y}(n) = \mathrm{negl}(n)\end{aligned}$$

where the first inequality is due to the triangle inequality and the second is due to conditional probability being upper bounded by 1. Intuitively, there does not exist enough distinguishable outcomes to distinguish the ensembles even if there exists an adversary that always correctly recognize an outcome from X and Y respectively.

3.2 Distributions

When working over integer lattices, the errors need to be sampled from discrete distributions. Furthermore, the desire is to keep the errors small. In this section we introduce the discrete Gaussian distribution, which is the most common distribution used in lattice-based cryptography. We also introduce subgaussian distributions, which are used in the error analysis of the GSW scheme in Chapter 8.

3.3 Discrete Gaussian Distribution

To add discrete noise to ciphertexts, it is typical to use the discretized Gaussian distribution introduced by Micciancio and Regev [MR07]. Since this distribution is used so frequently, we give a brief presentation of it here. A continuous Gaussian distribution in $\mathbb{R}^n$ centered at $\vec{c}$ where each coordinate is independently sampled from normal distirbution with standard deviation σ, then the multivariate Gaussian distribution is simplified to $\rho_{\vec{c}}(\vec{x};\sigma) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2}\|\vec{x}-\vec{c}\|^2\right)$. By introducing the scale factor $s \overset{\text{def}}{=} \sqrt{2\pi}\sigma$ we get $\rho_{s,\vec{c}}(\vec{x}) = \frac{1}{s^n}\exp\left(-\frac{\pi}{s^2}\|\vec{x}-\vec{c}\|^2\right)$. The goal is to discretize the Gaussian over lattices, see Section 5, by restricting the domain to lattice points. For a given lattice $L \subset \mathbb{R}^n$ we define the normalization constant $\rho_{s,\vec{c}}(L) \overset{\text{def}}{=} \sum_{\vec{x}\in L} \rho_{s,\vec{c}}(\vec{x})$.

Definition 14 (Discrete Gaussian distribution)**.** Let $L \subset \mathbb{R}^n$ be a lattice. Then the discrete Gaussian distribution is defined as

$$\forall \vec{x} \in L\colon \quad \mathrm{D}_{L,s,\vec{c}}(\vec{x}) \overset{\text{def}}{=} \frac{\rho_{s,\vec{c}}(\vec{x})}{\rho_{s,\vec{c}}(L)}.$$

Definition 15 (β-bounded distribution)**.** Let χ be a distribution over the integers. χ is β-bounded if $x \leftarrow \chi \implies |x| < \beta$

Remark. It is also possible (and common) to define a β-bounded distribution such that $|x| \geq \beta$ with negligible probability.

Subgaussian distributions

In this section we introduce subgaussian distributions. We use subgaussian distributions for a tighter error analysis when discussing the GSW scheme in Chapter 8. Much of this section can be found in [AP13].

Definition 16 (Subgaussian distribution)**.** A distribution χ over $\mathbb{R}$ is subgaussian with parameter $K > 0$ (K not nesessarily constant) if for every non negative $t \in \mathbb{R}$.

$$\chi(\{x \in \mathbb{R} \text{ such that } |x| > t\}) \leq 2\exp\left(-\pi\frac{t^2}{K^2}\right)$$

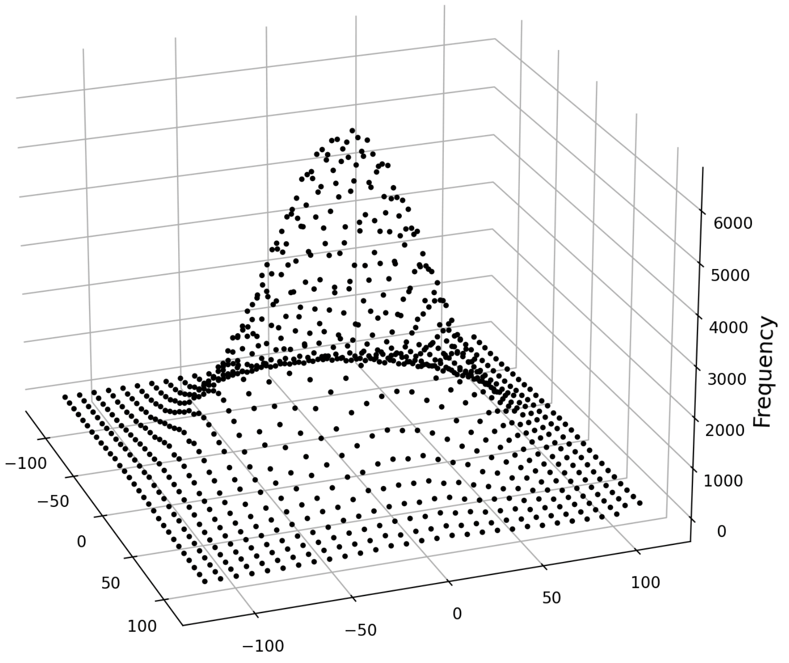

Figure 1: Discrete Gaussian distribution $D_{\mathbb{Z}^2,100,\vec{0}}$ with 1 million samples. Every point has integer coordinates. See code in Appendix A.

Or equivalently, a real valued random variable X has subgaussian distribution if

$$\Pr[|X| > t] \leq 2\exp\left(-\pi\frac{t^2}{K^2}\right)$$

We say X is a subgaussian random variable.

The intuition of subgaussian distributions is that the probability of a random variable X taking a large value is less than the probability of a Gaussian taking the same value. In other words, the tails of a subgaussian distribution decays at least as fast as the tails of a Gaussian distribution. In particular, the β-bounded distributions are subgaussian with parameter $\beta\sqrt{2\pi}$. To see why, clearly, for all $t \geq \beta$, $\Pr[|X| > t] = 0$ whereas the exponential function is non-negative. Furthermore, for $t < \beta$ we have $2\exp\left(-\pi\frac{t^2}{2\pi\beta^2}\right) > 2\exp\left(-\pi\frac{\beta^2}{2\pi\beta^2}\right) = 2\exp(-\frac{1}{2}) > 1 > \Pr[|X| > t]$. Hence, the definition is satisfied for all non-negative t. Notice that this means that a bounded discrete Gaussian is subgaussian. In particular, for a positive constant real value v, setting the bound $\beta = v\sigma$ on the discrete Gaussian means that the discrete Gaussian is subgaussian with parameter $v\sigma\sqrt{2\pi} = vs$ where s is the scale parameter.

Subgaussian variables are closed under scalar multiplication in the sense that if X is a subgaussian random variable with parameter K, then for any non-zero real number a, aX is subgaussian with parameter $|a|K$. This follows from $\Pr[|aX| > t] = \Pr[|X| >$

$\frac{t}{|a|}] \leq 2\exp\left(-\pi\frac{t^2}{(|a|K)^2}\right)$. Furthermore, it can be shown by using moment generating functions that a finite sum of n subgaussian random variables with parameters $K_1, \ldots, K_n$ is subgaussian with parameter $\sqrt{\sum_{i=1}^n K_i^2}$.

We extend the definition of subgaussian random variables to vectors of subgaussian random variables. Let $u \in \mathbb{R}^n$ be any fixed unit vector and let $\vec{X} = (X_1, \ldots, X_n)$ be a vector of random variables. We say $\vec{X}$ is subgaussian with parameter K if $\left\langle u, \vec{X} \right\rangle$ is subgaussian with parameter K. The intuition is that for any direction, the probability of sampling points far away from origin decays at least as fast as a Gaussian. In this paper, the focus is on random vectors of independent subgaussian entries. For such vectors, the vector is subgaussian if and only if all its elements are subgaussian. More specifically, they satisfy the following useful lemma.

Lemma 1 (Proposition 5.10 [Ver11]). Let $\vec{X} = (X_1, \ldots, X_n)$ be a vector of independent subgaussian random variables with parameters $K_1, \ldots, K_n$. Then $\vec{X}$ is subgaussian with parameter $\max_{i \in [n]} K_i$. Furthermore, for any vector $\vec{e} = (e_1, \ldots, e_n) \in \mathbb{R}^n$, the inner product $\left\langle \vec{e}, \vec{X} \right\rangle$ is subgaussian with parameter $O(K\|\vec{e}\|)$ where $K = \max_{i \in [n]} K_i$.

By considering each element individually, it is clear that multiplication with non-negative scalar and addition of vectors preserves subgaussianity in the same way. In other words, subgaussianity satisfies the following important lemma

Lemma 2 (Additivity and homogenity). Let $x_1, \ldots, x_n$ be any subgaussian random variables over the integers with parameters $k_1, \ldots, k_n$. Let $\vec{X}_1, \ldots, \vec{X}_m$ be any discrete subgaussian random vectors over the integers with parameters $K_1, \ldots, K_m$ and let a be a positive real number. Then

1. $\sum_{i=1}^n x_i$ is subgaussian with parameter $\|k\|$ where $k = (k_1, \ldots, k_n) \in \mathbb{Z}^n$
2. ax_i is subgaussian with parameter ak_i for all i.
3. $\sum_{i=1}^m \vec{X}_i$ is subgaussian with parameter $\|K\|$ where $K = (K_1, \ldots, K_m) \in \mathbb{Z}^m$
4. $a\vec{X}_i$ is subgaussian with parameter aK_i for all i.

We will use the previous two lemmas extensively throughout our analysis of the GSW scheme in Chapter 8. As a last point, we introduce a bound on the euclidean norm of a subgaussian vector.

Theorem 2 (Lemma 2.1 [AP13]). *Let $\vec{X}$ be an n dimensional subgaussian vector with* *parameter K, containing mutually independent entries. Then there exists real positive constant C such that*

$$\Pr[\|\vec{X}\| > CK\sqrt{n}] \leq 2^{-\Omega(n)}$$

In particular, the length $\|\vec{X}\|$ is $O(K\sqrt{n})$ except with negligible probability.

3.4 Embedding integers as permutations

In this section we show how to embed integers in a finite group as a tuple of cyclic shifts by showing the isomorphism $\mathbb{Z}_q \cong C_{r_1} \times \cdots \times C_{r_t}$ where C_i is the group of cyclic shifts and $q = \prod_{i=1}^{t} r_i$ for pairwise coprime r_i. Representing integers in this way will allow us to represent a standard q-modular additions of two integers as t compositions of permutation matrices. This is useful for the error analysis in the implementation of the GSW scheme, see Chapter 8. The isomorphism is established through the Chinese Remainder Theorem and Cayley's theorem.

Theorem 3 (Chinese Remainder Theorem)**.** *Let $r_1, \ldots, r_t$ be pairwise coprime integers and $q = \prod_{i=1}^{t} r_i$. The system of congruences*

$$\begin{aligned} x &\equiv a_1 \pmod{r_1} \\ &\vdots \\ x &\equiv a_t \pmod{r_t} \end{aligned}$$

has a unique solution $x \in \mathbb{Z}_q$. We say $(a_1, \ldots, a_t)$ is the RNS representation of x with respect to the moduli set $\{r_1, \ldots, r_t\}$.

The Chinese Remainder Theorem allows us to construct an isomorphism from $\mathbb{Z}_q$ to $\mathbb{Z}_{r_1} \times \cdots \times \mathbb{Z}_{r_t}$.

Definition 17 (CRT isomorphism)**.** We define the *CRT isomorphism* $\mathrm{CRT}\colon \mathbb{Z}_q \to \mathbb{Z}_{r_1} \times \cdots \times \mathbb{Z}_{r_t}$ as a mapping of x to its RNS representation $(a_1, \ldots, a_t)$ with respect to the moduli set $\{r_1, \ldots, r_t\}$.

Remark. To compute the CRT isomorphism on a given input x we simply compute $x \pmod{r_i}$ for each i (we will not need the inverse mapping but it can be computed using the Extended Euclidean Algorithm).

Important to note is that addition of integers in $\mathbb{Z}_q$ is equivalent to element-wise addition in $\mathbb{Z}_{r_1} \times \cdots \times \mathbb{Z}_{r_t}$. In other words, for $x, y \in \mathbb{Z}_q$ we have $x + y \equiv a_1 + b_1 \pmod{r_1}$, ...,

$x + y \equiv a_t + b_t \pmod{r_t}$ where $(a_1, \ldots, a_t)$ and $(b_1, \ldots, b_t)$ are the RNS representations of x and y respectively. We can therefore represent addition of integers in $\mathbb{Z}_q$ as addition of integers in $\mathbb{Z}_{r_1} \times \cdots \times \mathbb{Z}_{r_t}$ by computing the RNS representation of the sum. This is useful since addition of integers in $\mathbb{Z}_{r_i}$ can be represented as compositions of cyclic shifts in S_{r_i}, as discussed below. Now we show how to embed the finite additive group $\mathbb{Z}_r$ into the symmetric group S_r. The symmetric group S_r is the group of all permutations of the set $\{1, \ldots, q\}$. A standard way to represent a permutation $\pi \colon \{(x_1, \ldots, x_r) \mid x_a \neq x_b,\ a, b \in [r]\} \to \{(x_1, \ldots, x_r) \mid x_a \neq x_b,\ a, b \in [r]\}$ in S_r is to list where each entry is mapped to; $(x_1, \ldots, x_r) \overset{\pi}{\mapsto} (\pi(x_1), \ldots, \pi(x_r))$. For each seperate element we write $\pi(j) = i$ to mean that the jth entry is mapped to the ith entry (which in turn must be mapped somewhere else). In this paper we will also use a binary representation by using a square permutation matrix $\mathbf{P}^\pi = (p^\pi_{i,j}) \in \{0,1\}^{r \times r}$ such that $p^\pi_{i,j} = 1$ if and only if $\pi(j) = i$, or equivalently $\pi^{-1}(i) = j$. In particular, column j represents the mapping of element j. Since every entry is mapped to exactly one other entry, every column has exactly 1 non-zero entry. Since a permutations is a bijection, surjectivity implies every row has a non-zero entry. Therefore, every row and every column has exactly 1 non-zero element. $\mathbf{P}^\pi$ contains exactly r entries equal to 1 and $r^2 - r$ entries equal to 0. The matrix representation is inefficient but useful for understanding composition of permutations. As an example, consider a cyclic shift permutation $\pi \in S_3$ defined as $(\pi(x_1), \pi(x_2), \pi(x_3)) = (x_2, x_3, x_1)$. Then the corresponding permutation matrix $\mathbf{P}^\pi$ is

$$\mathbf{P}^\pi = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

The group operation of S_r is simple matrix multiplication for a total of r^3 arithmetic operations. A particular permutation group of interest with more efficient implementation of the group operation is the cyclic shift group C_r. C_r is the subgroup of the symmetric group S_r consisting of permutations shifting the input sequence by a fixed number of positions. Every element of this group can be represented by only the first column of the permutation matrix since the shift of the first element in the sequence must be the same as the shift for every other element. In other words, every element of C_r can also be represented by a r dimensional indicator vector $\pi \in \{0,1\}^r$. Our above example of a cyclic shift of 1 position to the right can therefore be represented by the indicator vector $\pi = (0, 1, 0)^T$ or by $\pi(i) = i + 1 \pmod 3$ for $i = 1, 2, 3$.

Theorem 4 (Cayley's theorem)**.** *Every group G is isomorphic to a subgroup of the symmetric group $S_{|G|}$.*

In particular, the additive group $\mathbb{Z}_r$ is isomorphic to C_r through the following natural embedding isomorphism.

Definition 18 (Embedding isomorphism)**.** We define the *embedding isomorphism* $\mathrm{EMB}\colon \mathbb{Z}_r \to C_r$ as a mapping of x to the indicator vector $(b_0, \dots, b_x, \dots, b_{r-1})$ where $b_i = 1$ if and only if $i = x$.

Definition 19 (Canonical homomorphism)**.** We define the *canonical homomorphism* $\phi\colon \mathbb{Z}_q \to C_{r_1} \times \cdots \times C_{r_t}$ as $\phi(x) \overset{\text{def}}{=} \mathrm{EMB} \circ \mathrm{CRT}(x)$.
The coordinate function is defined $\phi_i(x) = \pi_i$, $i \in [t]$, where π_i is the ith cyclic shift in $C_{r_1} \times \cdots \times C_{r_t}$.

ϕ is an isomorphism mapping an integer x to a tuple of cyclic shifts. In other words, there is a one-to-one correspondence between integers in $\mathbb{Z}_q$ and tuples of cyclic shifts in $S_{r_1} \times \cdots \times S_{r_t}$ and thus we write $\phi(x) \sim x$.

Consider the following example showing how to represent an element in $\mathbb{Z}_q$ into $S_{r_1} \times \cdots \times S_{r_t}$. Say we want to represent the integer $7 \in \mathbb{Z}_{15}$ as a tuple of cyclic shifts with respect to the coprime modulo set $\{3, 5\}$. In other words, we want to calculate $\phi(7)$. Then $t = 2$, $r_1 = 3$ and $r_2 = 5$. We have $x \pmod 3 = 1$ and $x \pmod 5 = 2$. The RNS representation of 7 is therefore $(1, 2)$. Applying the embedding isomorphism on each element of the RNS representation gives $1 \mapsto (0, 1, 0)^T$ and $2 \mapsto (0, 0, 1, 0, 0)^T$. Therefore, $\phi_1(7) = (0, 1, 0)^T$, $\phi_2(7) = (0, 0, 1, 0, 0)^T$ and

$$\phi(7) = (\phi_1(7), \phi_2(7)) = (\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix})$$

Addition as composition of cyclic shifts

To perform composition $\pi \circ \sigma$ of cyclic shifts we let π be the square representation and σ be the indicator vector representation and do standard matrix multiplication since this represents where the first element is mapped to under σ and then π. In this way, composition of cyclic shifts requires r^2 operations instead of the r^3 operations required for an arbitrary permutation in S_r.[1] Note that shifting with x positions followed by shifting with y positions is the same as shifting with $x + y$ positions. In general, compositions of cyclic shifts can be used to calculate modular addition of integers by using the structure preserving property of the isomorphism ϕ. Specifically, for $x, y \in \mathbb{Z}_q$, $\phi(x + y) = (\phi_1(x) \circ \phi_1(y), \dots, \phi_t(x) \circ \phi_t(y))$. This follows from applying the Chinese Remainder Theorem to the sum $x + y$ and observing that the remainder for any moduli is the sum of the remainders of the terms. By induction, the sum of n integers $x_1, \dots, x_n \in \mathbb{Z}_q$ using cyclic shifts is

$$\sum_{j=1}^{n} x_i \sim \phi(\sum_{j=1}^{n} x_i) = (\bigcirc_{j=1}^{n} \phi_1(x_j), \dots, \bigcirc_{j=1}^{n} \phi_t(x_j)) \in S_{r_1} \times \cdots \times S_{r_t} \tag{3.1}$$

To illustrate this point, consider the sum $7 + 11$ in $\mathbb{Z}_{15}$ using the modulo set $\{3, 5\}$. We have that $\phi_1(7) = (0, 1, 0)^T$, $\phi_2(7) = (0, 0, 1, 0, 0)^T$ and $\phi_1(11) = (0, 0, 1)^T$, $\phi_2(11) = (0, 1, 0, 0, 0)^T$. Then,

$$\phi_1(7) \circ \phi_1(11) = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Similarly,

$$\phi_2(7) \circ \phi_2(11) = (0, 0, 0, 1, 0)^T.$$

Therefore, $\phi(7 + 11) = ((1, 0, 0)^T, (0, 0, 0, 1, 0)^T) = \phi(3)$. By injectivity, $7 + 11 \equiv 3 \pmod{15}$

Efficient representation of integers as cyclic shifts

As it stands, representing integers as cyclic shifts can be inefficient. For instance, for a prime q, an element in $\mathbb{Z}_q$ is represented in binary using $\log q$ bits whereas the corresponding cyclic shift is represented with an indicator vector using q bits, thus yielding an exponential expansion in the representation size. For cyclic shift representation to be efficient, we want to choose modulus q yielding $\tilde{O}(1)$ bit size representation. In other words, the goal is to guarantee choise of q such that representing

[1]Composition can be computed in linear time if the elements of σ and π are unencrypted by simply observing where the non-zero entry is located for each cyclic shift

an integer using a tuple of cyclic shifts requires $O(\text{polylog(q)})$ bits. To do so, we want to find $q = \prod_{i=1}^{t} r_i$ such that $\max_{i\in[t]} r_i = O(\log q)$ and $t = O(\frac{\log q}{\log\log q})$. This would yield that every $x \in S_{r_1} \times \cdots \times S_{r_t}$ can be represented in $O(\frac{\log^2 q}{\log\log q}) = \tilde{O}(1)$. We will now show that such a modulus q exists and how to find it.

Definition 20. The maximal prime powers bounded by an positive integer r is defined as the set

$$\text{MPP}(r) \overset{\text{def}}{=} \{p^{\lfloor \log_p r \rfloor} \mid p \leq r, p \text{ is prime}\}.$$

An element in $\text{MPP}(r)$ is called a maximal prime power.

For example, the maximal prime powers bounded by 10 is $\text{MPP}(10) = \{2^3, 3^2, 5^1, 7^1\} = \{8, 9, 5, 7\}$.

Lemma 3 (Lemma 2.2 [AP13]). Let $r \geq 7$ be an integer. Then

$$\prod_{r_i \in \text{MPP}(r)} r_i \geq \exp\left(\frac{3r}{4}\right).$$

The idea is to choose q as the product of maximal prime powers bounded by some $r = O(\log q)$. In other words, we consider $t = |\text{MPP}(r)|$ and $\{r_1, \ldots, r_t\} = \text{MPP}(r)$. We can efficiently find such q by first determining an acceptable lower bound $q_0 \leq q$. Since $r \geq 7$, we require $q_0 \geq \exp(\frac{3\times 7}{4}) > 190$. The second step is to find an r such that the product of its maximal prime powers is greater than or equal to our lower bound q_0. Choosing $r = \lceil \frac{4}{3}\log q_0 \rceil$ works since $q = \prod_{r_i \in \text{MPP}(r)} r_i$ is, by lemma 3, at least $\exp\left(\frac{3r}{4}\right) = \exp\left(\frac{3}{4}\lceil\frac{4}{3}\log q_0\rceil\right) \geq \exp\left(\log q_0\right) = q_0$.

Note that r is logarithmic with respect to q, implying $\max_{i\in[t]} r_i = O(\log q)$. We now show that $t = O(\frac{\log q}{\log\log q})$.

Lemma 4 (Prime number theorem). Let $f(x)$ be the number of primes less than or equal to the real value x. Then

$$\lim_{x\to\infty} \frac{f(x)}{x/\ln(x)} = 1$$

Note that prime number theorem gives that the number of primes bounded by an integer x is $O(x/\log(x))$. In particular, $\text{MPP}(r)$ contains exactly one element per prime bounded by r, meaning $t = O(\frac{r}{\log r}) = O(\frac{\log q}{\log\log q})$.

Theorem 5. *Let* $r = \log(\lambda)$ *and* $q = \prod_{r_i \in MPP(r)} r_i$. *Then* $q = \Theta(\lambda)$

Proof. By lemma 3, $q = \Omega(\lambda)$. We need to show $q = O(\lambda)$. Since r is an upper bound on $\mathrm{MPP}(r)$ and t is its cardinality, Lemma 4 gives $q = O(r^t) = O(\log(\lambda)^{\frac{\log\lambda}{\log\log\lambda}})$. Define y as $\log(\lambda)^{\frac{\log\lambda}{\log\log\lambda}}$. Then $\log(y) = \frac{\log\lambda}{\log\log\lambda}\log\log\lambda = \log\lambda$ and so $y = \lambda$. Thus, $q = O(\lambda)$. $\square$

As an example, say we want a modulus $q \geq 1000$. Then $r = \lceil \frac{4}{3}\log 1000 \rceil = 14$. We have $q = \prod_{r_i \in \mathrm{MPP}(14)} r_i = 2^3 \times 3^2 \times 5^1 \times 7^1 \times 11^1 \times 13^1 = 360360$. We can now represent an element $x \in \mathbb{Z}_{360360}$ as a tuple of cyclic shifts in $S_8 \times S_9 \times S_5 \times S_7 \times S_{11} \times S_{13}$. Note that $t = |\mathrm{MPP}(14)| = 6 = O(\frac{\log q}{\log\log q})$ and $\max_{i\in[t]} r_i = \max\{\mathrm{MPP}(14)\} = 13 = O(\log q)$.

4. Cryptographic primitives

In the following definitions we let the message space be denoted $\mathcal{X}$, the ciphertext space be denoted $\mathcal{Y}$, and the key space be denoted $\mathcal{K} = \mathcal{K}_{pk} \times \mathcal{K}_{sk}$.

Definition 21 (Encryption scheme)**.** A correct asymmetric *encryption scheme* $\mathcal{E} = (\text{KeyGen}, \text{Enc}, \text{Dec})$ is a triple of algorithms satisfying the following:

- KeyGen : $\{1\}^* \to \mathcal{K}$ is PPT given by $1^\lambda \mapsto (pk, sk)$.
- Enc : $\mathcal{K}_{pk} \times \mathcal{X} \to \mathcal{Y}$ is PPT given by $(pk, m) \mapsto c$.
- Dec : $\mathcal{K}_{sk} \times \mathcal{Y} \to \mathcal{X}$ is deterministic and satisfies $(pk, sk) \leftarrow \text{KeyGen}(1^\lambda) \implies \text{Dec}(sk, \text{Enc}(pk, m)) = m$.

Remark. We also allow the decryption function to decrypt incorrectly with negligible probability. Furthermore, the decryption algorithm can be considered a PPT algorithm that with probability 1 outputs the correct message for the correct secret key. In other words, the algorithm ignores the input coin toss sequence.

In this paper, we consider homomorphic encryption (HE) schemes. These schemes include a fourth algorithm, Eval, called the evaluation algorithm which is used by the server calculating on encrypted data.

Definition 22 ($\mathcal{C}$-homomorphic encryption scheme)**.** An encryption scheme $\mathcal{E}$ is a $\mathcal{C}$-*homomorphic encryption scheme* for the set of circuits $\mathcal{C}$ if there exists an extra algorithm Eval such that for any $C \in \mathcal{C}$ taking t inputs the following condition holds:

- Eval: $\mathcal{K}_{pk} \times \mathcal{C} \times \mathcal{Y}^* \to \mathcal{Y}$ is PPT and satisfies $(pk, sk) \leftarrow \text{KeyGen}(1^\lambda) \implies \text{Dec}(sk, \text{Eval}(pk, C, \langle \text{Enc}(pk, m_1), \ldots, \text{Enc}(pk, m_t)\rangle)) = C(m_1, \ldots, m_t)$

We say $\mathcal{E}$ can evaluate all circuits in $\mathcal{C}$ and is $\mathcal{C}$-homomorphic.

The evaluation algorithm runs the ciphertexts through the permissible circuit while also satisfying the requirement that decrypting the resulting ciphertext yields the same result as the plaintexts running through the circuit. To its disposal, the evaluation algorithm is given the public key. The ciphertexts returned by the Eval algorithm are called *evaluated*

$$\begin{tikzcd}
\mathcal{Y}^t \arrow[r, "\mathrm{Eval}(pk{,}C{,}\langle c_1{,}\ldots{,}c_t\rangle) = c_f"] \arrow[d, "\mathrm{Dec}(sk{,}\langle c_1{,}\ldots{,}c_t\rangle)"'] & \mathcal{Y} \arrow[d, "\mathrm{Dec}(sk{,}c_f)"] \\
\mathcal{X}^t \arrow[r, "C(m_1{,}\ldots{,}m_t)"'] & \mathcal{X}
\end{tikzcd}$$

Figure 2: The decryption homomorphism. The path through $\mathcal{Y}$ represents computing before decrypting. The path through $\mathcal{X}^t$ represents decrypting before computing.

ciphertexts (suggesting the circuit has evaluated the ciphertexts) and those returned by the encryption algorithm are called *fresh ciphertexts.* Remark that correctness is only guaranteed if the Eval algorithm is given fresh ciphertexts. If the circuit corresponds to the computable function f acting on a vector c of ciphertexts, we denote the evaluated ciphertexts c_f (i.e., $c_f \overset{\text{def}}{=} \mathrm{Eval}(pk, f, c)$). Similarly, for the vector m of plaintexts, we denote the evaluated plaintexts m_f (i.e., $m_f \overset{\text{def}}{=} f(m)$). Thus, the condition for the Eval algorithm can be written $(pk, sk) \leftarrow \mathrm{KeyGen}(1^\lambda) \implies \mathrm{Dec}(sk, c_f) = m_f$. In a homomorphic encryption scheme that supports one addition and multiplication of fresh ciphertexts, the decryption function is a ring homomorphism. Consider a valid key pair (sk, pk) which are, for notational simplicity, hard-wired into the decryption and evaluate functions respectively, plaintext ciphertext pairs $(m_1, c_1 = \mathrm{Enc}_{pk}(m_1))$, $(m_2, c_2 = \mathrm{Enc}_{pk}(m_2))$ and circuits $C_+(m_1, m_2) \overset{\text{def}}{=} m_1 + m_2$ and $C_\times(m_1, m_2) \overset{\text{def}}{=} m_1 \times m_2$ that can be evaluated by the scheme.

$$\begin{aligned}
\mathrm{Dec}_{sk}(c_1 + c_2) &= \mathrm{Dec}_{sk}(\mathrm{Eval}_{pk}(C_+, \langle c_1, c_2\rangle)) = C_+(m_1, m_2) = m_1 + m_2 \\
&= \mathrm{Dec}_{sk}(c_1) + \mathrm{Dec}_{sk}(c_2) \\
\mathrm{Dec}_{sk}(c_1 \times c_2) &= \mathrm{Dec}_{sk}(\mathrm{Eval}_{pk}(C_\times, \langle c_1, c_2\rangle)) = C_\times(m_1, m_2) = m_1 \times m_2 \\
&= \mathrm{Dec}_{sk}(c_1) \times \mathrm{Dec}_{sk}(c_2)
\end{aligned}$$

The main idea behind a homomorphic encryption scheme is to give a server encrypted data so that it can compute on that data and return the answer in encrypted form. However, the definition provided allows for trivial homomorphic encryption schemes where the server does nothing. More specifically, consider any set of circuits $\mathcal{C}$ and let $\mathrm{Eval}(pk, C, \langle c_1, \ldots, c_t\rangle) = (C, \langle c_1, \ldots, c_t\rangle)$. Eval takes a description of a circuit and a tuple of ciphertexts, one for each input wire of the circuit, and simply outputs the description of the circuit together with the given tuple. Clearly, Eval runs in polynomial time. Consider a decryption algorithm that, if given an input of this form, first decrypts the t ciphertexts and then computes m_f using C. To ensure that the server actually processes the given inputs we introduce compactness.

Definition 23 (Compactness)**.** A $\mathcal{C}$-homomorphic encryption scheme is compact if there exists a polynomial $p(\lambda)$ such that for all $(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)$, for every $C \in \mathcal{C}$ taking any number t inputs and any $c \in \mathcal{Y}^t$, the size of the output $\text{Eval}(pk, C, \langle c_1, \ldots, c_t \rangle)$ is less than $p(\lambda)$. We say that the scheme compactly evaluates $\mathcal{C}$.

For a compact $\mathcal{C}$-homomorphic encryption scheme, the size of the output is independent of the circuit function used. In particular, the previous, trivial homomorphic encryption scheme where $\text{Eval}(pk, C, \langle c_1, \ldots, c_t \rangle) = (C, \langle c_1, \ldots, c_t \rangle)$ is not compact for any set of circuits with unbounded circuit size, which includes circuit families with circuits that do not ignore all except for constantly many inputs, meaning essentially every application of a HE scheme.

Definition 24 (Fully Homomorphic Encryption (pure FHE))**.** Let $\mathcal{C}$ be the class of all circuits. An encryption scheme $\mathcal{E}$ is a fully homomorphic encryption (pure FHE) scheme if it is $\mathcal{C}$-homomorphic and compactly evaluates $\mathcal{C}$.

For a scheme to be fully homomorphic it is required that it can evaluate circuits of arbitrary size. Many times it suffices to consider only circuits of a beforehand specified depth, L, as any deeper circuits are irrelevant to the application. The following definition capture schemes that can evaluate any set of circuits with depths bounded by the client.

Definition 25 (Leveled fully homomorphic encryption (leveled FHE))**.** An encryption scheme $\mathcal{E}$ with the KeyGen algorithm modified is a leveled fully homomorphic encryption scheme if it satisfies the following:

- KeyGen : $\{1\}^* \times \{1\}^* \to \mathcal{K}$ is PPT given by $(1^\lambda, 1^L) \mapsto (pk, sk)$.
- Let $\mathcal{C}_L$ be the set of circuits with depth less than or equal to L. Then $\mathcal{E}$ is $\mathcal{C}_L$-homomorphic.
- $\mathcal{E}$ compactly evaluates the set of all circuits.

Remark. Notice that the length of the evaluated ciphertexts in a leveled FHE scheme is independent of the depth.

For any specified circuit C, a leveled FHE scheme can evaluate it by choosing sufficiently large depth parameter, L. For a pure FHE scheme, the circuit does not need to be specified. A pure FHE scheme can dynamically compute any circuit whereas the leveled FHE scheme requires the circuit chosen a priori.

4.1 Security definitions

In this paper, semantic security refers to security against chosen-plaintext attack (CPA). The definition relates to the following game where the challenger possess the secret key and the player is the adversary trying to break the scheme. Consider encryption scheme (KeyGen, Enc, Dec, Eval) and polynomial-size Boolean circuit family $C = \{C_n\}_{n\in\mathbb{N}}$. The CPA game is defined with the Boolean function $\text{CPA}_C(\lambda)$ as follows:

1. **Setup**: Challenger samples $pk \leftarrow \text{KeyGen}$ and sends it to player.
2. **Choose**: Player C selects two distinct plaintext messages $(m_0, m_1) \leftarrow C(pk)$ of the same length, and sends them to the challenger.
3. **Encrypt**: The challenger randomly picks a bit $b \in \{0, 1\}$ and encrypts the message m_b. The encrypted message $c \overset{\text{def}}{=} \text{Enc}(pk, m_b)$, called challenge ciphertext, is sent to the player.
4. **Guess**: Player C output guess $b' \in \{0, 1\}$.
5. **Win**: $\text{CPA}_C(\lambda) = \begin{cases} 1 & \text{if } b = b' \\ 0 & \text{if } b \neq b'. \end{cases}$

If $\text{CPA}_C(\lambda) = 1$ then the adversary C guessed correctly which of their two chosen messages was encrypted, based only on observing the ciphertext. Notice that the game requires the player to choose messages of equal length in the 'choose' phase since the ciphertext length always leaks information about the length of the message, namely an upper bound on the message length.

Definition 26 (Semantic security (CPA))**.** An encryption scheme is semantically secure if, for all polynomial-size Boolean circuit families C,

$$|\Pr[\text{CPA}_C(\lambda)] - \frac{1}{2}| = \text{negl}(\lambda).$$

Semantic security means that there exists no algorithm in P/poly that can do more than negligibly better than guessing randomly in determining the message. Semantic security is equivalent to indistinguishability of encryptions (see [Gol04] for proof).

Definition 27 (Indistinguishability of encryptions)**.** An encryption scheme has indistinguishable encryptions if for any key $(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)$ and any two distinct messages m_1, m_2 of equal length, the ensembles $\{\text{Enc}(pk, m_1)\}_{\lambda\in\mathbb{N}}$ and $\{\text{Enc}(pk, m_2)\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable.

Usually, encryption schemes are required to be secure against a stronger type of attack, called chosen-ciphertext attack (CCA). There are two types of CCA attacks; adaptive (called CCA2) and non-adaptive (called CCA1). The CCA1 game is defined exactly like the CPA game but where the player also has oracle access to the decryption algorithm in the choose phase. In other words, the player can decrypt any ciphertexts of their choice before submitting the two messages m_0 and m_1 to the challenger. The CCA2 game is the same as CCA1 except that the player also has oracle access to the decryption algorithm in the guess phase for every ciphertext except the challenge ciphertext. Security against CCA1 and CCA2 attacks are defined analogously to semantic security. Clearly, CCA2 security implies CCA1 security and CCA1 security implies semantic security.

As a consequence of its design, homomorphic encryption schemes cannot be CCA2 secure. The reason is that the player can run the evaluate algorithm on the challenge ciphertext with any circuit of choice and then decrypt the evaluated ciphertext. More formally, consider any challenge ciphertext $\text{Enc}(pk, m_b)$ and the permissable circuit C. Player runs $\text{Eval}(pk, C, \text{Enc}(pk, m_b))$, generating a valid evaluated ciphertext of $C(m_b)$. Player then queries decryption and yields $C(m_b)$. Since C is known to the attacker, information about m_b is leaked. Homomorphic encryption schemes allow the attacker to transform the ciphertext of a message m to a ciphertext of a message related to m by a known function. This property is called *malleability.*

One last security definition that will be relevant later is circular security

Definition 28 (Circular security)**.** A semantically secure homomorphic encryption scheme is circular secure if it is semantically secure even when the adversary is given encryptions of the secret key.

Remark. Circular security is not implied by semantic security because an adversary with a random access oracle cannot efficiently query encryptions of the secret key [Bra18].

5. Hard problems

In this section, we assume every vector is a column vector and we denote a matrix $\mathbf{A}$ with boldface. By $\mathbf{A} \in \mathbb{F}^{n\times m}$ we mean $\mathbf{A}$ is a $n \times m$ matrix with entries in $\mathbb{F}$. Consider a basis $\mathbf{B} = \{b_1, \dots, b_k\}$. A lattice with basis $\mathbf{B}$ is defined $L(\mathbf{B}) \overset{\text{def}}{=} \{\sum_{i=1}^{k} a_i b_i \mid a_i \in \mathbb{Z}\}$.

5.1 LWE

In 2005, Oded Regev introduced [Reg05] a natural problem; solve a system of modular noisy linear equations. The problem is called learning with errors (LWE) and in 2012, a ring based version called ring-LWE (RLWE) was introduced [LPR12]. Despite being easy to state, the LWE problem turns out to be hard even on average instances. For certain parameter choices, the LWE problem is reducible to extensively studied hard lattice based problems (e.g., GapSVP, SIVP) whereas RLWE is reducible to hard problems on ideal lattices (e.g., ideal-SVP, NTRU). Hardness of these problems are mostly outside the scope of this thesis but we will revisit some basics when arguing security in Chapter 8; we refer the curious reader to [Reg05; Pei08; Bra+13; LPR12] for more details on hardness results and [Pei15] for a good rundown on hard lattice based problems. Today, essentially all homomorphic encryption schemes are based on LWE and RLWE.

The parameters for LWE are the integers $n = n(\lambda)$ for the dimension, $q = q(n)$ for the global modulus, m for number of samples and a discrete error distribution measure $\chi = \chi(\lambda)$ over $\mathbb{Z}$.

Consider the space of $n \times m$ matrices $\mathbb{Z}_q^{n\times m}$ with uniform distribution, space of secrets $\mathbb{Z}_q^n$ with uniform distribution[1], space of errors $\mathbb{Z}^m$ with discrete distribution χ^m and the random variable $A_{n,q,\chi,m}$ defined on the direct product as follows

$$\begin{aligned} A_{n,q,\chi,m} \colon \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^n \times \mathbb{Z}^m &\to \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^m \cong \mathbb{Z}_q^{(n+1)\times m} \\ (\mathbf{A}, s, e) &\mapsto (\mathbf{A}, b^T \overset{\text{def}}{=} s^T\mathbf{A} + e^T \pmod q) = \begin{bmatrix} \mathbf{A} \\ b^T \end{bmatrix} \end{aligned}$$

[1]The space of secrets can have distribution χ^n without loss of hardness. See [App+09].

Definition 29 (LWE distribution)**.** The learning with errors distribution is defined as the distribution of $A_{n,q,\chi,m}$

$$\mathrm{LWE}_{n,q,\chi,m} \overset{\text{def}}{=} \mathcal{L}(A_{n,q,\chi,m})$$

In words, sample a $n \times m$ matrix $\mathbf{A}$ with entries in $\mathbb{Z}_q$ at random, choose a uniformly random secret $s \leftarrow \mathbb{Z}_q^n$ and let e consist of m independent errors from the discrete distribution χ. Calculate b^T (i.e, $s^T\mathbf{A} + e^T \pmod q$) and output the pair $(\mathbf{A}, b^T)$. The LWE distribution specifies the probability of sampling each pair.

There are two versions of the LWE problem; search-LWE and decision-LWE. Informally, take a sample $x \leftarrow \mathrm{LWE}_{n,q,\chi,m}$ and consider the last row. The decision version is to decide whether the last row is a linear combination of the previous n rows or if it is uniformly random and the search version is to find the explicit linear combination assuming it exists.

Definition 30 (decision-LWE problem)**.** Let $\mathcal{U}_n$ be the uniform distribution on $\mathbb{Z}_q^{(n+1)\times m}$. Construct a PPT algorithm A such that

$$|\Pr[A(\mathcal{U}_n) = 1] - \Pr[A(\mathrm{LWE_{n,q,\chi,m}}) = 1]| > \mathrm{negl}(n)$$

The decision-LWE hardness assumption is that $\mathcal{U}_n$ and $\mathrm{LWE}_{n,q,\chi,m}$ are computationally indistinguishable (i.e., $\mathrm{LWE}_{n,q,\chi,m}$ is pseudorandom).

Definition 31 (search-LWE problem)**.** Let $x = (\mathbf{A}, b^T) \leftarrow \mathrm{LWE}_{n,q,\chi,m}$. Construct a PPT algorithm A such that

$$\Pr[A(x) = s] > \mathrm{negl}(n)$$

The search-LWE hardness assumption is that $\Pr[A(x) = s] = \mathrm{negl}(n)$. There exists a reduction from search-LWE to decision-LWE for $q = \mathrm{poly}(n)$, meaning they are equivalently hard [Bra+13]. We write LWE assumption to refer to hardness of both the search and decision versions.

For the sake of concreteness, typical parameters are $n = \Theta(\lambda), q = \mathrm{poly}(n)$, $m = \omega(n)$, $\chi = \mathrm{D}_{\mathbb{Z},\sqrt{n},\vec{0}}(\vec{x})$ for bound $\beta = 3\sigma$. Remember that this β implies this distribution is subgaussian with parameter $3s = 3\sqrt{n}$, which is exactly the error parameter we use for the subgaussian distribution in our implementation, see Section 8.6. The LWE problem can be thought of as the matrix $\mathbf{A}$ acting on the secret s as a linear transformation, generating the vector $s^T\mathbf{A}$ in the rowspace of $\mathbf{A}$ as the 'true' solution. The problem is then to find $s^T\mathbf{A}$ given a b in the m dimensional ball with radius specified by error bound, centered at $s^T\mathbf{A}$.[2]. In Chapter 8, we show how to base a cryptosystem on the LWE hardness assumption.

5.2 RLWE

The LWE problem is conceptually easy to understand but suffers from expensive overhead. Each element in b is a perturbed linear combination of the secret s and the corresponding column in $\mathbf{A}$, resulting in $O(n)$ operations. To get $m = n$ samples, the total number of operations is $O(n^2)$. In 2012, a more compact ring based learning with errors problem (RLWE) was introduced by Lyubashevsky, Peikert and Regev in [LPR12].

The parameters for RLWE are the integers $n = n(\lambda)$ for the dimension, $q = q(n)$ for the global modulus and a discrete error distribution measure $\chi = \chi(\lambda)$ over $\mathbb{Z}$.

Let the dimension $n = 2^k$ for some natural number k and consider the irreducible polynomial $f(x) = x^n + 1$. Define the polynomial ring $R_q \overset{\text{def}}{=} \mathbb{Z}_q[x]/\langle f(x)\rangle$. The ring R_q is viewed as the ring of polynomials with coefficients in $\mathbb{Z}_q$, having degree less than n. There is a natural correspondence between elements of R_q and $\mathbb{Z}_q^n$ where each polynomial coefficient corresponds to an element in the vector. Let the error space R'_q be R_q endowed with error distribution χ^n over the integer coefficients and consider the random variable $A'_{n,q,\chi}$ defined as follows

$$\begin{aligned} A'_{n,q,\chi}\colon R_q \times R_q \times R'_q &\to R_q^2 \\ (a, s, e) &\mapsto (a, b \overset{\text{def}}{=} a \cdot s + e \pmod q) \end{aligned}$$

Definition 32 (RLWE distribution)**.** The learning with errors distribution is defined as the distribution of $A'_{n,q,\chi}$

$$\mathrm{RLWE}_{n,q,\chi} \overset{\text{def}}{=} \mathcal{L}(A'_{n,q,\chi})$$

In words, sample a uniformly random polynomial $a \in R_q$, a uniformly random secret $s \in R_q$ and a random error $e \in R_q$ from the discrete distribution χ^n. Calculate b (i.e, $a \cdot s + e \pmod q$)) and output the pair (a, b). The RLWE distribution specifies the probability of sampling each pair.

[2]The knowledgable reader may see the resemblence to the bounded-distance decoding (BDD) problem

The decision-RLWE problem is to distinguish between the RLWE distribution and the uniform distribution on R_q^2 and the search problem is to find the secret s given a sample $(a, b) \leftarrow \text{RLWE}_{n,q,\chi}$.

Definition 33 (decision-RLWE problem)**.** Let $\mathcal{U}_n$ be the uniform distribution on R_q^2. Construct a PPT algorithm A such that

$$|\Pr[A(\mathcal{U}_n) = 1] - \Pr[A(\text{RLWE}_{\text{n,q},\chi}) = 1]| > \text{negl}(n)$$

The decision-RLWE hardness assumption is that $\mathcal{U}_n$ and $\text{RLWE}_{n,q,\chi}$ are computationally indistinguishable (i.e., $\text{RLWE}_{n,q,\chi}$ is pseudorandom).

Definition 34 (search-RLWE problem)**.** Let $x = (a, b) \leftarrow \text{RLWE}_{n,q,\chi}$. Construct a PPT algorithm A such that

$$\Pr[A(x) = s] > \text{negl}(n)$$

The search-RLWE hardness assumption is that $\Pr[A(x) = s] = \text{negl}(n)$.

Note that in LWE, the sample size m is embedded as a parameter while in RLWE, adversary generates $m = \text{poly}(\lambda)$ samples from the $\text{RLWE}_{n,q,\chi}$ themselves.

In LWE, the b vector is of size m. Each element is calculated by an inner product of the secret s and the corresponding column in $\mathbf{A}$, resulting in $O(n)$ operations. To get $m = n$ samples, the total number of operations is $O(n^2)$. For RLWE, the b vector is a polynomial with n coefficients. By using the fast fourier transform (FFT) algorithm, polynomial multiplication can be calculated in $O(n \log n)$ operations. To get n samples, only one polynomial multiplication is needed, resulting in $O(n \log n)$ operations. As a consequence, the cost of generating public keys is reduced from $O(n^2)$ to $O(n \log n)$ by exploiting the structure of the ring.

6. History of fully homomorphic encryption

The history of homomorphic encryption began with Rivest, Adleman and Dertouzos [RAD78] already in 1978 where they identified the use of delegating computation to a third party. They were not successful in constructing a secure scheme that supports both multiplication and addition of ciphertext, as required for arbitrary function evaluation. The problem of finding a secure scheme that supports both operations turned out to be difficult and remained unsolved for more than 30 years. Interestingly, multiple secure scheme that supports either multiplication or addition of ciphertexts were discovered. For example, the RSA scheme supports multiplication of ciphertexts and the Paillier scheme supports addition of ciphertexts. It was not until 2009, when Craig Gentry published his PhD thesis [Gen09], that a fully homomorphic encryption scheme was first constructed. Gentry's original scheme was inefficient and since then, many improved schemes have been introduced, both in terms of efficiency and security. In this chapter, we will go through the history of fully homomorphic encryption, split into four generations.

6.1 First generation

Gentry scheme is based on ideal lattices. The construction considers a polynomial ring over the integers modulo an ideal generated by a cyclotomic polynomial, similar to the RLWE setup. Then, there exists an ideal with norm polynomially bounded with respect to the dimension such that the ideal generates a lattice (see [Gen09] for details). The security of Gentry's scheme was based on three hardness assumptions: sparse subset-sum problem (SSSP), bounded-distance decoding problem (BDD) and the ideal shortest vector problem (ideal-SVP). The original decryption function in Gentry's scheme was not bootstrappable (see definition 35), without the use of a method he introduced, called squashing. The idea was to include extra information in the public key which allowed for easier decryption, which meant that security also required the SSSP hardness assumption.

The Gentry's scheme was improved by Smart and Vercauteren in [SV09] where they introduced batching of multiple plaintexts encrypted into a single ciphertext using the Chinese Remainder Theorem (ciphertext packing is the same idea where multiple ciphertexts are packed together). Gentry and Halevi implemented the scheme in [GH10], including the use of the batching technique and an optimized squashing technique to bring down the degree of the decryption polynomial from hundreds (estimated by Smart and Vercauteren) to 15. In Gentry's original scheme, the key generation algorithm was impractically slow. In their implementation, they reduced the asymptotic complexity to $\tilde{O}\left(n^{1.5}\right)$ for cyclotomic fields where the order of the root of unity is a power of 2 (i.e., $\mathbb{Z}[x]/\langle x^n + 1\rangle$ for $n = 2^k$). Scholl and Smart extended the implementation to arbitrary cyclotomic fields in [SS11].

In 2010, Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan finalized a new scheme (DGHV scheme) that was based on multiplication and addition over the integers as opposed to over polynomial rings [Dij+09]. The hardness of the DGHV scheme was based on the approximate greatest common divisor problem (AGCD), and the SSSP due to squashing of the decryption circuit. Subsequent works optimized the DGHV scheme through modulus switching in [CNT11] and plaintext batching in [Kim+13] and [CLT13].

6.2 Second generation

The second generation began in 2011 with Brakerski and Vaikuntanathan [BV11]. Their paper introduced two main contributions. Their first contribution was to base the hardness of the proposed BV scheme on the well known LWE problem, which means using arbitrary lattices instead of ideal lattices. Their second contribution was the removal of squashing from previous schemes (Gentry and Halevi, independently, also managed to remove squashing in [GH11]). This meant that the BV scheme did not require the SSSP hardness assumption. As an optimization technique, the paper was the first to introduce what would later be called *modulus switching.* Modulus switching is an alternative to Bootstrapping in managing noise growth (see Chapter 7) by scaling down the ciphertext noise and the global modulus, effectively switching the modulus of the scheme. Modulus switching keeps the ratio of noise to modulus the same but is still effective since the magnitude of the noise, and thus also its growth rate, is smaller. A downside of the BV scheme (and the rest of the schemes based on LWE in the second generation) was that an expensive step called re-linearization is necessary to prevent homomorphic multiplication causing ciphertext length from growing exponentially. In the BV scheme, homomorphic multiplication corresponds to a tensor product, changing the structure of the ciphertext. In order to re-linearize the result, the scheme uses key-switching. This requires applying a large re-linearization matrix of size $\Omega(n^3)$ (embedded in the public key) to the tensorproduct.

In a follow up paper in 2011, Brakerski, Gentry and Vaikuntanathan introduced the BGV scheme that showed, for the first time, bootstrapping was not necessary for a fully homomorphic encryption scheme. Their construction worked for both LWE and RLWE, and ironically used bootstrapping as an optimization technique. Brakerski later modified the LWE based BGV scheme by replacing the modulus switching technique with a similar *scale invariant* approach [Bra12]. The idea was to scale down the ciphertext by the constant global modulus q, thus reducing the noise bound $\beta < q$ to a corresponding fraction, mod 1. Brakerski showed that noise growth for scale invariant ciphertexts under homomorphic multiplication only grows polynomially w.r.t dimension n (and hence security parameter). Fan and Vercauteren converted Brakerski's scale invariant scheme to the RLWE setting, resulting in the BFV scheme [FV12]. It is worth mentioning that as a part of the second generation of homomorphic encryption, there were schemes based on the NTRU public key encryption scheme from 1998 [HPS98]. However, these schemes required parameter sizes larger than originally proposed due to vulnerabilities from subfield lattice attacks [ABD16].

6.3 Third generation

In 2013, Gentry, Sahai and Waters introduced a new scheme called the GSW scheme [GSW13]. The GSW scheme was also based on LWE, but unlike its predecessors, it did not require the re-linearization step. This is because the scheme operates on matrices which has an inherent natural addition and multiplication operation. Since the relinearization matrix is no longer needed, the space complexity of GSW is quasi-quadratic as opposed to quasi-cubic for BGV and BFV. Furthermore, GSW also did not require bootstrapping to achieve FHE. A downside of the GSW scheme is the complexity of ciphertext matrix multiplication, yielding in slower multiplication than the ring based second generation schemes.

A defining trait of the third generation is the efficiency gain from bootstrapping. There were two main new faster bootstrapping algorithms. Jacob Alperin-Sheriff and Chris

Peikert [AP13] introduced bootstrapping of non-packed ciphertexts in quasilinear time w.r.t the security parameter (which is important in the third generation as there is no packing) and Gama et al. [Gam+14] built on their work and introduced a new type of homomorphic gate. Léo Ducas and Daniele Micciancio developed a scheme called FHEW that used Alperin-Sheriff and Peikerts bootstrapping algorithm to allow for much faster bootstrapping using what is now called programmable bootstrapping [DM14]. Another paper introuced a scheme based on the LWE over the torus, meaning a stricter security assumption, called TFHE [Chi+18] which allowed for bootstrapping in 0.1 milliseconds after subsequent optimizations [MP20].

6.4 Fourth generation

The fourth generation of homomorphic encryption schemes began in 2016 when Jung Hee Cheon, Andrey Kim, Miran Kim and Yongsoo Song introduced a new RLWE based leveled FHE scheme called CKKS [Che+16] which they subsequently turned into a pure FHE scheme through bootstrapping in [Che+18]. CKKS is based on approximate arithmetic, allowing for computation on real and complex numbers. To achieve this, a vector of numbers (real or complex) are encoded using rounding as a integral plaintext polynomial in a cyclotomic polynomial ring. This is a fundamentally different approach as the scheme operates on an approximation of the message as opposed to the real message. The CKKS scheme is useful for applications concerning floating point numbers, such as machine learning and neural networks. Subsequent works focused on optimizing the bootstrapping and ciphertext packing techniques.

In 2020, Baiyu Li and Daniele Micciancio showed that the CKKS scheme was vulnerable to passive attacks with respect to IND-CPA security [LM20]. To mitigate this issue, a new security notion called IND-CPA+ was introduced. IND-CPA+ and IND-CCA1 differ in that the adversary is only allowed to query decryption on evaluated ciphertexts, as opposed to arbitrary ciphertexts. Li and Micciancio showed that IND-CPA+ is eequivalent to IND-CPA for exact encryption schemes but strictly stronger for approximate schemes like CKKS.

7. Noise management

The main problem with homomorphic encryption schemes is the noise. Noise is introduced in the ciphertexts during the encryption process and when the ciphertexts undergo homomorphic operations, the noise grows. After sufficiently many operations, the noise grows to the point where the decryption of the evaluated ciphertext fails. In order to reach FHE, it is necessary to control the noise to allow for sufficient number of operations. Noise management is the process of controlling the noise during homomorphic evaluation. In his 2009 seminal PHD thesis [Gen09], Craig Gentry showed that FHE was possible by using a noise management technique called Bootstrapping. Bootstrapping is an algorithm that transforms a possibly noisy ciphertext into a correct evaluated ciphertext with little noise by using an encryption of the secret key to decrypt the noisy ciphertext homomorphically.

Throughout this chapter, ciphertext are hard-wired into decryption algorithms, meaning that each decryption algorithm is nessesarily correct for only one specified ciphertext. This is only to simplify notation. It is equivalent to require only one decryption algorithm, passing any ciphertext as input. The difference is that the message need to be encrypted twice (possibly under different keys) in the latter case, since the final decryption removes one layer of encryption on both arguments resulting in a once encreypted message and pure secret key as required. Some authors, including Gentry in his original paper, specify bootstrapping in this way.

7.1 Key-Switching

This section is based on [Bra18].

As a natural segway into bootstrapping, we first introduce a related but different concept called key-switching. Since HE schemes are designed with the constraint of supporting homomorphic operations, they are often inefficient. Therefore, it would be desirable to use a more efficient non-homomorphic scheme to encrypt the large messages, but still retaining the homomorphic property. Key-switching is a technique that allows for homomorphic computation on ciphertexts encrypted under a non-homomorphic scheme.

In particular, it allows for transforming a ciphertext under a non-HE scheme to a corresponding ciphertext under a HE scheme. The idea is to encrypt the much shorter secret key of the non-HE scheme under the public key of the HE scheme and hardwire the ciphertexts into the function of interest. Let (H.KeyGen, H.Enc, H.Dec, Eval) be a homomorphic encryption scheme and (KeyGen, Enc, Dec) be a non-homomorphic encryption scheme. Let $(hpk, hsk) \leftarrow \mathrm{H.KeyGen}(\lambda)$ and $(pk, sk) \leftarrow \mathrm{KeyGen}(\lambda)$. Consider computable function C, the vector of ciphertexts $c \leftarrow \mathrm{Enc}(pk, m)$ and an encrypted secret key $sk' \leftarrow \mathrm{H.Enc}(hpk, sk)$. Define $\hat{C}_c(\cdot) \overset{\text{def}}{=} C(\mathrm{Dec}(\cdot, c))$. $C(m)$ can be computed homomorphically by decrypting $\mathrm{Eval}(hpk, \hat{C}_c, sk')$. Indeed, this is a correct encryption of $C(m)$ since

$$\mathrm{Dec}(hsk, \mathrm{Eval}(hpk, \hat{C}_c, sk')) = \hat{C}_c(sk) = C(\mathrm{Dec}(sk, c)) = C(m)$$

We have shown that it is possible to use a non-homomorphic encryption scheme to encrypt the message and still perform homomorphic computation on it. Key-switching transforms a ciphertext encrypted under a non-HE scheme to an evaluated ciphertext encrypted under a HE scheme. The underlying assumption is that the HE scheme can evaluate $\hat{C}_c$, meaning it can first run the decryption algorithm and then compute the desired function C to generate a valid evaluated ciphertext. Even under the assumption that the decryption algorithm is simple enough to be evaluated by the scheme, a general C consists of several multiplication and addition gates, meaning it is unlikely that the scheme can evaluate $\hat{C}_c$. However, if it is possible to split C into smaller components, $C = C^m \circ \cdots \circ C^1$, and evaluate each component separately, $c_{(i)} = \mathrm{Eval}(hpk, \hat{C}^i_{c_{(i-1)}}, sk')$, where $c_{(0)} \leftarrow Enc(pk, m)$, computing C homomorphically is achievable assuming $\hat{C}^i_{c_{(i-1)}}$ is permissible for all i. As the construction currently stands, further computation on the ciphertext is not possible. To see why, consider $c_1 = \mathrm{Eval}(hpk, \hat{C}^1_{c_{(0)}}, sk')$ and let $c_2 = \mathrm{Eval}(hpk, \hat{C}^2_{c_1}, sk')$. We hope decrypting c_2 yields $(C^2 \circ C^1)(m)$, but $\mathrm{Dec}(hsk, c_2) = \mathrm{Dec}(hsk, \mathrm{Eval}(hpk, \hat{C}^2_{c_1}, sk')) = C^2(\mathrm{Dec}(sk, c_1))$. However, since c_1 is encrypted under the HE scheme, the non-homomorphic decryption algorithm applied to c_1 does not make sense and decryption fails.

7.2 Bootstrapping

Key-switching is a nice optimization technique for encrypting messages faster, but it does not allow for further computation on the ciphertext. The requirement is that the

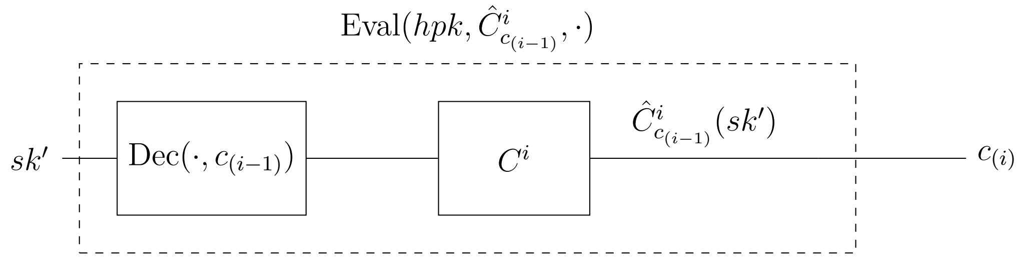
$\mathrm{Eval}(hpk, \hat{C}^i_{c_{(i-1)}}, \cdot)$
sk'
$\mathrm{Dec}(\cdot, c_{(i-1)})$
C^i
$\hat{C}^i_{c_{(i-1)}}(sk')$
$c_{(i)}$

Figure 3: Partial homomorphic computation of $C = C^m \circ \cdots \circ C^i \circ \cdots \circ C^1$ using bootstrapping. For $i = m$ the output decrypts to $C(m)$.

decryption algorithm and the secret key originate from the HE scheme. We therefore only consider the HE scheme. Let (KeyGen, Enc, Dec, Eval) be a HE scheme, $(pk, sk) \leftarrow \mathrm{KeyGen}(\lambda)$ and $sk' \leftarrow \mathrm{Enc}(pk, sk)$. Since this construction encrypts the secret key using its own public key, circular security is assumed (for now). Under this construction, decryption of c_1 is still correct as $\mathrm{Dec}(sk, c_1) = C^1_{c_{(0)}}(sk) = C^1(\mathrm{Dec}(sk, c_{(0)})) = C^1(m)$, but the difference is that decryption of $c_{(i)}$ also works since, by induction,

$$\mathrm{Dec}(hsk, \mathrm{Eval}(hpk, \hat{C}^i_{c_{(i-1)}}, sk')) = \hat{C}^i_{c_{(i-1)}}(sk) = C^i(\mathrm{Dec}(sk, c_{(i-1)})) = C^i \circ C^{i-1} \circ \cdots \circ C^1(m)$$

In essence, we have constructed an algorithm to evaluate C on a ciphertext c as follows: The first step is to split C into m smaller components, $C = C^m \circ \cdots \circ C^1$. The second step is to run the encrypted secret key through the decryption circuit for the ciphertext. The third step is to evaluate the first component on the ciphertext output of the previous step and let this be the ciphertext. The fourth step is to repeat the second and third step for a total of m times, incrementing the component each time. The final ciphertext decrypts to $C(m)$. One iteration of step 2 and 3 is illustrated in Figure 3.

Splitting the desired function C into multiple components $C^1, \ldots, C^m$ is what allows for constructing FHE. Consider the simplest case; each component function is either a multiplication gate or an addition gate. In other words, for a pair of ciphertexts $c = \langle c_1, c_2 \rangle$, define a multiplicative component circuit of C as $C^i(c) = c_1 \times c_2$, $\hat{C}^i_c(\cdot) = C^i(\mathrm{Dec}(\cdot, c)) = C^i(\langle \mathrm{Dec}(\cdot, c_1), \mathrm{Dec}(\cdot, c_2) \rangle) = \mathrm{Dec}(\cdot, c_1) \times \mathrm{Dec}(\cdot, c_2)$ and where the addition case is analogous. Note that $\hat{C}^i_c(sk) = m_1 \times m_2$. It turns out that if the scheme can evaluate just these two types of circuits, then it can evaluate every circuit. This is the idea behind bootstrapping.

Definition 35 (Bootstrappable encryption scheme)**.** Let $\mathcal{E}$ be a $\mathcal{C}$-homomorphic encryption scheme. Consider the following *augmented decryption circuits* for $\mathcal{E}$:

$$B^{(m)}_{c_1,c_2}(\cdot) \stackrel{\text{def}}{=} \text{Dec}(\cdot, c_1) \times \text{Dec}(\cdot, c_2)$$
$$B^{(a)}_{c_1,c_2}(\cdot) \stackrel{\text{def}}{=} \text{Dec}(\cdot, c_1) + \text{Dec}(\cdot, c_2)$$

$\mathcal{E}$ is *bootstrappable* if

$$\{B^{(m)}_{c_1,c_2}(\cdot), B^{(a)}_{c_1,c_2}(\cdot) \mid c_1, c_2 \in \mathcal{Y}\} \subset \mathcal{C}$$

The augmented decryption circuits have the ciphertexts hard-wired into its description and takes as input an encryption of the secret key. A bootstrappable scheme correctly evaluates the set of all augmented decryption circuits. In particular, it correctly evaluates its own decryption circuit by letting c_2 be an encryption of the multiplicative or additive identity respectively.

Theorem 6 (Gentrys bootstrapping theorem, simplified by Vaikuntanathan [Gen09; Vai11])**.** *Any bootstrappable scheme can be transformed into a leveled FHE. Furthermore, if circular security holds, it can be transformed into a pure FHE scheme.*

Remark. Bootstrapping is sufficient for leveled FHE, but not necessary. See [BGV14] for a leveled FHE scheme that does not require bootstrapping.

Proof. Let (KeyGen, Enc, Dec, Eval) be a bootstrappable scheme. Assume first that circular security holds. We construct a pure FHE scheme (KeyGen', Enc', Dec', Eval') as follows:

1. **Key generation**: Generate a key pair $(\hat{pk}, sk)$ using KeyGen(1^λ). Let $sk' \leftarrow$ Enc$(\hat{pk}, sk)$. Define $pk = (\hat{pk}, sk')$ and let KeyGen'(1^λ) return (pk, sk).
2. **Encryption**: Let Enc' be the same as Enc.
3. **Decryption**: Let Dec' be the same as Dec.
4. **Evaluation**: Let Eval'(pk, C, c) transform input circuit C as follows: For each layer of the circuit, consider the gate taking ciphertexts c_i, c_j. If it is a multiplication gate, swap it with $B^{(m)}_{c_i,c_j}(\cdot)$ and if it is an addition gate, swap it with $B^{(a)}_{c_i,c_j}(\cdot)$. Run the encrypted secret key[1] through the augmented decryption circuit and let the outputs act as ciphertext inputs to the next layer. Repeat for all layers of C and denote the transformed circuit C'. Let Eval'(pk, C, c) be the vector of ciphertexts

[1]The encrypted secret key can be seen as an advice wire over the circuit, assessible at any layer

at the output wires of C'.

To see why decryption is correct, remark that the scheme can evaluate each augmented decryption circuit. Therefore, the output ciphertexts decrypts to the gate applied to the decryption of the input ciphertexts. By induction, the input ciphertexts in turn also decrypts correctly since the input layer to the circuit is fresh ciphertexts. More formally, consider circuit output k, denoted $C(m)_k$, undergoing a (say multiplication) operation in the last layer. Then, $C(m)_k = C_1(m) \times C_2(m)$ for some subcircuits C_1, C_2 of C. Let ciphertext c be input to the transformed circuit C' and assume that, by induction, input ciphertexts for last layer z_1, z_2 satisfies $\text{Dec}(sk, z_1) = C_1(m)$ and $\text{Dec}(sk, z_2) = C_2(m)$. Then, decryption of $C'(c)_k$ yields

$$\begin{aligned}\text{Dec}(sk, \text{Eval'}(pk, C, c)_k) &= \text{Dec}(sk, B^{(m)}_{z_1,z_2}(sk')) = B^{(m)}_{z_1,z_2}(sk) \\ &= \text{Dec}(sk, z_1) \times \text{Dec}(sk, z_2) = C_1(m) \times C_2(m) = C(m)_k\end{aligned}$$

Security of the scheme holds by the fact that the original scheme is secure and circular security hold, meaning the encrypted secret key under its own public key does not compromise the security. In other words, the evaluate algorithm is only using public information to evaluate the circuit. Since the circuit was arbitrary, the scheme is a pure FHE scheme. Assume now that circular security does not hold. We construct a leveled FHE scheme (KeyGen', Enc', Dec', Eval') as follows:

1. **Key generation**: Given input parameters $(1^\lambda, 1^L)$, generate $L+1$ key pairs $(\hat{pk}_i, sk_i)_{i=0,\ldots,L}$ using KeyGen. Let $sk'_i \leftarrow \text{Enc}(\hat{pk}_{i+1}, sk_i)$ for all $i = 0, \ldots, L-1$. Define $sk = (sk_0, \ldots, sk_L)$ and define $pk = (pk_0, sk'_0, pk_1, sk'_1, \ldots, sk'_{L-1}, pk_L)$ and let KeyGen' return (pk, sk).

2. **Encryption**: Let Enc' be the same as Enc.

3. **Decryption**: Let Dec' be the same as Dec.

4. **Evaluation**: Let Eval' take input circuit of depth at most L and 'pad' it so that it has depth exactly L. Transform the circuit as follows: For layer $i = 1, \ldots, L$ of the circuit, consider the gate taking ciphertexts c_i, c_j. If it is a multiplication gate, swap it with $B^{(m)}_{c_i,c_j}(\cdot)$ and if it is an addition gate, swap it with $B^{(a)}_{c_i,c_j}(\cdot)$. Run the encrypted secret key sk'_{i-1} through the augmented decryption circuits and let the outputs act as ciphertext inputs to the next layer. Let Eval'(pk, C, c) be the vector of ciphertexts at the output wires of C'.

In the transformed circuit C', the input ciphertexts to layer i is encrypted under public key $i-1$. In other words, to decrypt the ciphertexts resulting from layer i, we use private key sk_i. Since the circuit has exactly L layers, the last layer is decrypted using sk_L. Using the same notation as before, the correctness of the decryption algorithm follows by same argument:

$$\begin{aligned}\text{Dec}(sk_L, \text{Eval'}(pk, C, c)_k) &= \text{Dec}(sk_L, B^{(m)}_{z_1,z_2}(sk'_{L-1})) = B^{(m)}_{z_1,z_2}(sk_{L-1}) \\ &= \text{Dec}(sk_{L-1}, z_1) \times \text{Dec}(sk_{L-1}, z_2) \\ &= C_1(m) \times C_2(m) = C(m)_k\end{aligned}$$

The security of the scheme follows by the fact the each encrypted secret key is encrypted under the public key of the next layer, meaning that it is computationally indistinguishable from a random ciphertext. In other words, the secret keys are not encrypted under their own public keys, avoiding circular security assumption. Since the circuit was arbitrary, the scheme is a leveled FHE scheme. □

The difference between the pure FHE scheme and the leveled FHE scheme is that in the former, there is only need for one key pair (pk, sk), where the encrypted secret key can be made public (by circular security assumption) and used to bootstrap in every layer. This implies the transformed circuit can have arbitrary depth since each layer outputs a valid evaluated ciphertext. In the latter, circular security does not hold and the encrypted secret key cannot be reused. Therefore, every layer needs a new encrypted secret key. In general, any scheme generating a fixed amount of valid keys pairs can only bootstrap finitely many times. Since ciphertext noise grows for each operation, the depth of the circuit has to be bounded. In particular, for a set of circuits with depth at most L, $L+1$ valid keypairs is sufficient.

Practical noise management and intuition

As of today, pure FHE requires circular security. Circular security assumption improves the efficiency of the scheme since the key generation algorithm is run only once. However, the construction is still extremely inefficient. Bootstrapping is generally an expensive operation since evaluation is on ciphertexts which contains more bits than the plaintexts they encrypt, resulting in computational overhead. In practice, bootstrapping is only done when necessary, meaning when the noise is about to cause decryption to fail. When this is about to happen, the ciphertext $c \leftarrow \text{Enc}(pk, m)$ is "refreshed" by replacing it with the less noisy $\text{Dec}(sk', c)$. Note that this is essentially an augmented decryption circuit where the other ciphertext is an encryption of the multiplicative (or additive) identity. Indeed, $\text{Dec}(sk, \text{Dec}(\cdot, c), sk') = \text{Dec}(sk, c) = m$. In the next chapter, we will implement bootstrapping in the GSW scheme which, together with the circular security assumption gives a pure FHE scheme. The GSW decryption function (as does all LWE based cryptosystems) uses a rounding function on an encrypted integer. The rounding function provides intuition on how noise can be reduced. To start, consider first the non-homomorphic case, meaning decryption is done normally using the noisy ciphertext and a normal, unencrypted secret key. If we can be confident that the noise is comparatively small, then by rounding to the nearest possible plaintext corresponds to removing the noise. For example, assume the possible plaintexts are 0 and 1 and the rounding function is defined to map to 0 the values closer to -100 and map to 1 the values closer to 100. Say we somehow know that it is very unlikely the magnitude of the noise surpasses 50. Then, observing an encrypted value of 80 corresponds to removing the noise -20 and mapping the encrypted integer to 1. This is essentially how we reduce noise in the implementation in the next chapter. The homomorphic case works the same way. When using the encrypted secret key, computing the decryption function means new noise is introduced from operations on the encrypted secret key. However, if the noisy ciphertext has noise small enough to allow for this extra introduced noise and if the bootstrapping algorithm indeed does implement the rounding function, then the rounding function is still applied on the ciphertext, removing the original noise. What we are left with is the noise from computing on the secret key. In Theorem 10, we prove an explicit bound on error of the refreshed ciphertext.

8. Implementation

In this chapter we will present a fully homomorphic encryption scheme, called GSW, developed by Gentry, Sahai and Waters [GSW13]. The GSW scheme is based on matrices and approximate eigenvectors. The appeal of the scheme is its relative conceptual simplicity due to natural matrix addition and matrix multiplication operations. This chapter is based on the implementation in the follow up work of Alperin-Sheriff and Peikert [AP13]. Alperin-Sheriff and Peikert introduced a syntactically simpler implementation of the GSW together with a new bootstrapping algorithm. Their bootstrapping algorithm relies on representing integers as tuples of cyclic shifts, as discussed in Section 3.4, and decrypting a ciphertext homomorphically using the inner product and an appropriate rounding function. To minimize confusion, we generally use similar notation to [AP13].

This chapter is structured by presenting concepts as follows: We present the GSW scheme, which is used to encrypt bits. We show the scheme's correctness, security guarantees, homomorphic operations and bounds on error growth. Then, we introduce the slightly more general HEPerm scheme, which is used to encrypt integers encoded as cyclic shift permutations by using the GSW scheme to encrypt each bit. We introduce two operations to HEPerm, called homomorphic composition and homomorphic equality test and then prove results on error growth similar to those in GSW. Finally, we present the bootstrapping algorithm and show how to decrypt a ciphertext homomorphically using the inner product and an appropriate rounding function. We then prove results for the bound of the error associated to the refreshed ciphertext. Specifically we show that the refreshed ciphertext has an associated subgaussian error vector with parameter small enough to decrypt correctly except for negligible probability.

At a high level, the idea behind this chapter is to concretely demonstrate a protocol for which a server can compute the data of a client, homomorphically. More specifically, for any program with arbitrary complexity, we show how a server can execute the program using only encryptions of the input bits, encryption of the secret key and the program's Boolean circuit representation of XOR and AND gates. The protocol has the client encrypting their input bits using the GSW scheme and their secret key using the HEPerm scheme. The client sends encrypted data to the server which runs the matrices through the homomorphic version of the circuit, where each XOR and AND gate is swapped for the corresponding addition and multiplication operation of the GSW scheme. If the depth of the circuit is known to be small a priori, then there is an efficient GSW initialization such that the error growth associated with the ciphertext matrices in the homomorphic version of the program does not cause decryption to fail. In this case, the server simply runs the homomorphic version of the circuit and sends back the GSW ciphertexts to the client who decrypts them for the final result. On the other hand, if the depth of the circuit is unspecified, the server may need to refresh the ciphertext matrices using the bootstrapping algorithm. For the bootstrapping algorithm, the client sends the server their encrypted secret key using the HEPerm scheme. The server then runs the decryption function homomorphically (i.e., by using the encrypted secret key) to reduce the noise and continues to run the homomorphic version of the circuit on the refreshed ciphertext.

8.1 Flattening gadget

In the implementation of the GSW scheme, the ciphertexts are matrices defined over $\mathbb{Z}_q^{n\times nl}$ where $l = \lceil \log q \rceil$. To avoid rapid error growth under homomorphic multiplication, we want to represent integers mod q as a corresponding low norm vector through the decomposition function (also called flattening gadget) G^{-1}. To get back the original integer x we require $\langle \mathbf{g}, G^{-1}(x)\rangle = x$ where $\mathbf{g} \stackrel{\text{def}}{=} (1, 2, \ldots, 2^{l-1}) \in \mathbb{Z}_q^l$. A particular number of interest later is the penultimate entry 2^{l-2}. Since $2^l \in [q, 2q)$ we have $2^{l-2} \in [q/4, q/2)$.

We can easily extend the decomposition function to work on vectors and matrices by applying the function to each element in the vector or matrix. For a column vector $\mathbf{v} \in \mathbb{Z}_q^n$, we define the column vector $G^{-1}(\mathbf{v}) \stackrel{\text{def}}{=} [G^{-1}(v_1)|G^{-1}(v_2)|\ldots|G^{-1}(v_n)] \in \mathbb{Z}^{nl}$ and for a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, define $G^{-1}(\mathbf{A}) \stackrel{\text{def}}{=} [G^{-1}(a_1)|G^{-1}(a_2)|\ldots|G^{-1}(a_m)] \in \mathbb{Z}^{nl\times m}$ where a_i is the ith column of $\mathbf{A}$. Then, for any integer ($n = m = 1$), vector (n or m is 1) or matrix $x \in \mathbb{Z}_q^{n\times m}$ we have $\mathbf{G}G^{-1}(x) = x$ where

$$\mathbf{G} \overset{\text{def}}{=} \begin{bmatrix} 1 \;\; 2 \;\; \cdots \;\; 2^{l-1} & & & \\ & 1 \;\; 2 \;\; \cdots \;\; 2^{l-1} & & \\ & & \ddots & \\ & & & 1 \;\; 2 \;\; \cdots \;\; 2^{l-1} \end{bmatrix} \in \mathbb{Z}_q^{n\times nl}$$

A simple way to implement the G^{-1} mapping is to map an integer $x \in \mathbb{Z}_q$ to its bit decomposition vector $\{0,1\}^l$. In the original implementation of the GSW scheme [GSW13], the decomposition function was precisely the bit decomposition function. However, in the implementation of Alperin-Sheriff and Peikert [AP13], the decomposition function is a randomized algorithm $G^{-1}\colon \mathbb{Z}_q \to \mathbb{Z}^l$ where every coordinate of $G^{-1}(x)$ is a subgaussian distribution over the integers with constant parameter. The reason for a randomized decomposition function is twofold: First, we can use subgaussian random variables for a tight error analysis when multiplying ciphertexts. Second, in the ofchance that a subgaussian error has large norm, it can easily be resampled.

Theorem 7 (Claim 3.1 [AP13])**.** *There is a randomized, efficiently computable function $G^{-1} : \mathbb{Z}_q \to \mathbb{Z}^\ell$ such that $G^{-1}(a)$ is a subgaussian random variable with parameter $O(1)$, and always satisfies $\langle \mathbf{g}, G^{-1}(a)\rangle = a$.*

Remark that, more generally, by applying G^{-1} on every element of a vector or matrix x, $\mathbf{G}G^{-1}(x) = x$. Notice that $\mathbf{G}G^{-1}$ acts as an identity mapping, unlike the randomized matrix $G^{-1}(\mathbf{G})$. We will use the above theorem and its implementation as a black box but the following example provides some intuition.

As an example, consider $q = 8$ and say we want to decompose the integer 5. Then $(1,0,1)$ and $(-1,3,0)$ are both valid samples of $G^{-1}(5)$ since $1\cdot 1 + 0\cdot 2 + 1\cdot 2^2 = 5$ and $-1\cdot 1 + 3\cdot 2 + 0\cdot 2^2 = 5$. Remember that the samples have ”small” entries as they are subgaussian with constant parameters (more specifically, the magnitude of the entries do not grow with q).

8.2 GSW scheme

Here we give a high level description of the GSW scheme. In the GSW scheme, the message is a bit μ and the ciphertext is a matrix $c \in \mathbb{Z}_q^{n\times nl}$. The GSW ciphertext matrix is, in some sense, a linear map where the secret key is an approximate eigenvector with a message bit as eigenvalue. In more detail, the secret vector $s \in \mathbb{Z}_q^n$ should satisfy $s^T c = \mu s^T \mathbf{G} + e^T$ for some message bit μ and some small error vector $e \in \mathbb{Z}_q^{nl}$ associated with ciphertext c. This equation is ubiquitous in the GSW scheme and is denoted the *invariant equation*. The intuition is to hide the bit in a LWE sample by adding $\mu\mathbf{G}$ to the sample. The security of the scheme is based on the LWE hardness assumption. The error distribution is the discrete Gaussian $\chi = \mathrm{D}_{\mathbb{Z},s,\vec{0}}$ with the error bound β. The scheme is thus parameterized by the tuple (n, q, χ, m) where $n(\lambda), q(\lambda), m(\lambda)$ are integers and χ is a distribution. For ciphertexts $c_1, c_2 \in \mathbb{Z}_q^{n \times nl}$, the homomorphic addition is defined as normal matrix addition whereas homomorphic multiplication is defined as the following right associative operation: $c_1 \times G^{-1}(c_2)$.

1. **Key generation**: Generate a uniformly random LWE secret $s' = (s_1, \dots, s_{n-1}) \leftarrow \mathbb{Z}_q^{n-1}$ and let $s = (s_1, \dots, s_{n-1}, s_n \overset{\text{def}}{=} -1) \in \mathbb{Z}_q^n$. Using s', generate the $n \times m$ matrix $\mathbf{A}' \overset{\text{def}}{=} (\mathbf{A}, b^T) \leftarrow \mathrm{LWE}_{n-1,q,\chi,m}$. Let $\mathrm{KeyGen}(1^\lambda, 1^L)$ return $\mathbf{A}' \in \mathbb{Z}_q^{n \times m}$ as the public key and $s \in \mathbb{Z}_q^n$ as the secret key.

2. **Encryption**: Generate a random 'subset' matrix $\mathbf{R} \leftarrow \{0,1\}^{m \times nl}$. To encrypt a bit μ, compute $c \leftarrow \mathrm{Enc}(\mathbf{A}', \mu) \overset{\text{def}}{=} \mu\mathbf{G} + \mathbf{A}'\mathbf{R} \pmod q \in \mathbb{Z}_q^{n \times nl}$.

3. **Evaluation**: For a given Boolean circuit, swap each addition gate to the standard matrix addition function $\boxplus \colon (c_1, c_2) \mapsto c_1 + c_2$ and swap each multiplication gate to $\boxdot \colon (c_1, c_2) \mapsto c_1 \times G^{-1}(c_2)$. Remark that multiplication is right associative. The evaluation algorithm also contains a bootstrapping algorithm which is presented later.

4. **Decryption**: Define the rounding function $[\cdot]_2 \colon \mathbb{Z}_q \to \{0,1\}$ where $[x]_2 = 0$ if $x \pmod{2^{l-1}}$ is closer to 0 than 2^{l-2} and $[x]_2 = 1$ otherwise. Decryption is defined as $[\langle s, c_{pen} \rangle]_2$ where c_{pen} is the penultimate column of ciphertext c and s is the secret key.

Before analyzing the scheme, we define some basic concepts and briefly discuss the scheme

Definition 36. A fresh ciphertext $c \in \mathbb{Z}_q^{n \times nl}$ is *designed to encrypt* bit μ if $c \leftarrow \mathrm{Enc}(\mathbf{A}', \mu)$. A ciphertext sum $c_1 \boxplus c_2$ is *designed to encrypt* bit $\mu_1 + \mu_2 \in \mathbb{Z}_2$ if c_1 and c_2 are designed to encrypt μ_1, μ_2 respectively. A ciphertext product $c_1 \boxdot c_2$ is *designed to encrypt* bit $\mu_1 \cdot \mu_2 \in \mathbb{Z}_2$ if c_1 and c_2 are designed to encrypt μ_1, μ_2 respectively.

Remark that the matrix $\mathbf{G}$ is a valid fresh encryption of the bit 1 where the random matrix is all zeros. $\mathbf{G}$ is therefore designed to encrypt 1. We will use $\mathbf{G}$ as an encryption of 1 to simplify the error analysis of homomorphic multiplication later.

Denote the error from the LWE samples by $e_{LWE} \in \mathbb{Z}_q^m$. Then, we know that $s^T\mathbf{A}' = e_{LWE}$. This error is associated with the LWE samples $\mathbf{A}'$ used to generate the ciphertext c. We are interested in a related but different error directly associated with c through the expression s^Tc. For a fresh ciphertext c with corresponding secret key s, notice that $s^Tc = \mu s^T\mathbf{G} + s^T\mathbf{A}'\mathbf{R} = \mu s^T\mathbf{G} + e_{LWE}^T\mathbf{R} \in \mathbb{Z}_q^{nl}$. Since e_{LWE} is sampled from the subgaussian distribution χ^m, each element $e_1, \dots, e_m$ is subgaussian with parameter s. Therefore, $e_{LWE}^T\mathbf{R}$ is a row vector where each element is an inner product of at most m subgaussian random variables. By additivity of subgaussian random variables, Lemma 2 gives that each element in the row vector is subgaussian with parameter $\sqrt{ms^2} = s\sqrt{m}$. Consequently, $e_{LWE}^T\mathbf{R}$ is a subgaussian row vector in $\mathbb{Z}_q^{nl}$ with parameter $s\sqrt{m}$. Remark that $e_{LWE}^T\mathbf{R}$ is directly associated with c. We simply denote its transpose (i.e., the column vector) e and call it the *ciphertext error* or the error associated with the ciphertext c. Each ciphertext matrix has its own associated error vector in $\mathbb{Z}_q^{nl}$ and each fresh ciphertext has an associated LWE error vector $e_{LWE} \in \mathbb{Z}_q^m$ sampled from χ^m.

The rounding function in the decryption function outputs 0 if and only if $|x - 0 \pmod{2^{l-1}}| < |x - 2^{l-2} \pmod{2^{l-1}}|$ and 1 otherwise. Note here that $-2^{l-2} \pmod{2^{l-1}} = 2^{l-2}$. The easiest way to understand the rounding function is to visualize a circle representing the equivalence classes of $\pmod{2^{l-1}}$ where the rightmost point is 0 and the leftmost point is 2^{l-2} (or its negative value). Encryptions of 1 will decrypt to the leftmost point whereas encryptions of 0 will decrypt to the rightmost point. The idea of the rounding function is to remove all ciphertext noise embedded in $\langle s, c_{pen}\rangle$. To do so, we require that the noise is small enough so that the encrypted messages are not mistaken as the incorrect bit. To implement the bootstrapping algorithm we need to decrypt, and in particular calculate the rounding function, homomorphically. The server can do this by comparing the inner product to all the values on the side of the circle that should be decrypted to 1, as we will see later.

Correctness

Consider a ciphertext $c = \mu \cdot \mathbf{G} + \mathbf{A}'\mathbf{R} \pmod q$ for some random binary matrix $\mathbf{R}$. Then, the penultimate column is $c_{pen} = \mu(0, \dots, 0, 2^{l-2}) + \mathbf{A}'\mathbf{R}_{pen}$ where $\mathbf{R}_{pen} \in \{0,1\}^m$ is the penultimate column of $\mathbf{R}$. Remember that $2^{l-2} \in [q/4, q/2)$. Therefore, $\langle s, c_{pen}\rangle =$

$s^T c_{pen} = -\mu 2^{l-2} + (s^T \mathbf{A}')\mathbf{R}_{pen} = -\mu 2^{l-2} + e_{pen}$ where e_{pen} is the penultimate element of the error vector e associated with c. Notice that for small error (e.g., zero error) e_{pen}, the bit $\mu = 0$ is clearly rounded to 0 and the bit $\mu = 1$ is clearly rounded to 1. What remains is to analyze the effect of the error.

Notice that $2^{l-1} \in [q/2, q)$. Therefore, to ensure that the error is small enough to not cause incorrect rounding, we require that $|e_{pen}| < q/8$. More intuitively, the "length" of the mod 2^{l-1} circle is at least $q/2$ and say the ciphertext is an encryption of the bit 0 with an associated error less than $q/8$. Then it is not possible for the inner product to "exist" on the left side of the circle since that would imply the magnitude of the error is more than a quarter of the circle, meaning $q/8$. The same argument holds for the leftmost point, corresponding to bit 1.

What is left to show is that the error is, except for with negligible probability, sufficiently small to not cause incorrect rounding. We leave this discussion for the upcoming sections where we show how to choose modulus such that decrypting ciphertexts, even after homomorphic operations, is correct with overwhelming probability.

Security

The intuition behind the scheme is to hide the plaintext bit in a LWE sample. Semantic security follows from the LWE hardness assumption. Since, by assumption, $\mathbf{A}'$ is pseudorandom and $\mathbf{R}$ is uniformly random, $\mathbf{A}'\mathbf{R}$ is also pseudorandom. Furthermore, adding $\mathbf{G}$ only translates the distributions, and thus maintains pseudorandomness. In other words, if there is an adversary that can distinguish encryptions of 0 and encryptions of 1, then it can distinguish between $\mathbf{A}'\mathbf{R}$ and $\mathbf{A}'\mathbf{R} + \mathbf{G}$ which is a contradiction to the LWE hardness assumption.

Preserving design under homomorphic operations

Stating that a ciphertext is designed to encrypt a bit is intended as a more relaxed concept of correctness in the sense that if c is designed to encrypt μ, then c decrypts to μ assuming the associated error vector is zero. For this definition of design to make sense, we need to show that the homomorphic operations indeed does decrypt correctly, given sufficiently small error. We say that the homomorphic operation *preserves design*. Let $c_1, c_2 \in \mathbb{Z}_q^{n \times nl}$ be two ciphertexts designed to encrypt the bits μ_1, μ_2 with associated errors e_1, e_2 respectively. In this paragraph, we show that the homomorphic operations decrypts correctly for small errors and we later analyze the error growth. In other words, we show that $c_1 \boxplus c_2$ decrypts to $\mu_1 + \mu_2 \pmod 2$ and $c_1 \boxdot c_2$ decrypts to $\mu_1\mu_2$. Remember, homomorphic addition $c = c_1 \boxplus c_2$ is defined as standard matrix addition and homomorphic multiplication is right associative and defined as $c_1 G^{-1}(c_2)$. The invariant equation for the sum is

$$\begin{aligned} s^T c = s^T(c_1 + c_2) = s^T c_1 + s^T c_2 &= s^T \mathbf{G}\mu_1 + e_1^T + s^T \mathbf{G}\mu_2 + e_2^T \\ &= s^T \mathbf{G}(\mu_1 + \mu_2) + [e_1^T + e_2^T] \in \mathbb{Z}_q^{nl} \end{aligned} \tag{8.1}$$

where the square brackets denote the error associated with the homomorphic sum. Briefly assume the errors are small (say zero). Since the penultimate element of $s^T\mathbf{G}$ is $\big\langle (s_1, \dots, s_{n-1}, -1), (0, \dots, 0, 2^{l-2}) \big\rangle = -2^{l-2}$, decryption for small errors yields $[-2^{l-2}(\mu_1 + \mu_2)]_2$. To argue correctness of decryption, by using the same argument as before, we see that the only relevant case is when $\mu_1 = \mu_2 = 1$. Then, $-2^{l-2}(1+1) = -2^{l-1} = 0 \pmod{2^{l-1}}$ and thus the homomorphic addition of ciphertexts both designed to encrypt 1 yields a ciphertext designed to encrypt 0. The invariant equation for the product is

$$\begin{aligned} s^T(c_1 \boxdot c_2) = s^T(c_1 G^{-1}(c_2)) = s^T(c_1)G^{-1}(c_2) &= (s^T\mathbf{G}\mu_1 + e_1^T)G^{-1}(c_2) \\ &= \mu_1 s^T \mathbf{G} G^{-1}(c_2) + e_1^T G^{-1}(c_2) = \mu_1 s^T(c_2) + e_1^T G^{-1}(c_2) \\ &= \mu_1(s^T\mathbf{G}\mu_2 + e_2^T) + e_1^T G^{-1}(c_2) \\ &= \mu_1\mu_2 s^T \mathbf{G} + [e_1^T G^{-1}(c_2) + \mu_1 e_2^T] \in \mathbb{Z}_q^{nl} \end{aligned} \tag{8.2}$$

where, again, the square brackets denote the error associated with the homomorphic multiplication. Assuming small errors again and the decryption function becomes $[-2^{l-2}\mu_1\mu_2]_2$. It is clear by the same argument as before that this decrypts to 1 if and only if $\mu_1\mu_2 = 1$, meaning $\mu_1 = \mu_2 = 1$ and correctness holds.

Error growth of homomorphic operations

The square brackets in Equation 8.1 and 8.2 show how the associated error grows under homomorphic addition and multiplication. For addition, we observe that the error is merely added together and thus subgaussian by 2. For multiplication however, we use the fact that $\mathbf{G}$ is designed to encrypt 1 and has zero error associated with it. Then, $c_2 \boxdot \mathbf{G}$ is designed to encrypt μ_2 and $c_1 \boxdot (c_2 \boxdot \mathbf{G})$ is designed to encrypt $\mu_1\mu_2$. The reason for expressing the product ciphertext of $\mu_1\mu_2$ as $c_1 \boxdot (c_2 \boxdot \mathbf{G})$ instead of as $c_1 \boxdot c_2$ is to simplify the error analysis. More specifically, by performing the multiplication $c_2 \boxdot \mathbf{G}$ first, we ensure that G^{-1} is applied to c_2 as well, which gives us the following clean error analysis

Theorem 8. *Let $c_1, c_2 \in \mathbb{Z}_q^{n \times nl}$ be two ciphertexts designed to encrypt the bits μ_1, μ_2 with associated errors e_1, e_2 respectively. Then, $c_1 \boxdot c_2 \boxdot \mathbf{G}$ is designed to encrypt $\mu_1\mu_2$ and its associated error is subgaussian with parameter $O(\|e\|)$ where $e = (e_1, e_2) \in \mathbb{Z}^{2nl}$ is the concatenation of the two errors.*

Proof. Remember that $\boxdot$ is right associative, meaning we first compute $c_2 \boxdot \mathbf{G}$ and then $c_1 \boxdot (c_2 \boxdot \mathbf{G})$. The invariant equation, Equation 8.2, gives that the error associated with $c_2 \boxdot \mathbf{G}$ is $e_2^T G^{-1}(\mathbf{G})$ since $\mathbf{G}$ has zero error. Remember that G^{-1} applied on a matrix is defined by applying the decomposition function G^{-1} on every element of the matrix. Therefore, $G^{-1}(G)$ is a $nl \times nl$ matrix where column j contains n decompositions, each containing l integers. By Theorem 7, each of the n decomposition vectors in column j is subgaussian with parameter $O(1)$, and therefore the concatenation, meaning the entire column j, is also subgaussian with parameter $O(1)$. To analyze the error vector $e_2^T G^{-1}(\mathbf{G}) \in \mathbb{Z}_q^{nl}$ associated with $c_2 \boxdot \mathbf{G}$, consider element $j \in [nl]$. Element j is simply an inner product of e_2 and column j of $G^{-1}(\mathbf{G})$. By Lemma 1, element j is therefore subgaussian with parameter $O(\|e_2\|)$. The same analysis holds for every element in $e_2^T G^{-1}(\mathbf{G})$ and thus, the error associated with $c_2 \boxdot \mathbf{G}$ is subgaussian with parameter $O(\|e_2\|)$. Consider now $c_1 \boxdot (c_2 \boxdot \mathbf{G})$. The invariant equation, Equation 8.2, gives that the error associated with $c_1 \boxdot (c_2 \boxdot \mathbf{G})$ is $e_1^T G^{-1}(c_2 \boxdot \mathbf{G}) + \mu_1 e_2^T G^{-1}(\mathbf{G})$. Notice that the second term is, by above argument, subgaussian with parameter $O(\|e_2\|)$. Consider the first term, $e_1^T G^{-1}(c_2 \boxdot \mathbf{G})$. Remember that $c_2 \boxdot \mathbf{G}$ is a valid ciphertext designed to encrypt μ_2 similar to how $\mathbf{G}$ is designed to encrypt the bit 1. By the same argument as before, $e_1^T G^{-1}(c_2 \boxdot G)$ is subgaussian with parameter $O(\|e_1\|)$. By Lemma 2, the sum of subgaussian vector with parameters $\|e_1\|$ and $\|e_2\|$ is subgaussian with parameter $\sqrt{\|e_1\|^2 + \|e_2\|^2} = \|e\|$ where $e = (e_1, e_2) \in \mathbb{Z}^{2nl}$ is the concatenation of the two errors. Homogenity of vectors from Lemma 2 concludes the proof. $\square$

The above theorem can, by induction, be extended to a homomorphic product of k ciphertexts.

Corollary 1. Let $c_1, \ldots, c_k$ be GSW ciphertext designed to encrypt $\mu_1, \ldots, \mu_k$ with associated errors $e_1, \ldots, e_k \in \mathbb{Z}_q^{nl}$, respectively. Then,

$$c_1 \boxdot \cdots \boxdot c_k \boxdot \mathbf{G} \tag{8.3}$$

is designed to encrypt $\mu_1 \times \cdots \times \mu_k$ and its associated error is subgaussian with parameter $O(\|e\|)$ where $e = (e_1, \ldots, e_k) \in \mathbb{Z}^{knl}$ is the concatenation of the k errors.

Proof. The base case follows from Theorem 8. Notice that the invariant equation states that every new ciphertext we multiply into the existing chain of ciphertexts, the error vector grows by an extra error term. More specifically, multiplying c_i to the existing chain $c_{i+1} \boxdot \cdots \boxdot c_k \boxdot \mathbf{G}$ adds the error vector term $e_i^T G^{-1}(c_{i+1} \boxdot \cdots \boxdot c_k \boxdot \mathbf{G})$ to the error of the product ciphertext. Thus, the error associated with $c_1 \boxdot \cdots \boxdot c_k \boxdot \mathbf{G}$ is $\sum_{i=1}^{k} e_i^T G^{-1}(c_{i+1} \boxdot \cdots \boxdot c_k \boxdot \mathbf{G})$. Each term is subgaussian with parameter $O(\|e_i\|)$ by same argument as above and the result follows by addition of subgaussian vectors and homogenity, by Lemma 2. $\square$

The above corollary shows how the error grows in a circuit computing on GSW ciphertexts. Say our circuit requires computing k homomorphic multiplication of fresh GSW encryptions. Since each ciphertext has error with parameter $s\sqrt{m}$, the concatenation of the k errors also has the same parameter. Therefore, the error associated with the final ciphertext is subgaussian with parameter $O(s\sqrt{km})$. For a depth d circuit, there are at most 2^d fresh encryptions and thus the error associated with the final ciphertext is subgaussian with parameter $O(s\sqrt{2^d m})$. To achieve FHE, we need to show how to refresh the ciphertext, thus decreasing the error. The remainder of this chapter is dedicated to showing bootstrap and thus achieve unbounded FHE.

8.3 HEPerm scheme

Remember that integers can be represented as tuples of cyclic shifts through the canonical homomorphism, see Definition 19. Since cyclic shifts are represented as indicator vectors, encrypting the tuples of cyclic shifts corresponds to encrypting integers. The reason for using cyclic shifts is that composition makes computing homomorphic addition easy. The GSW scheme is used to encrypt bits and the HEPerm scheme is used to encrypt integers by encrypting the cyclic shift corresponding to the integer, using the GSW scheme as a subroutine. To simplify notation, we denote the space of GSW ciphertexts as $\mathcal{Y} = \mathbb{Z}_q^{n \times nl}$ and we denote the space of associated errors as $\mathcal{E} = \mathbb{Z}_q^{nl}$.

1. **Key generation**: Same as GSW scheme.
2. **Encryption**: For a given cyclic shift $\pi \in \{0,1\}^r$, encrypt each bit by using the GSW scheme. Output the encrypted column vector $\mathbf{C}^\pi \in \mathcal{Y}^r$.
3. **Decryption**: For a given HEPerm ciphertext in $\mathcal{Y}^r$, apply the GSW decryption function on each GSW ciphertext. Return the unencrypted column in $\{0,1\}^r$.

Similarly to the GSW scheme, every HEPerm ciphertext $\mathbf{C}^\pi \in \mathcal{Y}^r$ has an error associated with it. $\mathbf{E}^\pi \in \mathcal{E}^r$ is the error associated with $\mathbf{C}^\pi$ if every $e_i \in \mathcal{E}$ is the error associated with GSW ciphertext $c_i \in \mathcal{Y}$. Similarly to how $\mathbf{G}$ encrypts the multiplicative identity (i.e., the bit $\mu = 1$) with zero error, we define the HEPerm ciphertext $\mathbf{J} \in \mathcal{Y}^r$ as an encryption of the identity cyclic shift $(1,0,0,\dots,0) \in \{0,1\}^r$. In other words, $\mathbf{J}$ is a zero error encryption of the integer 0 which acts as the identity under composition of HEPerm ciphertexts.

An important remark here is that HEPerm ciphertexts are defined as column vectors of GSW ciphertexts. However, to perform composition of HEPerm ciphertext (corresponding to addition of cyclic shifts), we matrix multiplication requires the left ciphertext to be of dimension $r \times r$. To make the HEPerm ciphertext square we let column j be a copy the the column 1 but cyclically shifted downwards with $j-1$ GSW ciphertexts. We call this the *rotation expanded* HEPerm ciphertext and it is done implicitly whenever there is a composition of HEPerm ciphertexts.

Notice that HEPerm is similar to the GSW scheme. Correctness and security follows immediately from the GSW scheme. We will use the HEPerm scheme to implement the bootstrapping algorithm, allowing homomorphic decryption. To do so, we need to compute the inner product $\langle s, c \rangle$ and the rounding function homomorphically. This requires two operations on HEPerm ciphertexts: composition and equality test. Remember that normal matrix multiplication of permutation matrices corresponds to addition of the underlying integers, mod r (see Section 3.4). The composition operation of the HEPerm scheme is a matrix multiplication of the HEPerm ciphertexts, using $\boxplus$ and $\boxdot$ for addition and multiplication of GSW ciphertexts. The composition of two HEPerm ciphertexts corresponds to modular addition of the unencrypted integers. Equality test is an operation used to implement the rounding function. It pick out one of the GSW ciphertexts in the HEPerm ciphertext, returning a GSW ciphertext designed to encrypt 1 if and only if the encrypted integer is equal to the integer we are testing for. More formally, the operations are defined as follows:

Definition 37 (HEPerm composition)**.** Define the map $\boxcircle\colon \mathcal{Y}^r \times \mathcal{Y}^r \to \mathcal{Y}^r$ as a right associative matrix multiplication where the addition operation is replaced by $\boxplus$ and the multiplication operation is replaced by $\boxdot$. More precisely, take the ration-expansion of the input HEPerm ciphertexts and denote them $\mathbf{C}^\pi = (c^\pi_{i,j} \in \mathcal{Y}), \mathbf{C}^\sigma = (c^\sigma_{i,j} \in \mathcal{Y}) \in \mathcal{Y}^{r\times r}$ element $c_{i,j}$ of $\mathbf{C}^\pi \boxcircle \mathbf{C}^\sigma$ is defined as

$$c_{i,j} = \boxplus_{l\in[r]} c^\pi_{i,l} \boxdot c^\sigma_{l,j} \tag{8.4}$$

Remark. Note that homomorphic addition of integers is denoted by a square with a ring inside using HEPerm ciphertexts whereas homomorphic multiplication of bits is denoted by a square with a dot inside using GSW ciphertexts.

We now argue that $\mathbf{C}^\pi \boxcircle \mathbf{C}^\sigma$ preserves the design, meaning that we argue that the first column $(c_{i,1})_{i\in[r]} \in \mathcal{Y}^r$ of $\mathbf{C}^\pi \boxcircle \mathbf{C}^\sigma$ is designed to encrypt $\pi \circ \sigma$. Note that every GSW ciphertext element in $(c_{i,1})_{i\in[r]}$ is an inner product of two r dimensional vectors (row i of $\mathbf{C}^\pi$ and column 1 of $\mathbf{C}^\sigma$) of GSW ciphertexts. Since $\boxdot$ and $\boxplus$ preserves design, every element preserves design and therefore $\boxcircle$ preserves design too.

Notice that the first column of $\mathbf{C}^\pi$ contains the encryption of π. This suggests that it is unnecessary to expand π into its permutation matrix, saving us $r^2 - r$ GSW encryptions and $(r-1)rn^2 l \log q$ bits of space per HEPerm ciphertext. However, the composition operation requires the left HEPerm ciphertext to be square: it is impossible to expand a column ciphertext designed to encrypt an integer correctly as there is no way to know what the integer is. On the other hand, if we are told what integer the ciphertext encrypts, then the first column of the HEPerm ciphertext is sufficient. We can use this fact to implement the equality test operation by choosing exactly one GSW ciphertext in the first column of the HEPerm ciphertext, as seen below.

Definition 38 (Equality test)**.** Define the map $\mathrm{Eq?}\colon \mathcal{Y}^r \times C_r \to \mathcal{Y}$ as follows. Given an HEPerm column ciphertext $\mathbf{C} = (c_1, \dots, c_r) \in \mathcal{Y}^r$ designed to encrypt a cyclic shift and an unencrypted cyclic shift $\pi = (b_1, \dots, b_r) \in \{0,1\}^r$ where $b_i = 1$ for exactly one $i \in [r]$, return c_i. In other words

$$\mathrm{Eq?}(\mathbf{C}, \pi) = c_i \in \mathcal{Y} \tag{8.5}$$

Notice that the equality test function Eq? returns a GSW ciphertext designed to encrypt 1 if and only if $\mathbf{C}$ is designed to encrypt π. In other words, the equality test function takes a HEPerm ciphertext encrypting some cyclic shift and returns a GSW ciphertext designed to encrypt 1 if and only if it also receives the correct cyclic shift. We use the equality test function to implement the rounding function homomorphically by testing if the encrypted rounding function input (represented by $\mathbf{G}$) is designed to encrypt an integer that is supposed to be rounded to 1. If and only if that is the case does the equality test function output a GSW ciphertext designed to encrypt 1.

We now show that the HEPerm composition operation $\boxcircle$ exhibits very similar error growth to the GSW composition operation $\boxdot$. In the following proof, denote $\mathbf{J}$ as $(J_{i,j}) \in \mathcal{Y}^{r\times r}$.

Lemma 5. Let $\mathbf{C}^\sigma = (c^\sigma_{i,j}) \in \mathcal{Y}^{r\times r}$ be a HEperm ciphertext designed to encrypt the cyclic shift $\sigma \in C_r$ with associated error $\mathbf{E}^\sigma = (e^\sigma_{i,j}) \in \mathcal{E}^{r\times r}$. Then, $\mathbf{C}^\sigma \boxcircle \mathbf{J}$ is designed to encrypt σ and its associated error is $\mathbf{E}^\sigma G^{-1}(\mathbf{J}) \in \mathcal{E}^{r\times r}$. Furthermore, row i is subgaussian vector over $\mathcal{E}^r$ with parameter $O(\|e^\sigma_i\|)$ where e^σ_i is the ith row of $\mathbf{E}^\sigma$.

Proof. The fact that $\mathbf{C}^\sigma \boxcircle \mathbf{J}$ preserves design follows immediately from the fact that $\boxdot$ and $\boxplus$ preserves design, the definition of $\boxcircle$ and the fact that $\mathbf{J}$ is designed to encrypt the identity cyclic shift. By Equation 8.4, the GSW ciphertext on row i and column j of $\mathbf{C}^\sigma \boxcircle \mathbf{J}$ is $\bigboxplus_{l\in[r]} c^\sigma_{i,l} \boxdot J_{l,j}$. Since $J_{l,j}$ contains zero errors, the invariant equation on the product 8.2 yields that the error for each GSW ciphertext $c^\sigma_{i,l} \boxdot J_{l,j}$ is $(e^\sigma_{i,l})^T G^{-1}(J_{l,j}) \in \mathcal{E}$. The total error associated with $\bigboxplus_{l\in[r]} c^\sigma_{i,l} \boxdot J_{l,j}$ is therefore $\sum_{l=1}^{r} (e^\sigma_{i,l})^T G^{-1}(J_{l,j})$. By lemma 1, $(e^\sigma_{i,l})^T G^{-1}(J_{l,j})$ is subgaussian with parameter $O(\|e^\sigma_{i,l}\|)$. Addition of subgaussian vectors from Lemma 2 gives that the total error is subgaussian with parameter $O(\|e^\sigma_i\|)$ where e^σ_i is the ith row of $\mathbf{E}^\sigma$. The same analysis applies to every entry in the error associated with $\mathbf{C}^\sigma \boxcircle \mathbf{J}$ and thus the error is $\mathbf{E}^\sigma G^{-1}(\mathbf{J}) \in \mathcal{E}^{r\times r}$. Furthermore, row i is a subgaussian vector over $\mathcal{E}^r$ with parameter $O(\|e^\sigma_i\|)$ since every entry is subgaussian with parameter $O(\|e^\sigma_i\|)$. $\square$

Theorem 9. *Let $\mathbf{C}^\pi = (c^\pi_{i,j}), \mathbf{C}^\sigma = (c^\sigma_{i,j}) \in \mathcal{Y}^{r\times r}$ be two HEperm ciphertexts designed to encrypt the cyclic shifts $\pi, \sigma \in C_r$ with associated errors $\mathbf{E}^\pi = (e^\pi_{i,j}), \mathbf{E}^\sigma = (e^\sigma_{i,j}) \in \mathcal{E}^{r\times r}$ respectively. Then, $\mathbf{C}^\pi \boxcircle \mathbf{C}^\sigma \boxcircle \mathbf{J}$ is designed to encrypt $\pi \circ \sigma \in C_r$ and its associated error is*

$$\mathbf{E}^\pi G^{-1}(\mathbf{C}^\sigma \boxcircle \mathbf{J}) + \mathbf{P}^\pi \mathbf{E}^\sigma G^{-1}(\mathbf{J}) \in \mathcal{E}^{r\times r} \tag{8.6}$$

where $\mathbf{P}^\pi = (p^{pi}_i, j) \in \{0,1\}^{r\times r}$ is the permutation matrix corresponding to π. Furthermore, row i is subgaussian vector over $\mathcal{E}^r$ with parameter $O(\|e_i\|)$ where $e_i \in \mathcal{E}^{2r}$

is the ith row of the concatenated matrix $[\mathbf{E}^\pi|\mathbf{E}^\sigma]$.

Proof. Remember that $\boxdot$ is right associative. Therefore, $\mathbf{C}^\pi$ acts on the HEPerm ciphertext $\mathbf{C}^{right} \overset{\text{def}}{=} \mathbf{C}^\sigma \boxdot \mathbf{G} \in \mathcal{Y}^{r\times r}$ with error $\mathbf{E}^{right} \overset{\text{def}}{=} \mathbf{E}^\sigma G^{-1}(\mathbf{J})$. For the HEPerm ciphertext , consider the GSW ciphertext element on row i and column j. Unlike the above proof where $\mathbf{J}$ contained zero errors, here the right hand ciphertext $\mathbf{C}^{right}$ does contain errors. Therefore, the error associated with the GSW ciphertext on row i, column j of $\mathbf{C}^\pi \boxdot \mathbf{C}^\sigma \boxdot \mathbf{J}$ is, by 8.1 and 8.2 along row i of $\mathbf{C}^\pi$ and column j of $\mathbf{C}^{right}$, given by

$$\sum_{l=1}^{r} (e^\pi_{i,l})^T G^{-1}(c^{right}_{l,j}) + p^\pi_{i,l} e^{right}_{l,j}$$

where the bit $p^\pi_{i,l}$ is 1 for exactly one l, namely $l = \pi^{-1}(i)$. In other words, the error for the entry on row i, column j is $\sum_{l=1}^{r} (e^\pi_{i,l})^T G^{-1}(c^{right}_{l,j}) + e^{right}_{\pi^{-1}(i),j}$ By the same argument as before, the first term (i.e., the sum over l) is subgaussian with parameter $O(\|e^\pi_i\|)$ where e^π_i is the ith row of $\mathbf{E}^\pi$. The second term is, by Lemma 5, subgaussian with error $O(\|e^\sigma_{\pi^{-1}(i)}\|)$ where $e^\sigma_{\pi^{-1}(i)}$ is the $\pi^{-1}(i)$th row of $\mathbf{E}^\sigma$. By the result on addition of subgaussian vectors, Lemma 2, the error associated with the GSW ciphertext on row i, column j is therefore $O(\|e_i\|)$. The same analysis applies to every entry in the error associated with $\mathbf{C}^\pi \boxdot \mathbf{C}^\sigma \boxdot \mathbf{J}$ and thus the error is $\mathbf{E}^\pi G^{-1}(\mathbf{C}^\sigma \boxdot \mathbf{J}) + \mathbf{P}^\pi \mathbf{E}^\sigma G^{-1}(\mathbf{J}) \in \mathcal{E}^{r\times r}$. Furthermore, row i is a subgaussian vector over $\mathcal{E}^r$ with parameter $O(\|e_i\|)$ since every entry in that row is subgaussian with parameter $O(\|e_i\|)$. □

Corollary 2. Let $\mathbf{C}_1, \ldots, \mathbf{C}_k$ be HEperm ciphertext designed to encrypt cyclic shifts $\pi_1, \ldots, \pi_k$ with associated errors $\mathbf{E}_1, \ldots, \mathbf{E}_k \in \mathcal{E}^{r\times r}$, respectively. Then,

$$\mathbf{C}_1 \boxdot \ldots \boxdot \mathbf{C}_k \boxdot \mathbf{J} \tag{8.7}$$

is designed to encrypt $\pi_1 \circ \cdots \circ \pi_k$. Furthermore, the ith row of its associated error is subgaussian with parameter $O(\|e_i\|)$ where $e_i \in \mathcal{E}^{kr}$ is the ith row of the concatenated matrix $[\mathbf{E}_1|\ldots|\mathbf{E}_k]$.

This means that if you have a chain of k fresh HEPerm ciphertexts, each containing GSW ciphertexts with the same error parameter $s\sqrt{m}$, then the error associated with the final HEPerm ciphertext is a error matrix where each entry is subgaussian with parameter $O(s\sqrt{mkr})$. Notice the similarity to the analysis of Corollary 1 for the GSW scheme. We will use the same reasoning to analyze the final error of the ciphertext returned by the bootstrap algorithm in the next section.

8.4 Bootstrapping

In this section we show how to use the HEPerm scheme to bootstrap the GSW scheme, using the HEPerm composition operation and the Equality test operation. Remember that bootstrapping algorithm takes as input a noisy ciphertext with an encryption of a secret key and outputs a new ciphertext with smaller noise, by decrypting the ciphertext homomorphically (meaning using the encrypted the secret key instead of the real secret key). Technically, the bootstrapping procedure consists of two algorithms, BootGen() which is run by the client to generate an encryption of the secret key and Bootstrap() which is run by the server for homomorphic decryption, using the encrypted secret key. BootGen() returns a so called bootstrap key, bk, that is a matrix of HEPerm ciphertexts. Element i of the secret key is encrypted and represented by column i of bk. The server computes the inner product and the rounding function homomorphically. The inner product is implemented by using the HEPerm composition operation and the rounding function is implemented by using the equality test operation.

Before presenting the algorithms, let us first consider the decryption process. Decryption starts with calculating the inner product of the secret key, $s \in \mathbb{Z}_q$ with the penultimate column of the ciphertext $c_{pen} \in \mathbb{Z}_q$. Remark that if c_{pen} is defined over $\{0,1\}$, then this inner product becomes a subset sum over $(s_1, \ldots, s_n)$. As we will see later, this would allow us to calculate the inner product homomorphically using the HEPerm scheme. The idea is therefore to convert c_{pen} to binary using the ordinary bit decomposition of c_{pen}, denoted $c'_{pen} \in \{0,1\}^{nl}$. Remark that c'_{pen} satisfies $\mathbf{G}c'_{pen} = c_{pen}$. To make the dimensions match, we consider the vector $s' \overset{\text{def}}{=} G^T s \in \mathbb{Z}_q^{nl}$. Then, we have that $\left\langle s', c'_{pen} \right\rangle = s'^T c'_{pen} = (\mathbf{G}^T s)^T c'_{pen} = s^T \mathbf{G} c'_{pen} = s^T c_{pen} = \langle s, c_{pen} \rangle$. Therefore, we can calculate the inner product homomorphically by calculating $\left\langle s', c'_{pen} \right\rangle$ homomorphically. Instead of using the encryption of the normal secret key s, the bootstrapping algorithm uses the encryption of s' using the HEPerm scheme. In other words, the client runs the BootGen() algorithm using the modified secret key $s' \in \mathbb{Z}_q^{nl}$ whereas the server runs Bootstrap() using the modified ciphertext $c'_{pen} \in \{0,1\}^{nl}$.

This section is heavily based on the theory presented in Section 3.4. In particular, remember that we can choose modulus $q = \prod_{i \in [t]} r_i$ such that every integer mod q can be represented by $t = O(\frac{\log q}{\log \log q})$ cyclic shifts, where each $r_i = O(\log q)$ is a prime power.

Also remember that for an integer $s_j \in \mathbb{Z}_q$, $\phi_i(s_j)$ is the cyclic shift representing the integer $s_j \pmod{r_i}$. As a last note, the bootstrapping algorithm presented in [AP13] applies to all LWE based schemes whereas in this thesis, we will look specifically at the GSW scheme and therefore simplify the notation.

Definition 39. (BootGen) Define the BootGen algorithm taking the secret key $s' = (s'_1, \dots, s'_{nl}) \in \mathbb{Z}_q^{nl}$ and the public key $\mathbf{A}'$ of the GSW scheme as the following $t \times nl$ sized matrix of HEPerm ciphertexts

$$bk = \begin{bmatrix} \mathbf{C}_{1,1} & \cdots & \mathbf{C}_{1,nl} \\ \vdots & \ddots & \vdots \\ \mathbf{C}_{t,1} & \cdots & \mathbf{C}_{t,nl} \end{bmatrix} \tag{8.8}$$

where $\mathbf{C}_{i,j} \leftarrow \text{HEPerm.Enc}(\mathbf{A}', \phi_i(s'_j))$ for $i \in [t]$ and $j \in [nl]$.

Notice that $\mathbf{C}_{i,j}$ is an HEPerm encryption of $s'_j \pmod{r_i}$ and that row i therefore contains $s'_j \pmod{r_i}$ for every s'_j in the modified secret key. Furthermore, column j is an encryption of the canonical embedding of s'_j. Since every HEPerm ciphertext is a square matrix of GSW ciphertexts, every column of HEPerm ciphertexts contains $\sum\limits_{i \in t} r_i^2$ GSW ciphertexts for a total of $nl \sum\limits_{i \in t} r_i^2$ GSW ciphertexts in bk. Remark that in practice, this would be reduced to $nl \sum\limits_{i \in t} r_i$ GSW ciphertext by instead letting the every HEPerm ciphertext consist of a column of r_i ciphertexts. However, when computing the inner product homomorphically, $\boxdot$ requires the square representation with r_i^2 GSW ciphertexts. The server can expand the column by letting each new column be a cyclically rotated version of the first column.

Inner product

We start by discussing how to calculate the inner product normally (meaning when the secret key is not encrypted) using cyclic shifts. Note that inner product is equivalent to calculating a subset sum of s' since $\langle s, c_{pen} \rangle = \left\langle s', c'_{pen} \right\rangle = \sum\limits_{\{j \in [nl] : c'_{pen,j} = 1\}} s'_j$. By using Equation 3.1, we can calculate this sum with the canonical homomorphism by considering compositions of cyclic shifts. More specifically, we can consider each moduli r_i separately and calculate the integer $\sum\limits_{\{j \in [nl] : c'_{pen,j} = 1\}} s'_j \pmod{r_i}$ as the the corresponding cyclic shift $\bigcirc\limits_{\{j \in [nl] : c'_{pen,j} = 1\}} \phi_i(s'_j) \in S_{r_i}$. Repeating for every i yields a tuple of t cyclic shifts that together uniquely represents the subset sum (and thus also the inner product). Now, we want to do the same operation homomorphically using the bootstrap key bk. Note that row i in bk represents HEPerm ciphertexts of $s'_j \pmod{r_i}$ for every $j \in [nl]$. Therefore, to calculate the inner product homomorphically, the idea is to pick out the columns j such that $c'_{pen,j} = 1$ and calculate the HEPerm composition of ciphertext. Since HEPerm preserves design, the resulting HEPerm ciphertext is designed to encrypt $\sum\limits_{\{j \in [nl] : c'_{pen,j} = 1\}} s'_j$ $\pmod{r_i}$. More formally, the inner product is a column vector with t HEPerm ciphertexts where entry $\mathbf{C}_i$ is defined as the following HEPerm composition

$$\mathbf{C}_i = \mathop{\Large\boxcircle}_{\{j \in [nl] : c'_{pen,j} = 1\}} \mathbf{C}_{i,j} \boxcircle \mathbf{J} \in \mathcal{Y}^{r_i} \tag{8.9}$$

where $\mathbf{C}_{i,j}$ is the HEPerm ciphertext on row i, column j of bk which is designed to encrypt cyclic shift $\phi_i(s_j)$, meaning integer $s_j \pmod{r_i}$. We include the zero error HEPerm ciphertext $\mathbf{J}$ to allow for Corollary 2 in the error analysis later.

Consider the quick example where $c'_{pen} = (1, 0, 0, \ldots, 0, 0, 1) \in \{0, 1\}^{nl}$. Then, then the encrypted inner product is $(\mathbf{C}_{1,1} \boxcircle \mathbf{C}_{1,nl} \boxcircle \mathbf{J}, \mathbf{C}_{2,1} \boxcircle \mathbf{C}_{2,nl} \boxcircle \mathbf{J}, \ldots, \mathbf{C}_{t,1} \boxcircle \mathbf{C}_{t,nl} \boxcircle \mathbf{J})^T$ where $\mathbf{C}_{i,1} \boxcircle \mathbf{C}_{i,nl} \boxcircle \mathbf{J}$ is the HEPerm ciphertext designed to encrypt $\langle s, c \rangle \pmod{r_i}$

Rounding function

Denote the encrypted inner product calculated using Equation 8.9 by α. Since α is designed to encrypt $\langle s, c_{pen} \rangle$, to compute the decryption function homomorphically, we need to compute the rounding function on α. The Equality test function takes the encrypted inner product α and a cyclic shift representation of an integer $x \in \mathbb{Z}_q$ such that $[x]_2 = 1$. The idea is to pick out the t GSW ciphertexts in the t HEperm ciphertexts in α that corresponds to the integer x and then use $\boxdot$ operation to multiply the t GSW ciphertexts together. This results in a GSW ciphertext product designed to encrypt 1 if and only if every chosen GSW ciphertext is designed to encrypt one, which is the case if and only if α is designed to encrypt x. If any of the t GSW ciphertexts is not designed to encrypt the corresponding cyclic shift in $\phi(x)$, then the product ciphertext is designed to encrypt 0, meaning α is not designed to encrypt x. Note however that α can be designed to encrypt some other integer rounded to 1. Therefore, it is necessary to run the equality test function on every "good" integer x, where "good" means it is rounded to 1, and take the sum of all the resulting GSW ciphertexts. The sum is then designed to encrypt 1 if and only if α is designed to encrypt any one of the good integers and designed to encrypt 0 otherwise. More formally, the rounding function can be implemented homomorphically as follows:

$$[\alpha]_2 = \boxplus_{\{x \in \mathbb{Z}_q : [x]_2 = 1\}} \Big(\mathrm{Eq?}(\mathbf{C}_1, \phi_1(x)) \boxdot \cdots \boxdot \mathrm{Eq?}(\mathbf{C}_t, \phi_t(x)) \boxdot \mathbf{G} \Big) \in \mathcal{Y} \tag{8.10}$$

We include the zero error GSW ciphertext $\mathbf{G}$ to allow for Corollary 1 in the error analysis later. $[\alpha]_2$ is the refreshed GSW ciphertext of the original GSW ciphertext and is the output value of the bootstrap algorithm. In other words, $[\alpha]_2 \leftarrow \mathrm{Bootstrap}(bk, c'_{pen})$ where generating bk is preprocessing made by the client and calculating c'_{pen} from the original ciphertext is a preprocessing step made by the server. It is clear by the arguments above that $[\alpha]_2$ indeed is designed to encrypt the same bit that the original ciphertext encrypts.

So far, we have only talked about how computing the inner product and the rounding function homomorphically preserves the design of the original ciphertext. In other words, $[\alpha]_2$ decrypts to the same bit as the original ciphertext assuming error are small enough. However, in reality the the bootstrapping returns another noisy ciphertext as a result of the computations on the encrypted secret key. More specifically, when calculating the inner product homomorphically, the HEPerm ciphertexts in each row of the bootstrap key is composed using $\boxcircle$ for a maximum of nl times. Furthermore, when calculating the rounding function, there are $O(q)$ sums of GSW ciphertext where each is a product of t GSW ciphertexts. Below, we analyze the error growth in the bootstrapping algorithm and later show how to choose parameters such that the error associated to the refreshed ciphertext is small enough to result in correct decryption.

8.5 Errors in bootstrapped ciphertext

In the following section, we denote parameter in the subgaussian, discrete error distribution χ as s. Furthermore, we let r denote the sum $\sum_{i \in t} r_i$.

Lemma 6 (Error of homomorphic innerproduct)*.* Let $\mathbf{C}_i$ denote the HEPerm ciphertext designed to encrypt $\langle s, c_{pen} \rangle \pmod{r_i}$ for $i \in [t]$, as defined by Equation 8.9. Then, every row in the error $\mathbf{E}_i \in \mathcal{E}^{r_i}$ associated with $\mathbf{C}_i$ is subgaussian with parameter $O(snl\sqrt{mr_i})$. Furthermore, every row has length bounded by $O(s(nl)^{\frac{3}{2}}\sqrt{r_i})$ except for probability $2^{-\Omega(nl)}$.

Proof. Remember that $\mathbf{C}_{i,j}$ contains r_i GSW ciphertext with r_i associated errors in $\mathcal{E} = \mathbb{Z}^{nl}$. Since each GSW ciphertext in $\mathbf{C}_{i,j}$ is a fresh GSW encryption, we remember that each associated error contains subgaussian random variables over $\mathbb{Z}$ with parameter $O(s\sqrt{m})$. To bound the error growth, assume that $c'_{pen} \in \{0,1\}^{nl}$ is the vector with all 1, meaning that the homomorphic inner product consists of exactly nl $\boxdot$ operations in Equation 8.9. Concatenating all nl rotation expanded error matrices yields a matrix in $\mathcal{E}^{r_i \times r_i nl} = \mathbb{Z}^{r_i \times r_i n^2 l^2}$. Take any row x in this concatenated error vector. Then $x \in \mathbb{Z}^{r_i n^2 l^2}$ and by Theorem 2, $\|x\|$ is $O(snl\sqrt{mr_i})$ except for probability $2^{-\Omega(n^2 l^2 r_i)}$. Therefore, by Corollary 2, every row of the error associated with $\mathbf{C}_i$ is subgaussian with parameter $O(\|x\|) = O(snl\sqrt{mr_i})$. Futhermore, since any row of $\mathbf{E}_i \in \mathcal{E}^{r_i}$ is defined over $\mathbb{Z}^{nl}$, invoking Theorem 2 again yields that the length over the integers of any row vector in the error matrix is $O(s(nl)^{\frac{3}{2}}\sqrt{mr_i})$ except for probability $2^{-\Omega(nl)}$. Since $2^{-\Omega(nl)}$ dominates $2^{-\Omega(n^2 l^2 r_i)}$, the proof is complete. □

Theorem 10 (Error of bootstrapped ciphertext)**.** *The error associated with the GSW ciphertext* $[\alpha]_2$ *in Equation 8.10 is subgaussian with parameter* $O(s(nl)^{\frac{3}{2}}\sqrt{mrq})$ *except for probability* $2^{-\Omega(nl)}$.

Proof. By the previous Lemma, the error associated with each $\mathbf{C}_i$ has subgaussian row vectors with length over the integers bounded by $O(s(nl)^{\frac{3}{2}}\sqrt{mr_i})$. Consider the concatenated vector $(e_1, \ldots, e_t) \in \mathbb{Z}^{tnl}$ where $e_i \in \mathbb{Z}^{nl}$ is any row in error matrix $\mathbf{E}_i$. Then, this concatenated vector has length $\sqrt{\|e_1\|^2 + \cdots + \|e_t\|^2}$. By Corollary 1, the error associated with the GSW ciphertext product Eq?$(\mathbf{C}_1, \phi_1(x)) \boxdot \cdots \boxdot$ Eq?$(\mathbf{C}_t, \phi_t(x)) \boxdot \mathbf{G}$ is subgaussian with parameter $O(\sqrt{\|e_1\|^2 + \cdots + \|e_t\|^2}) = O(\sqrt{(s(nl)^{\frac{3}{2}}\sqrt{mr_1})^2 + \cdots + (s(nl)^{\frac{3}{2}}\sqrt{mr_t})^2}) = O(s(nl)^{\frac{3}{2}}\sqrt{mr})$. Since the rounding function computes the sum of q GSW ciphertexts in Equation 8.10, additivity of subgaussian vectors from Lemma 2 implies that the error associated with the final bootstrapped GSW ciphertext is subgaussian with parameter $O\Big(\sqrt{\sum_{i=1}^{q}(s(nl)^{\frac{3}{2}}\sqrt{mr})^2}\Big) = O(s(nl)^{\frac{3}{2}}\sqrt{mrq})$ □

8.6 Parameters

Our goal in this section is to show that there exists choice of parameters n, q, χ, m such that the GSW scheme is secure and decryption of ciphertexts are correct, including the bootstrapped ciphertext. We have chosen to present the GSW scheme with the focus on conceptual simplicity: as a result, our analysis require stronger hardness assumptions and larger parameters than necessary. To be more specific, we have chosen not to dimension-modulus reduce with the techniques discussed in [BV11] when calculating the homomorphic inner product. This allows us to use the same lattice dimension[1] n and the same modulus q for the bootstrapping algorithm as for the original GSW scheme. However, this has implications on the security and parameters of the scheme in contrast to the results presented in [AP13], which does use dimension-modulus reduction techniques.

- Security: We assume that the GapSVP problem is hard to approximate on n dimensional lattices within factors poly(n). This would imply 2^λ hardness of LWE. To justify this assumption, GapSVP is a well studied problem and there are no known sub-exponential time algorithms that can achieve polynomial approximation factors. Unfortunately, prioritizing simplicity means our assumption is stronger than the one used in [AP13] which only requires hardness to approximate within factors $\tilde{O}(\lambda^3)$.
- Parameter size: Since we do not utilize modulus reduction techniques, the bootstrapped ciphertexts has error with parameter that grows with $\sqrt{q}$, where q is the modulus of the GSW scheme. This means that an increase to the modulus also yields an increases to the potential magnitude of the error. To compensate for this, we will need to choose $q = \Theta(\lambda^6)$ whereas [AP13] does not have this issue and can choose $q = \tilde{\Theta}(\lambda^{\frac{5}{2}})$.

Initializing the scheme

All known algorithms for solving GapSVP on n dimensional lattices require $2^{\Omega(n)}$ time. To achieve 2^λ hardness of LWE, we require lattice dimension $n = \Omega(\lambda)$. Let $n = \lambda$. For the discrete subgaussian error distribution, by the security results from [Reg05], it is necessary to choose width parameter strictly greater than 2 times the square root of the lattice dimension. Therefore, we choose $s = 3\sqrt{n} = \Theta(\sqrt{n})$. For the magnitude of the LWE samples, m, we want there to be enough samples to be confident there is only one solution for the noisy system. Note that if we have less than n samples, then there are more equations than variables and there are many solutions to the system. However, choosing exactly n samples only guarantees unique solution if and only if all the samples are linearly independent. Therefore, it is customary to pick some m slightly larger than n. As explained in [Pei15] the

[1]technically we swapped lattice dimension from n to nl to get binary ciphertext column

exact value of m makes no difference in the hardness of the scheme. Similar to standard literature, we choose some $m = \Theta(n \log q)$. Note that, thus far, there is nothing special about our parameter choices in comparison to standard parameter choices in the literature. However, for the modulus, it is necessary for its magnitude to be sufficiently large to allow correct decryption of the refreshed ciphertext. In other words, we need to choose q such that the error parameter associated with the refreshed ciphertext is small enough to result in correct decryption, except for negligible probability with respect to the security parameter.

Choosing the modulus

If we substitute the parameter choices for s and m above into the error bound of the bootstrapped ciphertext, we get that the error parameter of the penultimate entry is $O(s(nl)^{\frac{3}{2}}\sqrt{mrq}) = O(\sqrt{n} \cdot n^{\frac{3}{2}} \log^{\frac{3}{2}} q \cdot \sqrt{n \log q} \cdot \sqrt{r} \cdot \sqrt{q}) = O(n^{\frac{5}{2}}\sqrt{qr} \log^2 q)$. The question now is how to choose a sufficiently large modulus q such that this error has a negligible probability of decrypting incorrectly. Remember that to avoid ”wrapping around” the modulus, we need to ensure that the magnitude of the error has a negligible probability of exceeding $q/8$. Let p denote the subgaussian error parameter $n^{\frac{5}{2}}\sqrt{qr}\log^2 q$. If our modulus satisfies $q = \tilde{\Omega}(p\sqrt{\lambda})$, then the probability of incorrect decryption is negligible with respect to λ. This follows from the definition of subgaussian random variables as $\Pr[|e_{pen}| > q/8] \leq 2\exp\left(-\pi \frac{q^2}{64p^2}\right) \leq 2\exp(-c\lambda) = \text{negl}(\lambda)$ for some constant $c > 0$.

As described in Section 3.4, we want to choose q as a product of the maximal prime powers bounded by some r. We can choose a pseudo-constant r such that our modulus q satisfies the above. Let $r = \log(\lambda^6)$. Then, $r = \tilde{O}(1)$ and more importantly, by Theorem 5, setting q as the product of the maximal prime powers bounded by r yields a modulus $q = \Theta(\lambda^6)$. What remains to be shown is that this modulus is sufficiently large to handle the error. In other words, we need to show $q = \tilde{\Omega}(p\sqrt{\lambda})$. Substituting $q = \Theta(\lambda^6)$ and $n = \lambda$ into the expression gives

$$\begin{aligned} p\sqrt{\lambda} &= n^{\frac{5}{2}}\sqrt{qr}\log^2 q\sqrt{\lambda} = \tilde{\Theta}(\lambda^{\frac{5}{2}}\sqrt{\lambda^6}\sqrt{\lambda}) \\ &= \tilde{\Theta}(\lambda^6) \end{aligned}$$

To be concrete, our parameter choices are as follows:

- $n = \lambda$

- $q = \Theta(\lambda^6)$ such that $q = \prod_{r_i \in \mathrm{MPP}(r)} r_i$ where $r = \log(\lambda^6)$
- $s = 3\sqrt{n} = \Theta(\sqrt{n})$
- $m = n \log q = \Theta(n \log q)$

Bibliography

[ABD16] Albrecht, Martin, Bai, Shi, and Ducas, Léo. *A subfield lattice attack on overstretched NTRU assumptions: Cryptanalysis of some FHE and Graded Encoding Schemes*. Cryptology ePrint Archive, Paper 2016/127. `https://eprint.iacr.org/2016/127`. 2016. URL: `https://eprint.iacr.org/2016/127`.

[AP13] Alperin-Sheriff, Jacob and Peikert, Chris. *Practical Bootstrapping in Quasilinear Time*. Cryptology ePrint Archive, Paper 2013/372. `https://eprint.iacr.org/2013/372`. 2013. URL: `https://eprint.iacr.org/2013/372`.

[App+09] Applebaum, Benny, Cash, David, Peikert, Chris, and Sahai, Amit. "Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems". In: *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference*. Vol. 5677. Lecture Notes in Computer Science. Springer, 2009, pp. 595–618. DOI: `10.1007/978-3-642-03356-8_35`. URL: `https://www.iacr.org/archive/crypto2009/56770585/56770585.pdf`.

[AB09] Arora, Sanjeev and Barak, Boaz. *Computational Complexity: A Modern Approach*. 1st. New York, NY, USA: Cambridge University Press, 2009.

[Bra12] Brakerski, Zvika. *Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP*. Cryptology ePrint Archive, Paper 2012/078. `https://eprint.iacr.org/2012/078`. 2012. URL: `https://eprint.iacr.org/2012/078`.

[Bra18] Brakerski, Zvika. "Fundamentals of Fully Homomorphic Encryption - A Survey". In: *Electronic Colloquium on Computational Complexity* 125 (2018).

[BGV14] Brakerski, Zvika, Gentry, Craig, and Vaikuntanathan, Vinod. "(Leveled) Fully Homomorphic Encryption Without Bootstrapping". In: *ACM Transactions on Computation Theory (TOCT)* 6.3 (2014). Preliminary version in ITCS 2012, p. 13.

[Bra+13] Brakerski, Zvika, Langlois, Adeline, Peikert, Chris, Regev, Oded, and Stehlé, Damien. *Classical Hardness of Learning with Errors*. 2013. arXiv: `1306.0281 [cs.CC]`.

[BV11] Brakerski, Zvika and Vaikuntanathan, Vinod. *Efficient Fully Homomorphic Encryption from (Standard) LWE*. Cryptology ePrint Archive, Paper 2011/344. `https://eprint.iacr.org/2011/344`. 2011. URL: `https://eprint.iacr.org/2011/344`.

[Che+18] Cheon, Jung Hee, Han, Kyoohyung, Kim, Andrey, Kim, Miran, and Song, Yongsoo. *Bootstrapping for Approximate Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2018/153. `https://eprint.iacr.org/2018/153`. 2018. URL: `https://eprint.iacr.org/2018/153`.

[Che+16] Cheon, Jung Hee, Kim, Andrey, Kim, Miran, and Song, Yongsoo. *Homomorphic Encryption for Arithmetic of Approximate Numbers*. Cryptology ePrint Archive, Paper 2016/421. `https://eprint.iacr.org/2016/421`. 2016. URL: `https://eprint.iacr.org/2016/421`.

[Chi+18] Chillotti, Ilaria, Gama, Nicolas, Georgieva, Mariya, and Izabachène, Malika. *TFHE: Fast Fully Homomorphic Encryption over the Torus*. Cryptology ePrint Archive, Paper 2018/421. `https://eprint.iacr.org/2018/421`. 2018. URL: `https://eprint.iacr.org/2018/421`.

[CNT11] Coron, Jean-Sebastien, Naccache, David, and Tibouchi, Mehdi. *Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers*. Cryptology ePrint Archive, Paper 2011/440. `https://eprint.iacr.org/2011/440`. 2011. URL: `https://eprint.iacr.org/2011/440`.

[CLT13] Coron, Jean-Sébastien, Lepoint, Tancrède, and Tibouchi, Mehdi. *Batch Fully Homomorphic Encryption over the Integers*. Cryptology ePrint Archive, Paper 2013/036. `https://eprint.iacr.org/2013/036`. 2013. URL: `https://eprint.iacr.org/2013/036`.

[Dij+09] Dijk, Marten van, Gentry, Craig, Halevi, Shai, and Vaikuntanathan, Vinod. *Fully Homomorphic Encryption over the Integers*. Cryptology ePrint Archive, Paper 2009/616. `https://eprint.iacr.org/2009/616`. 2009. URL: `https://eprint.iacr.org/2009/616`.

[DM14] Ducas, Léo and Micciancio, Daniele. *FHEW: Bootstrapping Homomorphic Encryption in less than a second*. Cryptology ePrint Archive, Paper 2014/816. `https://eprint.iacr.org/2014/816`. 2014. URL: `https://eprint.iacr.org/2014/816`.

[FV12] Fan, Junfeng and Vercauteren, Frederik. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2012/144. `https://eprint.iacr.org/2012/144`. 2012. URL: `https://eprint.iacr.org/2012/144`.

[Gam+14] Gama, Nicolas, Izabachene, Malika, Nguyen, Phong Q., and Xie, Xiang. *Structural Lattice Reduction: Generalized Worst-Case to Average-Case Reductions and Homomorphic Cryptosystems*. Cryptology ePrint Archive, Paper 2014/283. `https://eprint.iacr.org/2014/283`. 2014. URL: `https://eprint.iacr.org/2014/283`.

[Gar23] Gartner. *Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach Nearly $600 Billion in 2023*. Accessed: 2023-06-07. 2023. URL: `https://www.gartner.com/en/newsroom/press-releases/2023-04-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-600-billion-in-2023`.

[Gen09] Gentry, Craig. “A Fully Homomorphic Encryption Scheme”. AAI3382729. PhD thesis. Stanford, CA, USA: Stanford University, 2009.

[GH11] Gentry, Craig and Halevi, Shai. *Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits*. Cryptology ePrint Archive, Paper 2011/279. `https://eprint.iacr.org/2011/279`. 2011. URL: `https://eprint.iacr.org/2011/279`.

[GH10] Gentry, Craig and Halevi, Shai. *Implementing Gentry’s Fully-Homomorphic Encryption Scheme*. Cryptology ePrint Archive, Paper

2010/520. https://eprint.iacr.org/2010/520. 2010. URL: https://eprint.iacr.org/2010/520.

[GSW13] Gentry, Craig, Sahai, Amit, and Waters, Brent. *Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based.* Cryptology ePrint Archive, Paper 2013/340. https://eprint.iacr.org/2013/340. 2013. URL: https://eprint.iacr.org/2013/340.

[Gol08] Goldreich, Oded. *Computational Complexity: A Conceptual Perspective.* Cambridge University Press, 2008. DOI: 10.1017/CBO9780511804106.

[Gol01] Goldreich, Oded. *Foundations of Cryptography.* Vol. I. Cambridge University Press, 2001.

[Gol04] Goldreich, Oded. *Foundations of Cryptography.* Vol. II. Cambridge University Press, 2004.

[HPS98] Hoffstein, Jeffrey, Pipher, Jill, and Silverman, Joseph H. “NTRU: A ring-based public key cryptosystem”. In: *Algorithmic Number Theory.* Ed. by Joe P. Buhler. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 267–288. ISBN: 978-3-540-69113-6.

[Kal21] Kallenberg, Olav. *Probability Theory and Stochastic Modelling.* 3rd ed. Probability Theory and Stochastic Modelling. Cham, Switzerland: Springer Nature Switzerland AG, 2021. ISBN: 978-3-030-61870-4. DOI: 10.1007/978-3-030-61871-1.

[Kim+13] Kim, Jinsu, Lee, Moon Sung, Yun, Aaram, and Cheon, Jung Hee. *CRT-based Fully Homomorphic Encryption over the Integers.* Cryptology ePrint Archive, Paper 2013/057. https://eprint.iacr.org/2013/057. 2013. URL: https://eprint.iacr.org/2013/057.

[LM20] Li, Baiyu and Micciancio, Daniele. *On the Security of Homomorphic Encryption on Approximate Numbers.* Cryptology ePrint Archive, Paper 2020/1533. https://eprint.iacr.org/2020/1533. 2020. URL: https://eprint.iacr.org/2020/1533.

[LPR12] Lyubashevsky, Vadim, Peikert, Chris, and Regev, Oded. *On Ideal Lattices and Learning with Errors Over Rings*. Cryptology ePrint Archive, Paper 2012/230. `https://eprint.iacr.org/2012/230`. 2012. URL: `https://eprint.iacr.org/2012/230`.

[MP20] Micciancio, Daniele and Polyakov, Yuriy. *Bootstrapping in FHEW-like Cryptosystems*. Cryptology ePrint Archive, Paper 2020/086. `https://eprint.iacr.org/2020/086`. 2020. URL: `https://eprint.iacr.org/2020/086`.

[MR07] Micciancio, Daniele and Regev, Oded. "Worst-Case to Average-Case Reductions Based on Gaussian Measures". In: *SIAM Journal on Computing* 37.1 (2007), pp. 267–302. DOI: `10.1137/S0097539705447360`. eprint: `https://doi.org/10.1137/S0097539705447360`. URL: `https://doi.org/10.1137/S0097539705447360`.

[MF21] Mittelbach, Arno and Fischlin, Marc. *The Theory of Hash Functions and Random Oracles*. Vol. I. Springer Cham, 2021.

[Pei15] Peikert, Chris. *A Decade of Lattice Cryptography*. Cryptology ePrint Archive, Paper 2015/939. `https://eprint.iacr.org/2015/939`. 2015. URL: `https://eprint.iacr.org/2015/939`.

[Pei08] Peikert, Chris. *Public-Key Cryptosystems from the Worst-Case Shortest Vector Problem*. Cryptology ePrint Archive, Paper 2008/481. `https://eprint.iacr.org/2008/481`. 2008. URL: `https://eprint.iacr.org/2008/481`.

[Reg05] Regev, Oded. "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography". In: STOC '05. Baltimore, MD, USA: Association for Computing Machinery, 2005, pp. 84–93. ISBN: 1581139608. DOI: `10.1145/1060590.1060603`. URL: `https://doi.org/10.1145/1060590.1060603`.

[RAD78] Rivest, Ron, Adleman, Leonard, and Dertouzos, Michael. "On data banks and privacy homomorphisms". In: *Found. of Sec. Comp.* 1978, pp. 169–180.

[SS11] Scholl, P. and Smart, N. P. *Improved Key Generation For Gentry's Fully Homomorphic Encryption Scheme*. Cryptology ePrint Archive, Paper 2011/471. `https://eprint.iacr.org/2011/471`. 2011. URL: `https://eprint.iacr.org/2011/471`.

[SV09] Smart, N. P. and Vercauteren, F. *Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes*. Cryptology ePrint Archive, Paper 2009/571. `https://eprint.iacr.org/2009/571`. 2009. URL: `https://eprint.iacr.org/2009/571`.

[Vai11] Vaikuntanathan, Vinod. "Computing Blindfolded: New Developments in Fully Homomorphic Encryption". In: *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. University of Toronto. IEEE, 2011. DOI: `10.1109/FOCS.2011.98`.

[Ver11] Vershynin, Roman. *Introduction to the non-asymptotic analysis of random matrices*. 2011. arXiv: `1011.3027 [math.PR]`.

*

A. Appendix

Below is the code for Figure 1.

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import math

def sample_discrete_gaussian(sigma, size=1):
    """Sample from the discrete Gaussian distribution."""
    x = np.arange(-3*sigma, 3*sigma+1)
    pmf = np.exp(-x**2 / (2 * sigma**2))
    pmf /= pmf.sum()
    return np.random.choice(x, size=size, p=pmf)

# Set parameters
n_samples = 1000000
alpha = 100
sigma = alpha / math.sqrt(2 * math.pi)

# Generate discrete Gaussian samples in Z^2
samples = np.array([sample_discrete_gaussian(sigma, size=n_samples)
    for _ in range(2)]).T

# Compute 2D histogram
H, xedges, yedges = np.histogram2d(samples[:, 0], samples[:, 1], bins
    =(30, 30))

# Get x, y, z for non-zero counts
x, y = np.nonzero(H)
z = H[x, y]
x = xedges[x]
y = yedges[y]

# Create a new 3D subplot
fig = plt.figure(figsize=(8, 6))  # Define the figure size
ax = fig.add_subplot(111, projection='3d')
```

```python

# Plot a dot at the top of each line
scatter = ax.scatter(x, y, z, color='black', s=6, alpha=1)

# Set labels with larger font size
ax.set_zlabel('Frequency', fontsize=14)

# Set grid color to white and linestyle to dotted
ax.grid(color='white', linestyle=':', linewidth=0.75)

# Set the background color of the 3d plot to be transparent
ax.xaxis.pane.fill = False
ax.yaxis.pane.fill = False
ax.zaxis.pane.fill = False

# Make the grid lines transparent
ax.xaxis.pane.set_edgecolor('w')
ax.yaxis.pane.set_edgecolor('w')
ax.zaxis.pane.set_edgecolor('w')

ax.grid(True)
plt.tight_layout()
plt.show()
```