Notes on Gentry, Sahai and Water's Fully Homomorphic Encryption scheme

A V Sreejith ✉
IIT Goa

Abstract

Fully Homomorphic Encryption (FHE) is a form of encryption that allows computation on ciphertexts, generating an encrypted result that, when decrypted, matches the result of operations performed on the plaintext. This property enables secure data processing in untrusted environments, such as cloud computing, where sensitive information can be manipulated without exposing it to potential threats. FHE has significant implications for privacy-preserving computations, secure multi-party computations, and various applications in fields like healthcare, finance, and data analytics. The development of efficient FHE schemes remains a challenging area of research, with ongoing efforts to improve performance and reduce computational overhead.

2012 ACM Subject Classification Author: Please fill in 1 or more \ccsdesc macro

Keywords and phrases cryptography, homomorphic encryption

Digital Object Identifier 10.4230/LIPIcs...

Contents

1 Introduction

Imagine you are stationed in Goa, but you need to query sensitive information from servers in Delhi. Two key privacy concerns immediately arise:

- *Eavesdropping in transit* — You don't want anyone spying on your query or its results as they travel across the network.
- *Compromised servers* — Even if the server in Delhi is hacked, you still want your query and its results to remain secret.

The first problem is familiar and can be handled with standard public key encryption. You encrypt your query with Delhi's public key, they decrypt it with their private key, process it, and send the response back encrypted with your public key. Only you can decrypt it. This protects against eavesdropping during transmission. Encryption schemes like RSA and ElGamal work this way. However, some of these schemes are are vulnerable to quantum attacks. In the post-quantum era, we'll need new cryptographic tools. In this writeup, we'll explore one public key encryption scheme that is believed to be secure even against quantum adversaries. Additionally, it also solves the second issue.

The second problem is trickier. If the server itself is compromised, the attacker can see your query and its results. One option is to keep all the data encrypted on the server, but then every time you want something, the server would have to send the entire database to you. Clearly, that's inefficient and impractical.

So here's the crazy question researchers asked: *What if the server stores the data in encrypted format and could also process data without ever decrypting it?* Imagine sending an encrypted query to the server, which then performs the computation directly on encrypted data, and returns an encrypted answer. You decrypt the result locally, and abracadabra — you get the answer, while the server (or an attacker controlling it) never sees the actual data or your query.

This is exactly what *Fully Homomorphic Encryption* (FHE) makes possible. With FHE, you can run all kinds of computations on encrypted data — from simple yes/no queries like "Has my funding been approved?" to complex machine learning tasks like fingerprint matching. The possibilities are vast, and the implications for privacy and security are revolutionary. In particular, one can imagine a world where all data will be stored on servers (offcourse encrypted) and office/home processors will only be doing encryption and decryption while communicating continously with the server. Offcourse, there is a long way to go. In particular, making FHE fast and practical is the challenge.

The concept of homomorphic encryption was posed by Rivest, Adleman, and Dertouzos [11] in 1978. However, it wasn't until 2009 that Craig Gentry, a Ph.D. student then, presented the first plausible construction of a fully homomorphic encryption scheme [5]. Gentry, not only provided an FHE scheme but also gave a blueprint for constructing FHE schemes. Since then, numerous FHE schemes have been proposed, and there has been significant progress in making FHE simpler and efficient (see [8] for the history). However, an idea of Gentry, *bootstrapping*, remains the only idea to achieve FHE from somewhat homomorphic encryption schemes. Along with Zvika Brakerski and Vinod Vaikuntanathan [1, 2], Gentry won the Gödel prize in 2022 for their contributions to the field of FHE. In this presentation we will present a simple FHE scheme due to Gentry, Sahai and Waters [6]. For practical application, however, there are more efficient FHE schemes [3].

2 The Setting

At its core, *cryptography* is about hiding information so that only the intended person can read it. The basic trick is to take a readable message (called *plaintext*) and scramble it into an unreadable form (called *ciphertext*). Later, someone with the right "secret" can unscramble it back to the original message.

Different systems have been invented to do this. Famous ones include RSA [12], ElGamal [4], etc. Each uses clever mathematics to make sure that while it's easy to scramble (*encrypt*) and unscramble (*decrypt*) with the proper secret, it's practically impossible for an outsider to reverse the process without the secret.

2.1 Encryption schemes

There are two main styles of encryption.

- *Symmetric key encryption* — Here the same secret key is used for both encrypting and decrypting. It's fast and efficient, but there's a catch: both parties need to somehow share the key in secret first. This is like having one master key that both of you must carry. The other type of encryption is called *public key encryption*. We will describe it in detail.
- *Public key encryption* — This encryption system solves the key sharing problem of symmetric key encryption. There are two keys, a *public key* and a *private key*. As the name suggests, the public key is public and known to everyone. The private key is secret and known only to the owner of the key pair. A user can encrypt a message using the public key. Only the owner of the private key can decrypt the message. Think of it like putting a letter in a postbox. Only the postman who has the key can unlock and retrieve the letter.

Note that, in modern cryptography, we assume that everyone knows the encryption and decryption algorithms — the only thing kept secret is the private key. This transparency is actually what makes the systems strong, since their security depends on hard mathematical problems, not on keeping the method hidden.

We will now formally define fully homomorphic encryption.

2.2 The Fully Homomorphic Encryption scheme

A public key FHE scheme is a tuple $(KeyGen, Enc, Dec, Eval)$ of four algorithms. The first three algorithms are the same as in a public key encryption scheme. The fourth algorithm, $Eval$, allows for computations on ciphertexts. Let $\mathcal{P}$ be the plaintext space and $\mathcal{C}$ be the ciphertext space.

- Key Generation, $KeyGen(1^k) \to (pk, sk, ek)$: This algorithm takes as input a security parameter k - a measure of how secure the system should be. The larger the value of k, the more secure the system is. Based on this parameter, the key generation algorithm generates a public key (denoted by pk), a private key (denoted by sk), and an evaluation key (denoted by ek). The public key is used for encryption, the private key is used for decryption, and the evaluation key is used for performing computations on ciphertexts. The security parameter k is a measure of the security of the system.
- Encryption, $Enc(pk, \mu \in \mathcal{P}) \to \mathcal{C}$: This algorithm takes a plaintext and a public key as input and outputs a ciphertext.
- Decryption, $Dec(sk, ct \in \mathcal{C}) \to \mathcal{P}$: This algorithm takes a ciphertext and a private key as input and outputs a plaintext that satisfies the following correctness property:

$$Dec(sk, Enc(pk, \mu)) = \mu, \quad \forall \mu \in \mathcal{P}$$

- Evaluation, $Eval(ek, f, ct_1, ct_2, \ldots, ct_t) \to \mathcal{C}$: This algorithm takes as input an evaluation key, a function $f : \mathcal{P}^t \to \mathcal{P}$, and t ciphertexts $ct_1, ct_2, \ldots, ct_t$. It outputs a ciphertext ct_f such that the following correctness property holds:

$$Dec(sk, Eval(ek, f, ct_1, \ldots, ct_t)) = f(\mu_1, \ldots, \mu_t) \quad \text{where } \mu_i = Dec(sk, ct_i) \text{ for } i = 1, \ldots, t$$

  The evaluation algorithm allows for computations on ciphertexts without decrypting them. This is the key property of FHE.

  We now introduce the required mathematical tools.

3 Mathematical tools

3.1 Notations

Let q be a modulus (an integer) and $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$ denote the ring of integers modulo q under the operations addition and multiplication modulo q. Let $\mathbb{Z}_q^n$ denote the set of all column vectors of size n and with elements from $\mathbb{Z}_q$. For a vector $\mathbf{s} \in \mathbb{Z}_q^n$, we denote by $\mathbf{s}^\mathsf{T}$ the row vector that is the transpose of $\mathbf{s}$. Let $\mathbb{Z}_q^{l\times n}$ denote the set of all $l \times n$ matrices with elements from $\mathbb{Z}_q$.

Consider a matrix $\mathbb{A}$ and a vector $\mathbf{b}$. We denote by $(\mathbb{A} \mid \mathbf{b})$ the matrix obtained by appending $\mathbf{b}$ as the last column of $\mathbb{A}$. We denote by $(\mathbb{A}, \mathbf{b})$ the matrix obtained by appending $\mathbf{b}$ as a new row to $\mathbb{A}$.

$$(\mathbb{A} \mid \mathbf{b}) ::= \left(\begin{array}{c|c} \mathbb{A} & \mathbf{b} \end{array}\right) \quad \text{and} \quad (\mathbb{A}, \mathbf{b}) ::= \begin{pmatrix} \mathbb{A} \\ \mathbf{b} \end{pmatrix}$$

▶ **Example 1.** We have $\mathbf{u} = (1, 2)$, $\mathbf{v} = (u, 3)$, $\mathbf{w} = (-1, 0, 1)$, represent the column vectors

$$\mathbf{u} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \ \mathbf{v} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \text{ and } \mathbf{w} = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix},$$

$\mathbf{x} = (1 \mid 2)$ represents the row vector $\mathbf{x} = (1, 2)$,

$$\mathbb{A} = (u \mid 2u \mid 3u) = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \end{pmatrix} \text{ and } (\mathbb{A} \mid (5, 0)) = \begin{pmatrix} 1 & 2 & 3 & 5 \\ 2 & 4 & 6 & 0 \end{pmatrix}.$$

3.2 Learning with errors (LWE) assumption

Let $\mathbb{A} \in \mathbb{Z}_q^{l\times n}$ and $\mathbf{b} \in \mathbb{Z}_q^l$. Gaussian elimination can be used to solve the system of linear equation $\mathbb{A}\mathbf{s} = \mathbf{b} \pmod q$. In other words, in polynomial time we can find the solution $\mathbf{s}$. However, if we add some noise to the equations, the problem becomes very hard. Find $\mathbf{s}$ and $\mathbf{e}$ given $\mathbb{A}$ and $\mathbf{b}$ such that

$$\mathbf{b} = \mathbb{A}\mathbf{s} + \mathbf{e}, \qquad \text{where } \mathbf{e} \in [-\beta, \beta]^l \text{ is a small noise vector and } \beta \ll q.$$

It is believed that finding an "approximate" solution to a system of linear equations is difficult even for quantum computers. Furthermore, it is hard not just in the worst case, but in the average case too. This problem is called the Learning With Errors (LWE) problem introduced by Regev[9] for which he won the Gödel prize.

We say that a distribution is β-bounded if the probability of sampling an element outside the interval $[-\beta, \beta]$ is negligible. Let us now formally define the LWE assumption.

▶ **Definition 2** (Learning with errors (LWE) Assumption [9, 10])**.** *For any n, there are $q = exp(n)$, $l = (n+1)\log q$, $\beta \geq \omega(\log n)\sqrt{n}$ and a β-bounded probability distribution χ*

over $\mathbb{Z}_q$, such that for any probabilistic polynomial time distinguisher $\mathcal{D}$, the following two distributions are indistinguishable[1] *for $\mathcal{D}$:*

- *$\mathcal{D}$ is given samples of the form $(\mathbb{A} \mid \mathbb{A}\mathbf{s}+\mathbf{e})$ where $\mathbb{A} \xleftarrow{R} \mathbb{Z}_q^{l\times n}$, $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, and $\mathbf{e} \xleftarrow{R} \chi^l$.*
- *$\mathcal{D}$ is given samples of the form $(\mathbb{A} \mid \mathbf{u})$ where $\mathbb{A} \xleftarrow{R} \mathbb{Z}_q^{l\times n}$ and $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^l$.*

That is, the following two distributions are computationally indistinguishable (denoted by $\overset{c}{\approx}$):

$$\{(\mathbb{A} \mid \mathbb{A}\mathbf{s}+\mathbf{e}) \mid \mathbb{A} \xleftarrow{R} \mathbb{Z}_q^{l\times n}, \mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n, \mathbf{e} \xleftarrow{R} \chi^l\} \quad \overset{c}{\approx} \quad \{(\mathbb{A} \mid \mathbf{u}) \mid \mathbb{A} \xleftarrow{R} \mathbb{Z}_q^{l\times n}, \mathbf{u} \xleftarrow{R} \mathbb{Z}_q^l\}$$

The above assumption implies that a probabilistic polynomial time adversary (given $\mathbb{A} \in \mathbb{Z}_q^{l\times n}$ and $\mathbf{u} \in \mathbb{Z}_q^l$) cannot distinguish whether $\mathbf{u}$ is a random vector or it is of the form $\mathbb{A}\mathbf{s}+\mathbf{e}$ for some unknown $\mathbf{s}$ and small error vector $\mathbf{e} \in \chi^l$. In short, the LWE assumption is equivalent to the statement that it is hard to find $\mathbf{s}$.

We can briefly discuss the parameters in LWE assumption. This assumption is believed to be true also for q that is polynomial in n (with a slightly weaker reason). The restriction on l is very little. More or less any $l > n$ that is polynomial in n works. The probability distribution χ is assumed to be a Gaussian distribution centered at 0 and tapers off before reaching $\pm\beta$. For more details about the parameters see Regev [10].

3.3 Leftover hash lemma

We will require the following lemma to prove the security of our cryptosystem.

▶ **Lemma 3** (Leftover hash lemma [7]). *The following two distributions are statistically indistinguishable.*

$$\{(\mathbb{A}, \mathbb{R}\mathbb{A}) \mid \mathbb{A} \xleftarrow{R} \mathbb{Z}_q^{l\times n}, \mathbb{R} \xleftarrow{R} \{0,1\}^{r\times l}\} \quad \overset{s}{\approx} \quad \{(\mathbb{A}, \mathbb{B}) \mid \mathbb{A} \xleftarrow{R} \mathbb{Z}_q^{l\times n}, \mathbb{B} \xleftarrow{R} \mathbb{Z}_q^{r\times n}\}$$

Given a matrix $\mathbb{A}$ and a row vector $\mathbf{v}$, the leftover hash lemma says that it is hard to identify if $\mathbf{v}$ is a random vector or if $\mathbf{v}$ is got by adding some rows of $\mathbb{A}$. More generally, if we take a matrix $\mathbb{A}$ and multiply it with a random binary matrix $\mathbb{R}$ on the left (thereby taking random $\{0,1\}$ linear combinations of the rows of $\mathbb{A}$), the resulting matrix $\mathbb{R}\mathbb{A}$ is statistically indistinguishable from a truly random matrix $\mathbb{B}$. Statisticaly Indistinguishability is a stronger notion than computational indistinguishability. It means that no adversary, even with unbounded computational power, can distinguish between the two distributions with a non-neglible advantage.

4 A Fully Homomorphic Encryption Scheme

We now give the Gentry-Sahai-Waters (GSW) fully homomorphic encryption scheme. The scheme is based on the LWE assumption. We first give the *KeyGen*, *Enc* and *Dec* algorithms. We then show how to perform homomorphic operations on ciphertexts. Finally, we prove the security of the scheme.

4.1 GSW encryption scheme

The GSW encryption scheme is a public key encryption scheme. The message space is $\{0,1\}$ and the ciphertext space is $\mathbb{Z}_q^{l\times(n+1)}$ for parameters q, n and $l = (n+1)\log q$. The scheme is as follows:

[1] Formally, indistinguishable means the probability that $\mathcal{D}$ correctly identifies the sample from the first Item is at most $1/2+\epsilon$ for an ϵ less than any 1/polynomial.

- $KeyGen(1^k) \to (pk, sk)$:
  - Let $n = k$, q be a modulus, $l = (n+1)\log q$, and $\chi = [-\beta, \beta] \subseteq \mathbb{Z}_q$ be the error distribution for a $\beta \ll q/4$ as given by the LWE assumption.
  - Generate:

    $$\mathbf{e} \xleftarrow{R} \chi^l, \qquad \mathbf{s}' \xleftarrow{R} \mathbb{Z}_q^n, \qquad \text{and} \qquad \mathbb{B} \xleftarrow{R} \mathbb{Z}_q^{l\times n}.$$

    and let $\mathbb{A} = (\mathbb{B} \mid \mathbb{B}\mathbf{s}' + \mathbf{e}) \in \mathbb{Z}_q^{l\times(n+1)}$ and $\mathbf{s} = (-\mathbf{s}', 1) \in \mathbb{Z}_q^{(n+1)}$.

    $$\mathbb{A} ::= \left(\begin{array}{c|c} \mathbb{B} & \mathbb{B}\mathbf{s}' + \mathbf{e} \end{array}\right), \qquad \text{and} \qquad \mathbf{s} ::= \begin{pmatrix} -\mathbf{s}' \\ 1 \end{pmatrix}$$

  - Public key, $pk = \mathbb{A}$.
  - Secret key, $sk = \mathbf{s}$.

  Intuition: One should think of each row of $\mathbb{A}$ as an encoding of the bit 0 (with a small noise $\mathbf{e}$). The bit 1 will be a "shift" of this encoding. Note that, from the LWE assumption, the distribution of $\mathbb{A}$ is indistinguishable from a truly random matrix and hence an adversary cannot identify the secret key $\mathbf{s}$.

- $\text{Enc}(pk = \mathbb{A},\ \mu \in \{0,1\}) \to \mathbb{Z}_q^{l\times(n+1)}$:
  - Generate: $\mathbb{R} \xleftarrow{R} \{0,1\}^{l\times l}$.
  - Ciphertext,

    $$\mathbb{C} = \mathbb{R}\mathbb{A} + \mu\mathbb{G}$$

    where $\mathbb{G} \in \mathbb{Z}_q^{l\times(n+1)}$ is a fixed error correcting code matrix.

  Intuition: As we noted earlier, each row of $\mathbb{A}$ is an encoding of the bit 0. A row of $\mathbb{R}$ takes a random $\{0,1\}$ linear combination of rows of $\mathbb{A}$. This linear combination is another encoding of the bit 0. Thus $\mathbb{R}\mathbb{A}$ consists of l rows where each row is an encodings of the bit 0. Due to the leftover hash lemma, one cannot distinguish $\mathbb{R}\mathbb{A}$ from a truly random matrix. This gives us an encoding of bit 0 that "looks" random. To encode bit 1 we add $\mathbb{G}$ to $\mathbb{R}\mathbb{A}$. Since $\mathbb{R}\mathbb{A}$ "looks" random so does $\mathbb{C}$. Thus, an adversary cannot identify if $\mathbb{C}$ encodes bit 0 or bit 1. How do we get back the bit if we know the secret key? We see that

  $$\mathbb{R}\mathbb{A}\mathbf{s} = \left(\begin{array}{c|c} \mathbb{B} & \mathbb{B}\mathbf{s}' + \mathbf{e} \end{array}\right)\begin{pmatrix} -\mathbf{s}' \\ 1 \end{pmatrix} = -\mathbb{B}\mathbf{s}' + \mathbb{B}\mathbf{s}' + \mathbf{e} = \mathbf{e}$$

  and therefore

  $$\mathbb{C}\mathbf{s} = \mu\mathbb{G}\mathbf{s} + \mathbb{R}\mathbb{A}\mathbf{s} = \mu\mathbb{G}\mathbf{s} + \mathbf{e} \tag{1}$$

  For $\mathbf{t} = \mathbb{G}\mathbf{s}$, we have

  $$\mathbb{C}\mathbf{s} = \mu\mathbb{G}\mathbf{s} + \mathbf{e} = \mu\mathbf{t} + \mathbf{e} = \begin{cases} \mathbf{e} & \text{if } \mu = 0, \\ \mathbf{t} + \mathbf{e} & \text{if } \mu = 1. \end{cases}$$

Since $\mathbf{e} \in [-\beta, \beta]^l$, for a small $\beta < q/4$, we can distinguish between the two cases for μ. Since $\mathbf{s}$ is picked uniformly at random from $\mathbb{Z}_q$, for suitable matrices $\mathbb{G}$, with high probability at least one entry of $\mathbf{t}$ is not in $[-\beta, \beta]$. This leads to the decryption algorithm.

- Dec($sk = \mathbf{s}$, $\mathbb{C}$) $\to \{0, 1\}$:
  - Generate: $\mathbf{t} = \mathbb{C}\mathbf{s}$.
  - If $\mathbf{t} \in [-q/4, q/4]^l$, return bit 0.
  - Otherwise, return bit 1.

4.2 Security of GSW

We first argue why GSW is a public key encryption. We show that no probabilistic polynomial time algorithm can distinguish between an encryption of bit 0 and an encryption of bit 1 with a non-neglible advantage. In other words, we need to show that

$$Enc(pk, 0) \overset{c}{\approx} Enc(pk, 1).$$

This will ensure that GSW is CPA-secure - a security measure that is widely used. It suffices to show for $\mu \in \{0, 1\}$

$$Enc(pk, \mu) \overset{c}{\approx} \{\mathbb{C} \mid \mathbb{C} \overset{R}{\leftarrow} \mathbb{Z}_q{}^{l\times(n+1)}\}$$

since it will follow that

$$Enc(pk, 0) \overset{c}{\approx} \{\mathbb{C} \mid \mathbb{C} \overset{R}{\leftarrow} \mathbb{Z}_q{}^{l\times(n+1)}\} \overset{c}{\approx} Enc(pk, 1).$$

Let $\mu \in \{0, 1\}$. From the LWE assumption, we have

$$(\mathbb{B} \mid \mathbf{u}) \quad \overset{c}{\approx} \quad (\mathbb{B} \mid \mathbb{B}\mathbf{s}' + \mathbf{e}) = \mathbb{A}$$

where $\mathbb{B} \overset{R}{\leftarrow} \mathbb{Z}_q^{l\times n}$, $\mathbf{s}' \overset{R}{\leftarrow} \mathbb{Z}_q^n$, $\mathbf{e} \overset{R}{\leftarrow} \chi^l$, and $\mathbf{u} \overset{R}{\leftarrow} \mathbb{Z}_q^l$. That is, $\mathbb{A} = (\mathbb{B} \mid \mathbb{B}\mathbf{s}' + \mathbf{e})$ is computationally indistinguishable from $(\mathbb{B} \mid \mathbf{u})$ where $\mathbf{u}$ is a random vector. Therefore $\mathbb{R}\mathbb{A}$ is computationally indistinguishable from $\mathbb{R}(\mathbb{B} \mid \mathbf{u})$ for any random matrix $\mathbb{R}$. From leftover hash lemma, we have

$$\mathbb{R}(\mathbb{B} \mid \mathbf{u}) \quad \overset{s}{\approx} \quad \mathbb{C}$$

where $\mathbb{C} \overset{R}{\leftarrow} \mathbb{Z}_q^{l\times(n+1)}$ and $\mathbb{R} \overset{R}{\leftarrow} \{0, 1\}^{l\times l}$. That is, $\mathbb{R}(\mathbb{B} \mid \mathbf{u})$ is statistically indistinguishable from $\mathbb{C}$ for any random matrix $\mathbb{C}$. In other words, $\mathbb{R}(\mathbb{B} \mid \mathbb{B}\mathbf{s}' + \mathbf{e})$ is computationally indistinguishable from a random matrix $\mathbb{C}$. Therefore, a probabilistic polynomial time algorithm "sees" $\mathbb{R}\mathbb{A}$ as a random matrix. It looks random even if we add $\mathbb{G}$ (no matter which $\mathbb{G}$) to it. To summarize, our argument.

$$\mathbb{R}(\mathbb{B} \mid \mathbb{B}\mathbf{s}' + \mathbf{e}) + \mu\mathbb{G} \quad \overset{c}{\approx} \quad \mathbb{R}(\mathbb{B} \mid \mathbf{u}) + \mu\mathbb{G} \quad \overset{s}{\approx} \quad \mathbb{C}$$

This concludes our argument that GSW is CPA secure.

Before we show why GSW is a fully homomorphic encryption scheme, let us fix $\mathbb{G}$.

4.3 Finding the appropriate $\mathbb{G}$

To show that GSW is an FHE it suffices to show that it can work over any boolean circuits. Without loss of generality, we can assume that a boolean circuit consists only of negation and AND gates.

Let $\mathbb{C}$ be a ciphertext encrypting bit μ. The beauty of GSW encoding is that, the secret key $\mathbf{s}$ is an *approximate eigen vector* of a submatrix $\mathbb{C}'$ of $\mathbb{C}$. The corresponding eigen value is the hidden message μ. That is,

$$\mathbb{C}'\mathbf{s} = \mu\mathbf{s} + \mathbf{e} \quad \text{for a small error } \mathbf{e} \in [-\beta, \beta]^l$$

Note that, exact eigen values can be identified through various numerical methods. However, the approximate eigen vector problem turns out to be quite difficult to solve - just like solving approximate linear equations (the LWE problem). GSW's observation is that, this property holds over addition and multiplication of ciphertexts. To see this, consider two ciphertexts $\mathbb{C}_1$ and $\mathbb{C}_2$ encrypting bits μ_1 and μ_2 respectively. Let $\mathbb{C}_1'$ and $\mathbb{C}_2'$ be the corresponding matrices such that $\mathbb{C}_i'\mathbf{s} = \mu_i\mathbf{s} + \mathbf{e}_i$ for $i = 1, 2$. Then,

$$(\mathbb{C}_1' + \mathbb{C}_2')\mathbf{s} = \mathbb{C}_1'\mathbf{s} + \mathbb{C}_2'\mathbf{s} = (\mu_1 + \mu_2)\mathbf{s} + (\mathbf{e}_1 + \mathbf{e}_2)$$
$$(\mathbb{C}_1'\mathbb{C}_2')\mathbf{s} = \mathbb{C}_1'(\mathbb{C}_2'\mathbf{s}) = \mathbb{C}_1'(\mu_2\mathbf{s} + \mathbf{e}_2) = \mu_2(\mu_1\mathbf{s} + \mathbf{e}_1) + \mathbb{C}_1'\mathbf{e}_2 = (\mu_1\mu_2)\mathbf{s} + (\mu_2\mathbf{e}_1 + \mathbb{C}_1'\mathbf{e}_2)$$

In both the above equations, the error has increased. However, in the case of addition, the error has increased only by a small amount. In the case of multiplication, the error has increased by a large amount and hence we may not be able to distinguish between the two cases $\mu = 0$ and $\mu = 1$ during decryption. This is because of the term $\mathbb{C}_1'\mathbf{e}_2$. If we can ensure that this term is small, we can perform homomorphic operations on ciphertexts. The neat trick of GSW is to do the same thing but use a different matrix than $\mathbb{C}_1'$ during multiplication and ensure that the error remains bounded. Now, for all this to happen, we need to fix the matrix $\mathbb{G}$.

$$\mathbb{G} ::= \begin{pmatrix} 1 & & & \\ 2^1 & & & \\ \dots & & & \\ 2^{\log q - 1} & & & \\ & 1 & & \\ & \dots & & \\ & 2^{\log q - 1} & & \\ & & \dots & \\ & & & 1 \\ & & & \dots \\ & & & 2^{\log q - 1} \end{pmatrix} \in \mathbb{Z}_q^{l \times (n+1)} \qquad \text{where } l = (n+1)\log q$$

Let us try to understand $\mathbb{G}$. Any $a \in \mathbb{Z}_q$ can be decomposed into its binary representation. That is, there are bits $b_0, b_1, \dots, b_{\log q - 1} \in \{0, 1\}$ such that

$$a = b_0 1 + b_1 2^1 + \dots + b_{\log q - 1} 2^{\log q - 1} = (b_0, b_1, \dots, b_{\log q - 1})^{\mathsf{T}}(1, 2^1, \dots, 2^{\log q - 1})$$

For any $a \in \mathbb{Z}_q$ let $\mathit{BitDecomp}(a) = (b_0, b_1, \dots, b_{\log q - 1})^{\mathsf{T}} \in \{0, 1\}^{\log q}$ denote the binary decomposition of a. Recall that $(1, 2, \dots, 2^{\log q - 1})$ is a column vector of length $\log q$ according to our notation. Hence $\mathit{BitDecomp}(a)^{\mathsf{T}}(1, 2, \dots, 2^{\log q - 1}) = a$.

We can extend this to vectors. Consider the vector $\mathbf{a} = (a_1, a_2, \dots, a_{n+1}) \in \mathbb{Z}_q{}^{(n+1)}$ and let the binary decomposition of a_i be the vectors $\mathbf{b}^i$. Then $\mathit{BitDecomp}(\mathbf{a}) = (\mathbf{b}^1, \dots, \mathbf{b}^{n+1})$.

Then, we have that $\mathit{BitDecomp}(\mathbf{a})^{\mathsf{T}}\mathbb{G} = \mathbf{a}$.

$$\begin{pmatrix}\mathbf{b}^1 = (b^1_0, \ldots, b^1_{\log q-1})^{\mathsf{T}} & \mathbf{b}^2 & \ldots & \mathbf{b}^{n+1}\end{pmatrix}\begin{pmatrix} \begin{matrix}1\\2^1\\\cdots\\2^{\log q-1}\end{matrix} & & & \\ & \begin{matrix}1\\\cdots\\2^{\log q-1}\end{matrix} & & \\ & & \ldots & \\ & & & \begin{matrix}1\\\cdots\\2^{\log q-1}\end{matrix}\end{pmatrix} = \begin{pmatrix}a_1\\a_2\\\ldots\\a_{n+1}\end{pmatrix}$$

We can further extend *BitDecomp* to act on matrices. For a matrix $\mathbb{D} \in \mathbb{Z}_q^{l\times(n+1)}$, let $\mathit{BitDecomp}(\mathbb{D})$ be the matrix obtained by replacing each row of $\mathbb{D}$ by its binary decomposition. Then, we have

$$\mathit{BitDecomp}(\mathbb{D})^{\mathsf{T}}\mathbb{G} = \mathbb{D} \quad \text{for any matrix } \mathbb{D} \in \mathbb{Z}_q^{l\times(n+1)}. \tag{2}$$

The above equation will play a crucial role for us. Another important property of $\mathit{BitDecomp}(\mathbb{D})$ is that it is a binary matrix. That is,

$$\mathit{BitDecomp}(\mathbb{D}) \in \{0,1\}^{l\times l} \quad \text{for all } \mathbb{D} \in \mathbb{Z}_q^{l\times(n+1)}. \tag{3}$$

We will see that this ensures the errors are not blown up during homomorphic operations.

Earlier we said that there is a submatrix $\mathbb{C}'$ of the ciphertext $\mathbb{C}$ such that the secret key $\mathbf{s}$ is an approximate eigen vector of $\mathbb{C}'$. Look at the $(n+1)$ rows of $\mathbb{G}$ that contains only 1 and the rest all 0s. That is to rows numbered $1, \log q, 2\log q, \ldots, (n+1)\log q$. Let $\mathbb{C}'$ be the restriction of $\mathbb{C}$ to these rows. Then

$$\mathbb{C}'\mathbf{s} = \mu\mathbb{G}'\mathbf{s} + \mathbf{e}' = \mu\mathbf{s} + \mathbf{e}'$$

where $\mathbb{G}'$ and $\mathbf{e}'$ are the corresponding restrictions of $\mathbb{G}$ and $\mathbf{e}$ to those rows. Note that $\mathbb{G}'\mathbf{s} = \mathbf{s}$, since $\mathbb{G}'$ is the identity matrix. Thus $\mathbf{s}$ is an approximate eigen vector of $\mathbb{C}'$ with eigen value μ. The beauty of GSW is that it can ensure approximate eigen vector property is preserved under homomorphic operations without blowing up the error.

4.4 Homomorphic operations

Any boolean circuit can be constructed using only Negation and AND gates. Hence, it suffices to show how to perform homomorphic encryption over circuits with Negation and AND gates. Usually, for evaluating functions in FHE, we use an evaluation key. However, GSW does not require an evaluation key for "shallow" depth circuits. We first see how to evaluate such circuits and then using Gentry's bootstrapping idea, we lift this evaluation to arbitrary depth circuits.

Let us now give an inductive evaluation of shallow depth circuits.

- $\mathit{Eval}(\text{Negation}, \mathbb{C})$: Let $\mathbb{C}$ be a ciphertext encrypting bit μ. We want to compute a ciphertext $\mathbb{C}_{neg}$ encrypting bit $1-\mu$.

  $$\mathbb{C}_{neg} = \mathbb{G} - \mathbb{C}$$

To see $\mathbb{C}_{neg}$ encrypts $1-\mu$, note that

$$\mathbb{C}_{neg}\mathbf{s} = (\mathbb{G}-\mathbb{C})\mathbf{s} = \mathbb{G}\mathbf{s} - (\mu\mathbb{G}\mathbf{s}+\mathbf{e}) = (1-\mu)\mathbb{G}\mathbf{s} - \mathbf{e}$$

The above equation preserves the error bounds since $-\mathbf{e} \in [-\beta,\beta]^l$. Thus $\mathbb{C}_{neg}$ is a valid ciphertext encrypting bit $1-\mu$.

- $Eval(\text{AND}, \mathbb{C}_1, \mathbb{C}_2)$: Let $\mathbb{C}_1$ and $\mathbb{C}_2$ be ciphertexts encrypting bits μ_1 and μ_2 respectively. We want to compute a ciphertext $\mathbb{C}_{and}$ encrypting bit $\mu_1 \wedge \mu_2$.

$$\mathbb{C}_{and} = BitDecomp(\mathbb{C}_1)\mathbb{C}_2$$

To see $\mathbb{C}_{and}$ encrypts $\mu_1 \wedge \mu_2$, note that

$$\begin{aligned}\mathbb{C}_{and}\mathbf{s} &= BitDecomp(\mathbb{C}_1)\mathbb{C}_2\mathbf{s} = BitDecomp(\mathbb{C}_1)(\mu_2\mathbb{G}\mathbf{s}+\mathbf{e}_2)\\ &= \mu_2 BitDecomp(\mathbb{C}_1)\mathbb{G}\mathbf{s} + BitDecomp(\mathbb{C}_1)\mathbf{e}_2\\ &= \mu_2\mathbb{C}_1\mathbf{s} + BitDecomp(\mathbb{C}_1)\mathbf{e}_2 = \mu_2(\mu_1\mathbb{G}\mathbf{s}+\mathbf{e}_1) + BitDecomp(\mathbb{C}_1)\mathbf{e}_2\\ &= (\mu_1 \wedge \mu_2)\mathbb{G}\mathbf{s} + (\mu_2\mathbf{e}_1 + BitDecomp(\mathbb{C}_1)\mathbf{e}_2)\end{aligned}$$

The above equation preserves the error bounds $\mu_2\mathbf{e}_1 + BitDecomp(\mathbb{C}_1)\mathbf{e}_2 \in [-l\beta-\beta, l\beta+\beta]^l$ since $BitDecomp(\mathbb{C}_1)$ is a binary matrix of width l. Thus $\mathbb{C}_{and}$ is a valid ciphertext encrypting bit $\mu_1 \wedge \mu_2$ as long as $(l+1)\beta \le q/4$.

- $Eval(f, \mathbb{C}_1, \mathbb{C}_2, \ldots, \mathbb{C}_k)$ where f is a depth d circuit: The worst case scenario is when there is a series of d AND gates. We can still retrieve the message if $(l+1)^d\beta^d \le q/4$. Taking $q = 2^{n^{\omega(1)}}$ and substituting for $l = (n+1)\log q$ and $\beta = \omega(\log n)\sqrt{n}$, we have that $(l+1)\beta \le n^{1+\epsilon}$ for some ϵ. Hence, we require $n^{d(1+\epsilon)} \le q/4$ or

$$d(1+\epsilon) \le \log q - 2 \quad \text{or} \quad d \le \frac{n^{\omega(1)}-2}{1+\epsilon} \quad \text{or} \quad d \le n^c \text{ for some } c < 1$$

Hence, the message can be retrieved for "almost" linear circuits f.

$$Dec(sk, Eval(f, \mathbb{C}_1, \ldots, \mathbb{C}_k)) = f(\mu_1, \ldots, \mu_k) \quad \text{for all } \mu_i \in \{0,1\} \text{ and } \mathbb{C}_i = Enc(pk, \mu_i) \tag{4}$$

To summarize the evalution, we note that AND gates increases error. Therefore, we can only homomorphically evaluate circuits of depth that are almost linear. Our next aim is to homomorphically evaluate circuits of arbitrary depth circuits. Gentry introduces an idea called *bootstrapping* to make this possible. Even after more than a decade since this idea was introduced, no one has come up with an alternate idea. As of now, there is all FHE schemes use bootstrapping. That is how crazy boostrapping is.

4.5 Bootstrapping

We have seen how to evaluate shallow depth circuits. To reach arbitrary depth, we need a way to control the growth of noise in ciphertexts. This is where bootstrapping comes in.

The core idea is simple: bootstrapping "refreshes" a ciphertext. Suppose we have a ciphertext $\mathbb{C}$ that encrypts a bit μ, but the error inside $\mathbb{C}$ has grown large. Bootstrapping takes $\mathbb{C}$ as input and produces a new ciphertext $\mathbb{C}'$ that still encrypts μ, but with a small, bounded error. This keeps the encryption usable for further computation.

Refreshing ciphertext: Consider the decryption algorithm

$$Dec : \mathcal{S} \times \mathcal{C} \to \{0,1\}$$

which takes a secret key $sk \in \mathcal{S}$ and a ciphertext $\mathbb{C} \in \mathcal{C}$, and outputs the plaintext bit μ.

We can view this decryption process as a Boolean circuit f such that

$$f(BitDecomp(sk), BitDecomp(\mathbb{C})) = Dec(sk, \mathbb{C}) = \mu.$$

The important thing to note is that, f has no noise in its output: it always produces the exact bit μ. Any noise in the ciphertext is absorbed by the logic of the decryption circuit. Now fix a ciphertext $\mathbb{C} = Enc(pk, \mu)$. Our goal is to obtain a fresh ciphertext $\mathbb{C}'$ of μ with small error.

If we hardcode $BitDecomp(\mathbb{C})$ into the inputs of f, we get a new circuit $f_{\mathbb{C}}$ that depends only on the secret key bits:

$$f_{\mathbb{C}}(s_1, s_2, \ldots, s_l) = \mu.$$

where $BitDecomp(sk) = (s_1, s_2, \ldots, s_l)$.

To get a fresh encryption of μ, we can homomorphically evaluate $f_{\mathbb{C}}$ on encryptions of the secret key bits. In the GSW scheme, an encryption of each bit s_i of the secret key is available as the evaluation key. That is,

$$ek = (\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_l) \quad \text{where } \mathbb{S}_i = Enc(pk, s_i) \text{ for } i = 1, \ldots, l.$$

We can now homomorphically evaluate the circuit $f_{\mathbb{C}}$ on these encrypted bits:

$$\mathbb{C}' = Eval(f_{\mathbb{C}}, \mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_l).$$

From Equation 4, we have that

$$Dec(sk, \mathbb{C}') = f_{\mathbb{C}}(s_1, \ldots, s_l) = \mu$$

as long as $f_{\mathbb{C}}$ has shallow enough depth to be evaluated homomorphically.

Why this works: The depth of $f_{\mathbb{C}}$ is $O(\log n)$, since it essentially computes matrix–vector products via inner products of l bit integers, each of which requires $O(\log l) = O(\log n)$ depth. Thus, $f_{\mathbb{C}}$ can be homomorphically evaluated.

The resulting ciphertext $\mathbb{C}'$ encrypts the same bit μ as $\mathbb{C}$, but its error depends only on the depth of $f_{\mathbb{C}}$ and the error in the encrypted secret key bits $\mathbb{S}_i$ — not on the error in $\mathbb{C}$. This means that even if $\mathbb{C}$ was "noisy but still decryptable," the bootstrapping process produces a clean ciphertext $\mathbb{C}'$ that can be used for further computations.

Evaluating circuits of arbitrary depth: We can now evaluate circuits of arbitrary depth by refreshing ciphertexts as needed. After every AND gate, we run the bootstrapping procedure. As long as the scheme can homomorphically evaluate the decryption circuit (plus an extra depth for the AND gate), we can build fully homomorphic encryption.

Circular security: A subtlety in this construction is that the evaluation key ek contains encryptions of the secret key bits. To ensure security, we need to assume that the scheme is "circular secure", meaning that it remains CPA secure even when encryptions of the secret key are part of the public information.

This concludes our construction of the fully homomorphic encryption by Gentry, Sahai and Waters.

5 Summary

We now summarize the FHE of Gentry, Sahai and Waters.

- Key Generation, $KeyGen(1^k) \to (pk, ek, sk)$:
  - Choose parameters (n, q, l, χ) based on the security parameter 1^k.
  - Fix the matrix

$$\mathbb{G} ::= \begin{pmatrix} 1 & & & \\ \cdots & & & \\ 2^{\log q - 1} & & & \\ & 1 & & \\ & \cdots & & \\ & 2^{\log q-1} & & \\ & & \cdots & \\ & & & 1 \\ & & & \cdots \\ & & & 2^{\log q - 1} \end{pmatrix} \in \mathbb{Z}_q^{l \times (n+1)}$$

  - Generate:

$$\mathbf{e} \xleftarrow{R} \chi^l, \qquad \mathbf{s}' \xleftarrow{R} \mathbb{Z}_q^n, \qquad \text{and} \qquad \mathbb{B} \xleftarrow{R} \mathbb{Z}_q^{l \times n}.$$

  - Public key, $pk ::= (\mathbb{B} \mid \mathbb{B}\mathbf{s}' + \mathbf{e})$.
  - Secret key, $sk ::= (-\mathbf{s}', 1)$.
  - Evaluation key, $ek ::= (Enc(pk, s_1), \dots, Enc(pk, s_l))$ where s_i is i^{th} bit of sk.

- Encryption, $Enc(\mathbb{A} = pk,\ \mu \in \{0,1\})$:

$$\mathbb{C} = \mathbb{R}\mathbb{A} + \mu\mathbb{G} \quad \text{where } \mathbb{R} \xleftarrow{R} \{0,1\}^{l \times l}$$

- Decryption $Dec(\mathbf{s} = sk,\ \mathbb{C})$:

$$\mu' = \begin{cases} 0 & \text{if } \mathbb{C}\mathbf{s} \in [-q/4, q/4]^l \\ 1 & \text{otherwise.} \end{cases}$$

- Evaluation, $Eval(f, \mathbb{C}_1, \dots, \mathbb{C}_k) \to$ ciphertext, where f is a n^ϵ-depth boolean circuit:
  - $Eval(\text{Negation}, \mathbb{C}) ::= \mathbb{G} - \mathbb{C}$.
  - $Eval(\text{AND}, \mathbb{C}_1, \mathbb{C}_2) ::= BitDecomp(\mathbb{C}_1)\mathbb{C}_2$.

- Bootstrapping, $Boot(\mathbb{C}, ek) \to$ ciphertext:
  - Let f be the boolean circuit $f(BitDecomp(sk), BitDecomp(\mathbb{C})) = Dec(sk, \mathbb{C})$.
  - Hardcode $\mathbb{C}$ and get the $O(\log n)$-depth circuit $f_{\mathbb{C}}(BitDecomp(sk)) = Dec(sk, \mathbb{C})$.
  - $\mathbb{C}' = Eval(f_{\mathbb{C}}, ek)$.

- Evaluation, arbitrary depth circuit: Apply bootstrapping after every AND gate.

- Security: GSW is CPA-secure under the LWE and circular secure assumption.

$$\mathbb{R}(\mathbb{B} \mid \mathbb{B}\mathbf{s}' + \mathbf{e}) + \mu\mathbb{G} \quad \overset{c}{\approx} \quad \mathbb{R}(\mathbb{B} \mid \mathbf{u}) + \mu\mathbb{G} \quad \overset{s}{\approx} \quad \text{random matrix } \mathbb{C}$$

References

1 Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325. ACM, 2012.

2 Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM Journal on Computing*, 43(2):831–871, 2014.

3 Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017.

4 T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

5 Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178. ACM, 2009.

6 Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2013.

7 Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 12–24. ACM, 1989.

8 Chiara Marcolla, Victor Sucasas, Marc Manzano, Riccardo Bassoli, Frank H.P. Fitzek, and Najwa Aaraj. Survey on fully homomorphic encryption, theory, and applications. Cryptology ePrint Archive, Paper 2022/1602, 2022.

9 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.

10 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *CoRR*, abs/2401.03703, 2024.

11 R. L. Rivest, L. Adleman, and M.L Deaouzos. On data banks and privacy homomorphism. *Foundations of Secure Computation*, pages 169–179, 1978.

12 R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.