ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
TECHNICAL UNIVERSITY OF CRETE

Integrated Master in
Electrical and Computer Engineering

---

# Implementation of Homomorphic Encryption Techniques

---

by

Ioannis-Leonidas Steiakakis

## Supervision and Committee

Professor George Karystinos (Supervisor)

Professor Athanasios Liavas

Professor Soterios Ioannidis

**Abstract**

Cryptography is the primary tool used to secure our data in public environments, where possible malicious adversaries may desire to intercept or tamper it. Encryption is the cryptographic technique used against eavesdropping adversaries. The newly arising concept of homomorphic encryption encompasses techniques for encrypting data in a way such that a desired algebraic structure is preserved, hence allowing to perform "blind computations", i.e., to perform operations directly on encrypted data, without the need for decryption first; the operand(s) and the result remain encrypted throughout the process. This diploma thesis is focused on lattice-based cryptosystems (i.e., the underlying mathematical problems ensuring security are problems on lattices), with lattice-based cryptography forming the new NIST post-quantum cryptography standard. We explore (a part of) the deep theory of lattice-based cryptography and study thoroughly the potentials and constraints of homomorphic encryption, which enables to design and implement efficiently a small suite of algorithms for encrypted computations. The programming library (written in C++) that we used for the implementation part is the Microsoft Simple Encrypted Arithmetic Library (SEAL), which provides some very basic encrypted math operations (like addition, subtraction, and multiplication). Using SEAL, some of the encrypted operations that we implemented are encrypted number inversion, encrypted square root, encrypted absolute value, encrypted linear algebra operations, encrypted fast Fourier transform (FFT), and an encrypted version of the if-else statement.

# Acknowledgments

I would like to express my deep gratitude to my diploma thesis supervisor, Professor George Karystinos, for suggesting this interesting topic of homomorphic encryption. Although this field was new to both of us, his guidance and ideas helped in forming the direction of my research. I am also grateful for his supportive and non-pressuring attitude, which at that time gave me the potential to investigate this topic on a deeper level at my own pace. This experience has greatly enriched my academic journey.

# Contents

# List of Figures

# 1 Introduction

Cryptography is the science of securing data in the presence of adversaries. More specifically, at its core, cryptography addresses the "confidentiality" and "integrity" principles from the CIA triad:

- **Confidentiality**: Protect data against unauthorized read access.
- **Integrity**: Protect data against unauthorized tampering.
- **Availability**: Ensure data is accessible on demand by authorized users.

Confidentiality is achieved through encryption.
Integrity is achieved through cryptographic hashing and digital signatures.

The primary purpose of encryption is to make data impossible to be interpreted by unauthorized parties. This is achieved via the encryption process that transforms the initial data that need protection into data that look like "garbage", namely, impossible to extract any information from it when accessing it. The encryption process should be reversible, of course, else, it is just a useless process that converts data to "garbage". However, the encryption process reversibility should be easy only for the authorized parties. For everyone else, it should be extremely hard to reverse encryption. So, in this encrypted form, data can safely travel across public channels or be stored in third party hosts.

This inaccessible form of data, though, could be an obstacle if one needs to process it. Both (mathematically guaranteed instead of policy-based) data privacy and data processing, one cannot have it all, right? Well, ...

The emerging, rapidly evolving, and fascinating field of Homomorphic Encryption (HE), which remains at the forefront of active research today, is here to say otherwise.

Homomorphic encryption enables data to be processed while encrypted. Even if one cannot "see" the data, one is able to process them, "blindly", but, of course, correctly.

The term "homomorphic" in "homomorphic encryption" comes from a mathematical concept known as "homomorphism". Generally speaking, a homomorphism is a structure-preserving map between two algebraic structures, such as groups, rings, or vector spaces.

# 2 Background

## 2.1 Introduction to Cryptography Theory

This chapter is based on the book [1]. The basic concepts in cryptography theory are mentioned here, skipping the proofs.

We begin by giving the definition of a cryptosystem.

**Definition 2.1** (Cryptosystem or encryption scheme)**.** A *cryptosystem* (or *encryption scheme*) can be defined as a family of algorithms $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$,[1] where:

- $\mathcal{K}_{\mathrm{E}}$ is a finite set with the keys used for encryption,
- $\mathcal{K}_{\mathrm{D}}$ is a finite set with the keys used for decryption,
- $\mathcal{P}$ is a set with the plaintexts,
- $\mathcal{C}$ is a set with the ciphertexts,
- $\mathsf{Gen} : \varnothing \to \mathcal{K}_{\mathrm{E}} \times \mathcal{K}_{\mathrm{D}}$ generates randomly encryption and decryption keys,
- $\mathsf{Enc} : \mathcal{K}_{\mathrm{E}} \times \mathcal{P} \to \mathcal{C}$ encrypts the given plaintext under the given encryption key; it may be randomized,
- $\mathsf{Dec} : \mathcal{K}_{\mathrm{D}} \times \mathcal{C} \to \mathcal{P}$ decrypts the given ciphertext under the given decryption key.

It makes sense to assume that $|\mathcal{P}| > 1$, else there is no point in communicating.

A simplifying convention (that also makes sense to assume) is that all the elements of $\mathcal{P}$ and $\mathcal{C}$ have nonzero probabilities.

This is a very general definition of a cryptosystem. For example, the above definition, being so general, allows a scheme to have keys, plaintexts, and ciphertexts with different sizes. Sometimes, it is needed that the cryptosystem adheres to certain specifications. This can be achieved through an extra parameter, the security parameter $\lambda$, which is used to configure things, like the key sizes, plaintext and ciphertext sizes, and the probabilistic behavior of $\mathsf{Gen}$ and $\mathsf{Enc}$; i.e., the elements of the scheme follow some specifications based on $\lambda$. For now, we ignore the sizes. Later, they will be taken into consideration.

One thing to note here is that $\mathsf{Gen}$ is sometimes presented in bibliography with an input parameter, the security parameter. Here, it is considered as a scheme parameter, instead of a $\mathsf{Gen}$ parameter.

[1] The sets $\mathcal{K}_{\mathrm{E}}, \mathcal{K}_{\mathrm{D}}, \mathcal{P}, \mathcal{C}$ may not appear in the $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ tuple (just for notation conciseness), but they are part of the cryptosystem, of course.

**Notation**: If the cryptosystem is parameterized by a security parameter $\lambda$, then it will be denoted as $\Sigma_\lambda$ instead of just $\Sigma$.

**Notation**: At $\mathsf{Enc}, \mathsf{Dec}$, the first argument will appear in subscript style, e.g., $\mathsf{Enc}_k(p) = \mathsf{Enc}(k, p)$.

For a cryptosystem to work properly, one must be able to retrieve the information that was previously encrypted.

**Definition 2.2** (Correctness)**.** For the correctness of the cryptosystem, it must hold that

$$\forall (e, d) \in \mathsf{Gen}(\varnothing), \forall p \in \mathcal{P},\ \Pr[\mathsf{Dec}_d(\mathsf{Enc}_e(p)) = p] = 1.$$

From the correctness definition, it is easily implied that $|\mathcal{C}| \geq |\mathcal{P}|$.

Cryptosystems are divided into two categories: secret-key (or symmetric) and public-key (or asymmetric) cryptosystems.

- Secret-key cryptosystems use the same key for both encryption and decryption. Hence, in the case of secret-key cryptosystems $\mathcal{K}_\mathrm{E} = \mathcal{K}_\mathrm{D}$ and $\mathsf{Gen}$ returns just one key.
- Public-key cryptosystems use two keys; one is used for encryption and the other one for decryption.

Now, we are ready to state formally the security of a cryptosystem, the perfect secrecy definition.

**Definition 2.3** (Perfect Secrecy)**.** A cryptosystem with plaintext set $\mathcal{P}$ and ciphertext set $\mathcal{C}$ is *perfectly secret* if and only if for every probability distribution over $\mathcal{P}$, every plaintext $p \in \mathcal{P}$ and every ciphertext $c \in \mathcal{C}$:

$$\Pr[P = p \mid C = c] = \Pr[P = p],$$

with $P, C$ being random variables over $\mathcal{P}$ and $\mathcal{C}$, respectively.

A cryptosystem is perfectly secret if and only if the distributions over plaintexts and ciphertexts are independent. In terms of Information Theory, this means that their mutual information is zero, so, no plaintext information is leaked after knowing the ciphertext.

There is a very important theorem which states that a secret-key cryptosystem is perfectly secret if and only if an attacker, given a ciphertext encrypting one of two arbitrary plaintexts, is unable to distinguish which plaintext of those two was encrypted.

In order to be able to state that theorem, we first define the following eavesdropper game $\mathrm{Eav}_{\Sigma}^{\mathcal{A}}$:

**Definition 2.4** (Eavesdropping attack)**.** For a cryptosystem $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and an adversary $\mathcal{A}$.[2]

1. An encryption key is generated:
   - $(k, \text{ignore}) \leftarrow \mathsf{Gen}()$
2. If $\Sigma$ is a public-key scheme, then $k$ is given to $\mathcal{A}$, else, it is kept secret.
3. $\mathcal{A}$ chooses two distinct plaintexts, $p_0, p_1 \in \mathcal{P}$, $p_0 \neq p_1$.
4. Without $\mathcal{A}$ knowing, a plaintext is chosen randomly (uniformly) between the two, and a ciphertext encrypting this plaintext under the generated key is computed:
   - $p \xleftarrow{\mathsf{R}} \{p_0, p_1\}$
   - $c \leftarrow \mathsf{Enc}_k(p)$
5. $\mathcal{A}$, having only the knowledge that was given to him in the above steps (i.e., $p_0, p_1$ and $k$ if and only if $\Sigma$ is a public-key scheme, no usage of the $\Sigma$ algorithms), is given the ciphertext $c$, from which tries to retrieve the original plaintext:
   - $c$ is given to $\mathcal{A}$;
   - $\mathcal{A}$ outputs a plaintext $p' \in \{p_0, p_1\}$;
   - if $p' = p$, then $\mathcal{A}$ succeeded, and it is denoted as $\mathrm{Eav}_{\Sigma}^{\mathcal{A}} = 1$,
   - else $\mathcal{A}$ failed, and it is denoted as $\mathrm{Eav}_{\Sigma}^{\mathcal{A}} = 0$.

**Theorem 2.5** (Adversarial indistinguishability)**.** *A secret-key cryptosystem $\Sigma$ is perfectly secret if and only if for every adversary $\mathcal{A}$ it holds that*

$$\Pr\left[\mathrm{Eav}_{\Sigma}^{\mathcal{A}} = 1\right] = \frac{1}{2}.$$

The above theorem shows the security of a secret-key cryptosystem from an attacker point of view, which is a more intuitive way to think of it. It states that a cryptosystem is perfectly secret if and only if any adversary can guess the original plaintext with the

[2] $\mathcal{A}$ can be thought as an algorithm that tries to "break" the system.

probability of a total random guess. No matter how sophisticated the algorithm $\mathcal{A}$ may be, the probability of success will be the same as in a random coin toss. It is a very important theorem, since it states that adversarial indistinguishability between any two (but just two) plaintexts is equivalent to perfect secrecy.

Note that we have assumed no limitation on the computational power of $\mathcal{A}$.

One such perfectly secret cryptosystem is the One-Time Pad.

For a plaintext length parameter $\ell \in \mathbb{Z}^{\geq 1}$:

- $\mathcal{K} = \mathcal{P} = \mathcal{C} = \{0,1\}^\ell$ (all the binary strings of length $\ell$)
- $\mathsf{Gen}$ chooses a key $k$ from $\mathcal{K}$ according to the uniform distribution
- $\mathsf{Enc}_k(p) = k \oplus p$, where $\oplus$ denotes the bitwise XOR
- $\mathsf{Dec} = \mathsf{Enc}$

For the correctness proof, we have that $\forall k \in \mathcal{K}$ and $\forall p \in \mathcal{P}$:

$$\mathsf{Dec}_k(\mathsf{Enc}_k(p)) = k \oplus \mathsf{Enc}_k(p) = k \oplus k \oplus p = p.$$

Intuitively, the One-Time Pad is perfectly secret because every bit is changing or not, totally randomly and independently of the other ones. The attacker has nothing to help him make a better guess than choosing at random.

However, perfect secrecy (or adversarial indistinguishability) comes with an inherent overhead.

**Theorem 2.6** (Perfect secrecy key size)**.** *Let a perfectly secret cryptosystem with key set $\mathcal{K}$ plaintext set $\mathcal{P}$. Then $|\mathcal{K}| \geq |\mathcal{P}|$.*

For the problem to become more evident, imagine the case where $\mathcal{K}$ contains fixed-length bit-strings for keys and $\mathcal{P}$ contains fixed-length bit-strings for plaintexts. This implies that the key must be at least as long as the message. So, this is not just the case for the One-Time Pad, but for every perfectly secret cryptosystem.

To address this, we must relax the strict security requirements. It is unrealistic to aim for security against adversaries with unlimited computational power. In the real world, there are limitations, and taking those into consideration enables more feasible key sizes at the expense of perfect secrecy.

To begin with, we constrain the computational power of adversaries to probabilistic polynomial-time (PPT) algorithms. This means that $\mathcal{A}$ terminates after a polynomial number of steps with respect to some parameter. This parameter will be the cryptosystem security parameter $\lambda$, and the PPT adversary will be denoted as $\mathcal{A}_\lambda$.

The new relaxed eavesdropping attack game will be the same as above, but with the security parameterized cryptosystem $\Sigma_\lambda$ and the PPT adversary $\mathcal{A}_\lambda$, namely $\mathrm{Eav}_{\Sigma_\lambda}^{\mathcal{A}_\lambda}$.

Since the computational power is not taken into consideration, we will also consider the elements of the sets $\mathcal{K}, \mathcal{P}, \mathcal{C}$ to be bit-strings of some length determined by the security parameter $\lambda$ (each set having its own length); more specifically, the aforementioned bit-string lengths are polynomial in $\lambda$.

To proceed with the definition of the computationally relaxed definition of security, we first define the notion of negligible success probability. In the same way that we consider polynomial running times to be feasible, we consider inverse-polynomial probabilities to be significant. Probabilities that are asymptotically smaller than any inverse polynomial are what we will define as negligible probabilities.

**Definition 2.7.** A function $f$ is negligible if and only if for every polynomial $p(\cdot)$ there exists an $N$ such that for all integers $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.

**Lemma 2.8.** *Let* $\mathrm{negl}_1$ *and* $\mathrm{negl}_2$ *be negligible functions.*

- *The function* $\mathrm{negl}_3$ *defined by* $\mathrm{negl}_3(n) = \mathrm{negl}_1(n) + \mathrm{negl}_2(n)$ *is negligible.*
- *For any positive polynomial* $p$*, the function* $\mathrm{negl}_4$ *defined by* $\mathrm{negl}_4(n) = p(n) \cdot \mathrm{negl}_1(n)$ *is negligible.*

Now we are ready to proceed to the computationally relaxed definition of security.

**Definition 2.9** (PPT eavesdropper indistinguishability)**.** A secret-key cryptosystem $\Sigma_\lambda$ has indistinguishable encryptions in the presence of a PPT eavesdropper if and only if for all PPT adversaries $\mathcal{A}_\lambda$ there exists a negligible function negl such that

$$\Pr\left[\mathrm{Eav}_{\Sigma_\lambda}^{\mathcal{A}_\lambda} = 1\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

In the above scenarios, the attacker may be the one who chooses which two plaintexts will be involved in the experiment, but beyond that, they remain completely passive.

For this reason, we introduce two additional experiments where the attacker it actively involved in the encryption and decryption processes. These experiments are known as the Chosen Plaintext Attack (CPA) and the Chosen Ciphertext Attack (CCA).

Before proceeding to the definitions of CPA and CCA, it is essential to introduce the notion of "oracle" in cryptography.

In cryptography, an oracle is an abstract entity or algorithm that provides answers to specific queries. It is a black-box function that given an input returns an output. The potential of an oracle does not have to be realistic. Often, it is assumed that an adversary has access to an oracle capable of solving in $O(1)$ a problem, even if no known efficient algorithm exists for that problem. This assumption allows us to overestimate the adversary's capabilities and show that the system is secure in these conditions, ensuring its security even when the adversary has less potential. Moreover, if the problem that the oracle is assumed to solve is an open problem, we can ensure security, even if future research discovers an efficient algorithm for it.

We begin with CPA.

**Definition 2.10** (Chosen Plaintext Attack (CPA))**.** For a cryptosystem $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and an adversary $\mathcal{A}$.

1. An encryption key is generated:
   - $(k, \text{ignore}) \leftarrow \mathsf{Gen}()$
2. If $\Sigma$ is a public-key scheme, then $k$ is given to $\mathcal{A}$, else, it is kept secret.
3. $\mathcal{A}$ is given oracle access to $\mathsf{Enc}_k(\cdot)$.
4. $\mathcal{A}$ chooses two distinct plaintexts of the same length, $p_0, p_1 \in \mathcal{P}$, $p_0 \neq p_1$.
5. Without $\mathcal{A}$ knowing, a plaintext is chosen randomly (uniformly) between the two, and a ciphertext encrypting this plaintext under the generated key is computed:
   - $p \xleftarrow{\mathsf{R}} \{p_0, p_1\}$
   - $c \leftarrow \mathsf{Enc}_k(p)$
6. $\mathcal{A}$ continues to have oracle access to $\mathsf{Enc}_k(\cdot)$.
7. $\mathcal{A}$, having only the knowledge and access that was given to him in the above steps, is given the ciphertext $c$, from which tries to retrieve the original plaintext:
   - $c$ is given to $\mathcal{A}$;
   - $\mathcal{A}$ outputs a plaintext $p' \in \{p_0, p_1\}$;
   - if $p' = p$, then $\mathcal{A}$ succeeded, and it is denoted as $\text{CPA}_\Sigma^\mathcal{A} = 1$,

- else $\mathcal{A}$ failed, and it is denoted as $\text{CPA}_{\Sigma}^{\mathcal{A}} = 0$.

In a nutshell, this attack looks a lot like the eavesdropping attack, but the adversary now also has access to the $\mathsf{Enc}_k$ algorithm to use it as they wish. Note that in this attack $\mathcal{A}$ has access to the encryptor even in the scenario of a secret-key cryptosystem, though, in this case, $\mathcal{A}$ does not have access to the key.

In bibliography, the above attack scenario is often called CPA2 or adaptive-CPA, since the attacker is given oracle access to the encryptor even after knowing the challenge ciphertext. In this thesis, we call it just CPA, ignoring the other more restrictive variant.

The following definition formalizes when a cryptosystem is considered secure against CPA.

**Definition 2.11** (PPT CPA indistinguishability)**.** An either secret-key or public-key cryptosystem $\Sigma_\lambda$ has indistinguishable encryptions under a PPT chosen-plaintext attack (or is CPA-secure) if for all PPT adversaries $\mathcal{A}_\lambda$ there exists a negligible function negl such that

$$\Pr\left[\text{CPA}_{\Sigma_\lambda}^{\mathcal{A}_\lambda} = 1\right] \leq \frac{1}{2} + \text{negl}(\lambda).$$

We continue with CCA. Roughly, CCA is like CPA but with $\mathcal{A}$ having access to a decryption oracle, too. The only thing that $\mathcal{A}$ is not permitted to request from the decryption oracle is the decryption of the challenge ciphertext. CCA is the strongest notion of security in encryption schemes.

**Definition 2.12** (Chosen Ciphertext Attack (CCA))**.** For a cryptosystem $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and an adversary $\mathcal{A}$.

1. An encryption key and a decryption key are generated:
   - $(k, k') \leftarrow \mathsf{Gen}()$
2. If $\Sigma$ is a public-key scheme, then $k$ is given to $\mathcal{A}$, else, it is kept secret. (Of course, if $\Sigma$ is a secret-key cryptosystem, then $k = k'$.)
3. $\mathcal{A}$ is given oracle access to $\mathsf{Enc}_k(\cdot)$ and $\mathsf{Dec}_{k'}(\cdot)$.
4. $\mathcal{A}$ chooses two distinct plaintexts of the same length, $p_0, p_1 \in \mathcal{P}$, $p_0 \neq p_1$.
5. Without $\mathcal{A}$ knowing, a plaintext is chosen randomly (uniformly) between the two, and a ciphertext encrypting this plaintext under the generated key is computed:

- $p \xleftarrow{\mathsf{R}} \{p_0, p_1\}$
- $c \leftarrow \mathsf{Enc}_k(p)$

6. $\mathcal{A}$ continues to have oracle access to $\mathsf{Enc}_k(\cdot)$ and $\mathsf{Dec}_{k'}(\cdot)$, but it is forbidden to query the decryption oracle for $c$.
7. $\mathcal{A}$, having only the knowledge and access that was given to him in the above steps, is given the ciphertext $c$, from which tries to retrieve the original plaintext:
   - $c$ is given to $\mathcal{A}$;
   - $\mathcal{A}$ outputs a plaintext $p' \in \{p_0, p_1\}$;
   - if $p' = p$, then $\mathcal{A}$ succeeded, and it is denoted as $\mathrm{CCA}_{\Sigma}^{\mathcal{A}} = 1$,
   - else $\mathcal{A}$ failed, and it is denoted as $\mathrm{CCA}_{\Sigma}^{\mathcal{A}} = 0$.

Again, in bibliography, the above attack scenario is often called CCA2 or adaptive-CCA, since the attacker is given oracle access to the decryptor even after knowing the challenge ciphertext. In this thesis, we call it just CCA, ignoring the other more restrictive variant.

Similarly to CPA, the following definition formalizes when a cryptosystem is considered secure against CCA.

**Definition 2.13** (PPT CCA indistinguishability)**.** An either secret-key or public-key cryptosystem $\Sigma_\lambda$ has indistinguishable encryptions under a PPT chosen-ciphertext attack (or is CCA-secure) if for all PPT adversaries $\mathcal{A}_\lambda$ there exists a negligible function negl such that

$$\Pr\left[\mathrm{CCA}_{\Sigma_\lambda}^{\mathcal{A}_\lambda} = 1\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

## 2.2 Useful Tools from Number Theory and Abstract Algebra

In this section, we introduce just some basic and useful tools from Number Theory and Abstract Algebra that are going to be used in the following theory of this thesis.

We begin with number theory.

**Definition 2.14.** For any $n \in \mathbb{Z}^{\geq 1}$, define $\mathbb{Z}_n \triangleq \{0, \ldots, n-1\}$.

**Definition 2.15.** For any $n, d \in \mathbb{Z}$ with $d \neq 0$, define $r_d[n]$ to to be the remainder of the division of $n$ by $d$, i.e., $n = qd + r_d[n]$ for some $q \in \mathbb{Z}$ such that $r_d[n] \in \mathbb{Z}_{|n|}$.

**Definition 2.16.** For any $n, d \in \mathbb{Z}$ with $d \neq 0$, if $d$ divides $n$, denote it as $d \mid n$, otherwise, $d \nmid n$.

**Definition 2.17.** For any $a, b, n \in \mathbb{Z}$, $a$ and $b$ are congruent modulo $n$ if and only if $n$ divides $a - b$; in symbols, it is denoted as

$$a \equiv b \pmod{n} \iff n \mid a - b.$$

Otherwise, if $n$ does not divide $a - b$, it is denoted as $a \not\equiv b \pmod{n}$.

**Definition 2.18** (Primitive $n$-th root of unity)**.** Let $q, n, \omega \in \mathbb{Z}$. $\omega$ is a primitive $n$-th root of unity in $\mathbb{Z}_q$ if and only if

$$\omega^n \equiv 1 \pmod{q},$$

and for all $k \in \mathbb{Z}_n$

$$\omega^k \not\equiv 1 \pmod{q}.$$

Now, we proceed to abstract algebra.

**Definition 2.19** (Group)**.** Let a set $G$ and a binary operation $* : G \times G \to G$. $(G, *)$ is called a *group* if and only if:

- $\forall a, b, c \in G : (a * b) * c = a * (b * c)$
- $\exists e \in G$ such that $\forall a \in G$: $e * a = a * e = a$ ($e$ is called the identity element of the group)
- $\forall a \in G$: $\exists b \in G$ such that $a * b = b * a = e$, where $e$ is the identity element as defined above ($b$ is called the inverse of $a$ with respect to $*$; the expression "w.r.t. $*$" will be omitted if the operation is implicitly implied)

The $(G, *)$ group is called an abelian group if and only if its operation is also commutative, i.e., $\forall a, b \in G$: $a * b = b * a$.

**Definition 2.20** (Subgroup)**.** Let $G, H$ be two sets with $H \subseteq G$. Let $(G, *), (H, *)$ be two groups that have the same operation, but, in the second group, the operation is restricted to the elements of $H$. Then $(H, *)$ is called a *subgroup* of $(G, *)$, and it is denoted as $(H, *) \leq (G, *)$.

**Definition 2.21** (Left and right cosets)**.** Let two groups $(G, *), (H, *)$ such that $(H, *) \leq (G, *)$. For any $g \in G$ define the *left coset* of $(H, *)$ in $(G, *)$ as the set containing the elements $g * h$ for all $h \in H$, and denote it as $g * H$, i.e.,

$$g * H \triangleq \{g * h \mid h \in H\}.$$

For any $g \in G$, the *right coset* of $(H, *)$ in $(G, *)$ is

$$H * g \triangleq \{h * g \mid h \in H\}.$$

**Definition 2.22** (Normal subgroup)**.** Let two groups $(G, *), (N, *)$ such that $(N, *) \leq (G, *)$. $N, *$ is called a *normal subgroup* of $(G, *)$, and it is denoted as $(N, *) \triangleleft (G, *)$, if and only if for all $g \in G$

$$g * N * g^{-1} = N.$$

**Definition 2.23** (Quotient group)**.** Let two groups $(G, *), (N, *)$ such that $(N, *)$ is a normal subgroup of $(G, *)$. Define $G/N$ to be the set of all the left cosets of $(N, *)$ in $(G, *)$, i.e.,

$$G/N \triangleq \{g * N \mid g \in G\}.$$

Define the $*$ operation[3] for the elements of $G/N$ like this: for $g_1 * N, g_2 * N \in G/N$,

$$(g_1 * N) * (g_2 * N) = (g_1 * g_2) * N.$$

Let $e \in N$ be the identity element of $(N, *)$. Then, $e * N$ is the identity element of $(G/N, *)$.

Let $g \in G$, and $g^{-1} \in G$ be its inverse in $(G, *)$. Then, the inverse of $g * N$ in $(G/N, *)$ is $g^{-1} * N$.

Hence, $(G/N, *)$ is a group, and it is called the *quotient group* of $(G, *)$ by $(N, *)$.

[3]Careful: we may use the same symbol to denote the operation in order to show the connection with the operation of the initial group. However, the context of the operations is different, hence, strictly speaking, the operation is different.

**Definition 2.24** (Group homomorphism)**.** Let two groups $(G, *), (H, \diamond)$. Let a function $f : G \to H$. $f$ is called a *group homomorphism* from $(G, *)$ to $(H, \diamond)$ if and only if for all $u, v \in G$

$$f(u * v) = f(u) \diamond f(v).$$

Two immediate corollaries of the above definition are that a group homomorphism maps identities to identities and inverses to inverses:

- $f(e_G) = e_H$ , where $e_G, e_H$ are the identity elements of $(G, *), (H, \diamond)$, respectively.
- $\forall g \in G$ it holds that $f(g^{-1}) = f(g)^{-1}$

**Definition 2.25** (Group isomorphism)**.** Let $f$ be a group homomorphism from one group to another one. $f$ is a *group isomorphism* if and only if $f$ is a bijection.

**Definition 2.26** (Kernel of a group homomorphism)**.** Let two groups $(G, *), (H, \diamond)$. Let a group homomorphism $f$ from $(G, *)$ to $(H, \diamond)$. The *kernel* of $f$ is defined as the set of elements of $G$ that $f$ maps to the identity element of $(H, \diamond)$, i.e.,

$$\ker(f) = \{g \in G \mid f(g) = e_H\}.$$

Three immediate corollaries of the above definition are:

- $\ker(f) \supseteq \{e_G\}$
- $f$ is not an injection $\iff kef(f) \supset e_G$
- $(\ker(f), *) \leq (G, *)$

**Definition 2.27** (Ring)**.** Let a set $R$ and the binary operations $+ : R \times R \to R$ and $\cdot : R \times R \to R$. $(R, +, \cdot)$ is called a *ring* if and only if:

- $(R, +)$ is an abelian group
- $\forall a, b, c \in R : (a \cdot b) \cdot c = a \cdot (b \cdot c)$
- $\forall a, b, c \in R : a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
- $\forall a, b, c \in R : (b + c) \cdot a = (b \cdot a) + (c \cdot a)$

The ring $(R, +, \cdot)$ is called a *commutative ring* if and only if the $\cdot$ operation is also commutative, i.e., $\forall a, b \in R$: $a \cdot b = b \cdot a$.

The ring $(R, +, \cdot)$ is called a *ring with identity* if and only if the $\cdot$ operation has an identity element, i.e., $\exists 1 \in R$ such that $\forall a \in R : 1 \cdot a = a \cdot 1 = a$.

In ring theory, as concepts, "ideals" correspond to "subgroups" in group theory, "ideal cosets" or "residue classes" correspond to "cosets," and "quotient rings" correspond to "quotient groups."

**Definition 2.28** (Left and right ideal)**.** Let a ring $(R, +, \cdot)$ and a set $I$ such that $I \subseteq R$. $(I, +, \cdot)$ is called a *left ideal* of $(R, +, \cdot)$ if and only if:

- $(I, +)$ is a subgroup of $(R, +)$
- $\forall r \in R, \forall i \in I$ it holds that $r \cdot i \in I$ (or, more concisely expressed, $R \cdot I \subseteq I$)

A *right ideal* is defined similarly, with the only difference being that the last condition is replaced by $i \cdot r \in I$ (or, more concisely, $I \cdot R \subseteq I$).

A *two-sided ideal* is a left ideal that is also a right ideal.

**Definition 2.29** (Residue class)**.** Let a ring $(R, +, \cdot)$ and a (left or right) ideal $(I, +, \cdot)$ of $(R, +, \cdot)$. The *residue class* of an element $r \in R$ is defined as

$$r + I \triangleq \{r + i \mid i \in I\}$$

This residue class is also sometimes written as $r \bmod I$ or as $[r]$ if $I$ is implicitly implied.

**Definition 2.30** (Quotient ring)**.** Let a ring $(R, +, \cdot)$ and a (left or right) ideal $(I, +, \cdot)$ of $(R, +, \cdot)$. Define $R/I$ to be the set of all the residue classes generated by $(I, +, \cdot)$, i.e.,

$$R/I \triangleq \{r + I \mid r \in R\}.$$

Define the $+, \cdot$ operations[4] for the elements of $R/I$ like this: for $r_1 + I, r_2 + I \in R/I$,

- $(r_1 + I) + (r_2 + I) = (r_1 + r_2) + I$
- $(r_1 + I) \cdot (r_2 + I) = (r_1 \cdot r_2) + I$

The additive identity element of $R/I$ is $(0 + I)$. The multiplicative identity element of $R/I$ is $(1 + I)$.

Hence, $(R/I, +, \cdot)$ is a ring, and it is called the *quotient ring* of $(R, +, \cdot)$ modulo $(I, +, \cdot)$.

**Definition 2.31** (Field)**.** Let a set $F$ and the binary operations $+ : F \times F \to F$ and $\cdot : F \times F \to F$. $(F, +, \cdot)$ is called a *field* if and only if:

[4]About the notation of the operations, the same thing that was highlighted in the quotient group definition applies here, too.

- $(F, +)$ is an abelian group
- $(F \setminus \{0\}, \cdot)$ is an abelian group

**Definition 2.32** (Polynomial)**.** Let $S$ be a set. A *polynomial* of degree $n$ (for $n \in \mathbb{Z}^{\geq 0}$) with coefficients in $S$ is an expression

$$a_0 + a_1 x + \cdots + a_n x^n,$$

where $a_i \in S$ for $i \in \{0, \ldots, n\}$. Denote as $S[x]$ the set of all the polynomials with coefficients in $S$ with finite degree, i.e.,

$$S[x] \triangleq \left\{ \sum_{i=0}^{n} a_i x^i \;\middle|\; a_i \in S, n \in \mathbb{Z}^{\geq 0} \right\}.$$

Two polynomials $f(x)$ and $g(x)$ are considered equal, even if they are of different degrees, as long as their corresponding coefficients are equal and the higher-degree terms of the longer polynomial are zero. In other words, $f(x) = g(x)$ if and only if there exists a degree $n$ such that $f(x) = \sum_{i=0}^{n} a_i x^i$ and $g(x) = \sum_{i=0}^{n} b_i x^i$ with $m \leq n$, where $a_i = b_i$ for all $i \leq m$ and $a_i = 0$ for all $i > m$.

A polynomial in abstract algebra is just an ordered sequence of elements. $x$ is called *indeterminant* and has clearly symbolic role, it is not used for the "classical" polynomial evaluation.

Sometimes, for conciseness, when it is clear from the context that $f(x)$ is a polynomial, and it is not useful that the indeterminant $x$ appears in the expression, $f(x)$ will be denoted as just $f$.

A polynomial of any degree may have leading zeros. Define the deg operator such that it ignores the leading zeros and returns the "effective" degree of a polynomial.

**Definition 2.33** ("Effective" degree of a polynomial)**.** Let a polynomial $f(x) = \sum_{i=0}^{n} a_i x^i$ of degree $n$.

$$\deg(f(x)) \triangleq \begin{cases} n & \text{if } \; a_n \neq 0 \\ -\infty & \text{if } \; a_i = 0 \;\; \forall i \in \mathbb{Z}_{n+1} \\ \min\left\{ k \in \mathbb{Z}_n \;\middle|\; a_i = 0 \;\; \forall i \in \mathbb{Z}_{n+1}^{\geq k+1} \right\} & \text{otherwise} \end{cases}$$

**Definition 2.34** (Monic polynomials)**.** A polynomial $f(x) = \sum_{i=0}^{n} a_i x^i$ is called *monic* if and only if $a_{\deg(f)} = 1$.

**Definition 2.35** (Polynomial ring)**.** Let a ring $(R, +, \cdot)$. Then, $(R[x], \oplus, \odot)$ is a ring of polynomials (or a *polynomial ring*), where the operations are defined as follows. For any two polynomials $f(x) = \sum_{i=0}^{m} a_i x^i, g(x) = \sum_{i=0}^{n} b_i x^i \in R[x]$:

- The $\oplus$ operation is just the elementwise $+$ operation between the corresponding coefficients (like the "classical" polynomial addition):

$$f(x) \oplus g(x) = \sum_{i=0}^{\max(m,n)} (a_i + b_i) x^i$$

- The $\odot$ operation is the convolution between the vectors of coefficients (like the "classical" polynomial multiplication):

$$f(x) \odot g(x) = \sum_{i=0}^{m+n} \left( \sum_{j=0}^{i} a_j \cdot b_{i-j} \right) x^i,$$

**Definition 2.36** (Polynomial quotient ring)**.** For a ring $(R, +, \cdot)$, let the polynomial ring $(R[x], \oplus, \odot)$. Let a polynomial $f(x) \in R[x]$ and define the left ideal[5] $(I[x], \oplus, \odot)$ where

$$I[x] \triangleq R[x] \odot f(x) = \{r(x) \odot f(x) \mid r(x) \in R[x]\}.$$

Then, the *polynomial quotient ring* is defined as the quotient ring $(R[x]/I[x], \oplus, \odot)$. The above may be denoted in the simpler notation $R[x]/f(x)$.

Informally, the polynomial quotient ring is constructed by taking the set of all polynomials in $R[x]$ and imposing the relation $f(x) = 0$, hence, $R[x]/f(x)$ contains all the polynomials with degree less than $\deg(f)$.[6]

For example, let $R = \mathbb{Z}_2$, the operations being the classical addition and multiplication modulo 2, and $f(x) = x^2 + 1$. Then, some examples for $\mathbb{Z}_2[x]/(x^2 + 1)$ are:

- $0, 1, x, 1 + x \in \mathbb{Z}_2[x]/(x^2 + 1)$
- $x^2 + 1 = 0$

[5] It could also be the right ideal; however, in most cases in cryptography that does not matter since, usually, the ring is commutative.

[6] Strictly speaking, $R[x]/f(x)$ is isomorphic to the set that contains all the polynomials with degree less than $\deg(f)$; however, we will ignore this theoretical technicality henceforth.

- $x^3 + x = (x^2 + 1) \odot x = 0 \odot x = 0$
- $x^3 = x^3 + (1 + 1)x = (x^3 + x) \oplus x = 0 \oplus x = x$

In the following chapters, when referring to a group/ring/field where the endowed operations are implicitly implied, the full notation may be simplified; e.g., for a ring, instead of $(R, +, \cdot)$, we may write it just $R$.

In addition, the notation in operations will be relaxed, e.g., the multiplication symbol may be completely omitted, and the polynomial addition and multiplication symbols will be the same with the underlying ring.

## 2.3 Introduction to Lattice Theory

This chapter is primarily based on the lectures from Regev's course [2] and [3]. Again, the basic concepts in lattice theory are mentioned here, skipping the proofs.

In this chapter, when referring to the length of a vector $x \in \mathbb{R}^n$, we mean the Euclidean norm, or the $\ell_2$ norm, defined as $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$. For notational clarity, we drop the index, and denote this norm simply by $\|\mathbf{x}\|$.

In addition, for $\mathbf{v} \in \mathbb{R}^n$ and $r \geq 0$, $\overline{\mathcal{B}}(\mathbf{v}, r) = \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x}\| \leq r\}$ is the closed ball of radius $r$ around $\mathbf{v}$.

### 2.3.1 Basic Definitions and Theorems

We begin by defining and understanding the concept of a lattice.

Informally, it is a set of discrete points in $m$-dimensional space with a periodic structure, for example, such as the one illustrated in Figure 1.

Formally, a lattice is defined as a discrete additive subgroup of $\mathbb{R}^m$.

**Definition 2.37** (Lattice)**.** $\mathcal{L}$ is an $m$-dimensional lattice (or, a lattice in $\mathbb{R}^m$) if and only if:

- $\mathcal{L}$ is an additive subgroup of $\mathbb{R}^m$:

  $\mathbf{0} \in \mathcal{L}$, and $-\mathbf{x}, \mathbf{x} + \mathbf{y} \in \mathcal{L}$ for all $\mathbf{x}, \mathbf{y} \in \mathcal{L}$,
- $\mathcal{L}$ is discrete:

  Every $\mathbf{x} \in \mathcal{L}$ has a neighborhood in $\mathbb{R}^m$ in which $\mathbf{x}$ is the only lattice point, i.e., there exists some $\varepsilon > 0$ such that $\overline{\mathcal{B}}(\mathbf{x}, r) \cap \mathcal{L} = \{\mathbf{x}\}$ for all $x \in \mathcal{L}$.[7]

[7] That means that there exists some $\varepsilon > 0$ such that $\|\mathbf{x} - \mathbf{y}\| > \varepsilon$ for all $\mathbf{x}, \mathbf{y} \in \mathcal{L}$ with $\mathbf{x} \neq \mathbf{y}$; this is important for defining the minimum distance of a lattice below.

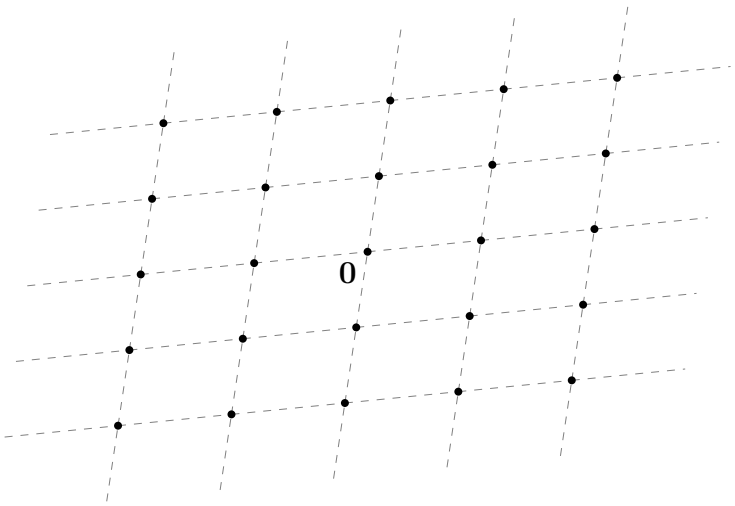



Figure 1: A lattice in $\mathbb{R}^2$.

An alternative and equivalent way to define a lattice is through its basis.

**Definition 2.38** (Lattice)**.** Let $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$ be linearly independent vectors. The lattice generated by them is defined as

$$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) \triangleq \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i \;\middle|\; x_i \in \mathbb{Z} \right\}.$$

We refer to $\mathbf{b}_1, \ldots, \mathbf{b}_n$ as a basis of the lattice. Equivalently, if we define $\mathbf{B}$ as the $m \times n$ matrix whose columns are $\mathbf{b}_1, \ldots, \mathbf{b}_n$, then the lattice generated by $\mathbf{B}$ is

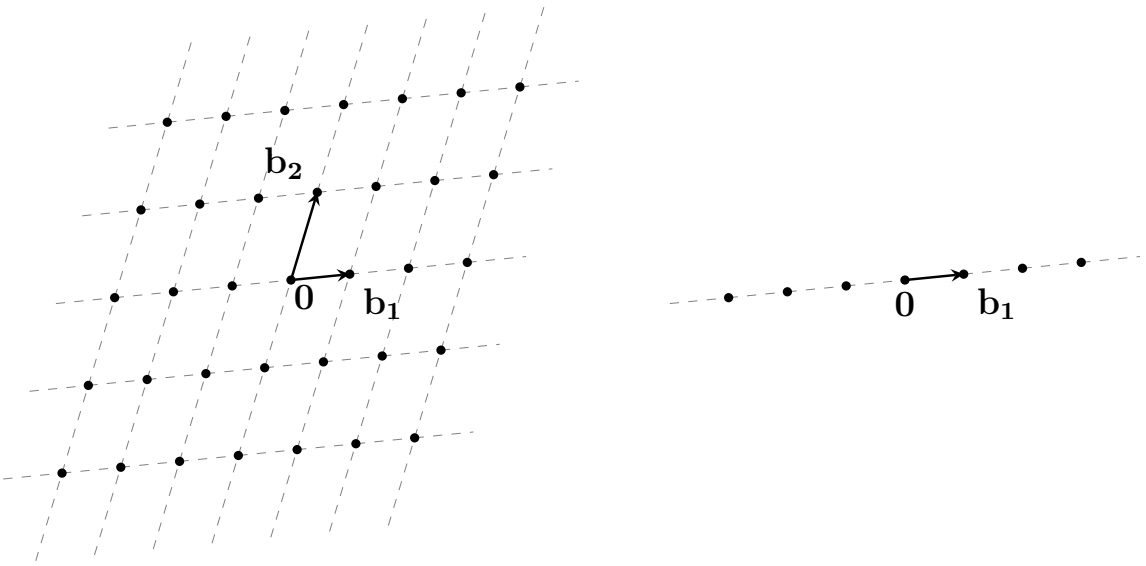
$$\mathcal{L}(\mathbf{B}) \triangleq \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{\mathbf{Bx} \mid \mathbf{x} \in \mathbb{Z}^n\}.$$

We proceed with the definition of the rank and the dimension of a lattice.

**Definition 2.39** (Lattice rank and dimension)**.** Let $\mathcal{L}$ be a lattice in $\mathbb{R}^m$. Let $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$ be a basis (namely, $\mathbf{b}_i$ are linearly independent) of $\mathcal{L}$. The rank of $\mathcal{L}$ is $n$, and it is denoted as $\text{rank}(\mathcal{L})$. The dimension of $\mathcal{L}$ is $m$, and it is denoted as $\dim(\mathcal{L})$.

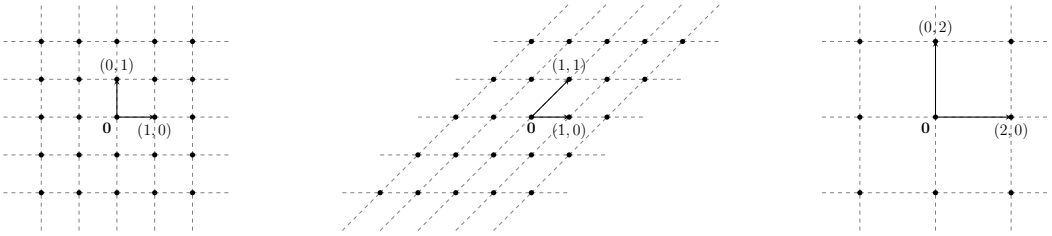
It is easily understandable that, since the $\mathbf{b}_i$ vectors in the above definitions are linearly independent, $n$ cannot be greater than $m$, i.e., $\text{rank}(\mathcal{L}) \leq \dim(\mathcal{L})$. Informally, when the rank of a lattice achieves its maximum value, i.e., $\text{rank}(\mathcal{L}) = \dim(\mathcal{L})$, then the lattice is "expanded" to all the possible dimensions. See Figure 2.

**Definition 2.40** (Full-rank lattice)**.** A lattice $\mathcal{L}$ is called a full-rank lattice, if and only if $\text{rank}(\mathcal{L}) = \dim(\mathcal{L})$.




(a) A full-rank lattice in $\mathbb{R}^2$. (b) A non-full-rank lattice in $\mathbb{R}^2$.

Figure 2: Lattices in $\mathbb{R}^2$ with their bases.




(a) A basis of $\mathbb{Z}^2$. (b) Another basis of $\mathbb{Z}^2$. (c) Not a basis of $\mathbb{Z}^2$.

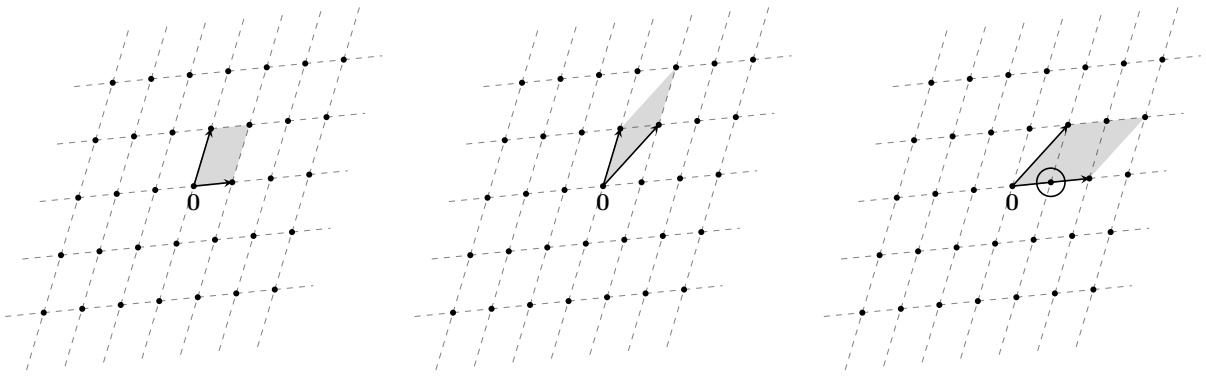
Figure 3: The $\mathbb{Z}^2$ lattice and some candidate bases.

Henceforth, we will focus on full-rank lattices, as the more general case does not differ significantly.

**Definition 2.41** (Lattice span)**.** Let $\mathbf{B} \in \mathbb{R}^{m \times n}$. The span of a lattice $\mathcal{L}(\mathbf{B})$ is the linear space spanned by its vectors,

$$\operatorname{span}(\mathcal{L}(\mathbf{B})) \triangleq \operatorname{span}(\mathbf{B}) = \{\mathbf{Bx} \mid \mathbf{x} \in \mathbb{R}^n\}.$$

It is easy to see that a basis of a lattice may not be unique. There may be another basis generating the same lattice, i.e., $\mathcal{L}(\mathbf{B}_1) = \mathcal{L}(\mathbf{B}_2)$. However, given a lattice $\mathcal{L}$ with $n = \operatorname{rank}(\mathcal{L})$, not every set of $n$ linearly independent vectors in $\mathcal{L}$ is a basis of $\mathcal{L}$. See Figure 3.




(a) A basis for the underlying lattice, no point in the fundamental parallelepiped.

(b) This is a basis, too.

(c) Not a basis for the underlying lattice, a point (circled) is in the fundamental parallelepiped.

Figure 4: The fundamental parallelepiped of a basis.

The first question we address is: how can we determine if a given set of vectors forms a basis for a lattice?

To answer this question, we first introduce the definition of the fundamental parallelepiped. Then, the following theorem provides a solution to this. See Figure 4

**Definition 2.42** (Fundamental parallelepiped)**.** For any lattice basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$ we define

$$\mathcal{P}(\mathbf{b}_1, \ldots, \mathbf{b}_n) \triangleq \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i \;\middle|\; \forall i : 0 \leq x_i < 1 \right\}.$$

Equivalently, if $\mathbf{b}_1, \ldots, \mathbf{b}_n$ are the columns of $\mathbf{B}$, then

$$\mathcal{P}(\mathbf{B}) \triangleq \mathcal{P}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{\mathbf{Bx} \mid \mathbf{x} \in [0,1)^n\}.$$

In bibliography, in the above definition, sometimes the $x_i$ interval is $\left[-\frac{1}{2}, \frac{1}{2}\right)$, but this does not change anything in the following analysis.

**Theorem 2.43.** *Let $\mathcal{L}$ be a lattice of rank $n$, and let $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathcal{L}$ be $n$ linearly independent lattice vectors. Then $\mathbf{b}_1, \ldots, \mathbf{b}_n$ form a basis of $\mathcal{L}$ if and only if*

$$\mathcal{P}(\mathbf{b}_1, \ldots, \mathbf{b}_n) \cap \mathcal{L} = \varnothing.$$

The second question we address is: how can we determine if two given bases $\mathbf{B}_1, \mathbf{B}_2$ are equivalent, meaning that they generate the same lattice, i.e., $\mathcal{L}(\mathbf{B}_1) = \mathcal{L}(\mathbf{B}_2)$.

To answer this question, we introduce the definition of the unimodular matrix. Then, the following theorem provides a solution.

**Definition 2.44** (Unimodular matrix)**.** A matrix $\mathbf{U} \in \mathbb{Z}^{n\times n}$ is called unimodular if and only if $\det(\mathbf{U}) = \pm 1$.

**Lemma 2.45.** *If* $\mathbf{U}$ *is unimodular, then* $\mathbf{U}^{-1}$ *is also unimodular, and in particular* $\mathbf{U}^{-1} \in \mathbb{Z}^{n\times n}$.

**Theorem 2.46.** *Two bases* $\mathbf{B}_1, \mathbf{B}_2 \in \mathbb{R}^{m\times n}$ *are equivalent if and only if* $\mathbf{B}_2 = \mathbf{B}_1\mathbf{U}$ *for some unimodular matrix* $\mathbf{U}$.

An equivalent solution to the bases equivalence is the following theorem.

**Theorem 2.47.** *Two bases are equivalent if and only if one can be obtained from the other by the following operations on columns:*

- $\mathbf{b}_i \leftarrow \mathbf{b}_i + k\mathbf{b}_j$ *for some* $k \in \mathbb{Z}$
- $\mathbf{b}_i \leftrightarrow \mathbf{b}_j$
- $\mathbf{b}_i \leftarrow -\mathbf{b}_i$

Another basic notion of lattices is the determinant of a lattice.

**Definition 2.48** (Lattice determinant)**.** Let $\mathcal{L} = \mathcal{L}(\mathbf{B})$ be a lattice of rank $n$. We define the determinant of $\mathcal{L}$, denoted $\det(\mathcal{L})$, as the $n$-dimensional volume of $\mathcal{P}(\mathbf{B})$. In symbols, this can be written as

$$\det(\mathcal{L}) \triangleq \sqrt{\det(\mathbf{B}^T\mathbf{B})}.$$

In the special case that $\mathcal{L}$ is a full-rank lattice, $\mathbf{B}$ is a square matrix, and we have $\det(\mathcal{L}) = |\det(\mathbf{B})|$.

Although the determinant of a lattice is defined through a basis of it, it is an invariant of the lattice. It is not dependent on the lattice base.

If we have a lattice $\mathcal{L}$ and two equivalent bases $\mathbf{B}_1, \mathbf{B}_2$ generating it, i.e., $\mathcal{L} = \mathcal{L}(\mathbf{B}_1) = \mathcal{L}(\mathbf{B}_2)$, then by Theorem 2.46 we have that there exists some unimodular matrix $\mathbf{U}$ such that $\mathbf{B}_2 = \mathbf{B}_1\mathbf{U}$, hence

$$\sqrt{\det(\mathbf{B}_2^T\mathbf{B}_2)} = \sqrt{\det(\mathbf{U}^T\mathbf{B}_1^T\mathbf{B}_1\mathbf{U})} = \sqrt{\det(\mathbf{B}_1^T\mathbf{B}_1)}$$

The determinant of a lattice is a metric of its sparsity, or equivalently, the inverse of the determinant of a lattice is a metric of its density.

Every lattice is intrinsically connected to another lattice called its dual. Dual lattices are a very useful tool, as certain concepts or problems are better interpreted through the dual lattice.

**Definition 2.49** (Dual lattice)**.** For a full-rank lattice $\mathcal{L}$ in $\mathbb{R}^m$, we define its dual lattice (sometimes known as the reciprocal lattice)

$$\mathcal{L}^* \triangleq \{\mathbf{y} \in \mathbb{R}^m \mid \forall \mathbf{x} \in \mathcal{L}, \langle \mathbf{x},\ \mathbf{y} \rangle \in \mathbb{Z}\}.$$

The above definition could be more general, involving non-full-rank lattices, if $\mathbf{y} \in \mathbb{R}^m$ was changed to $\mathbf{y} \in \text{span}(\mathcal{L})$.

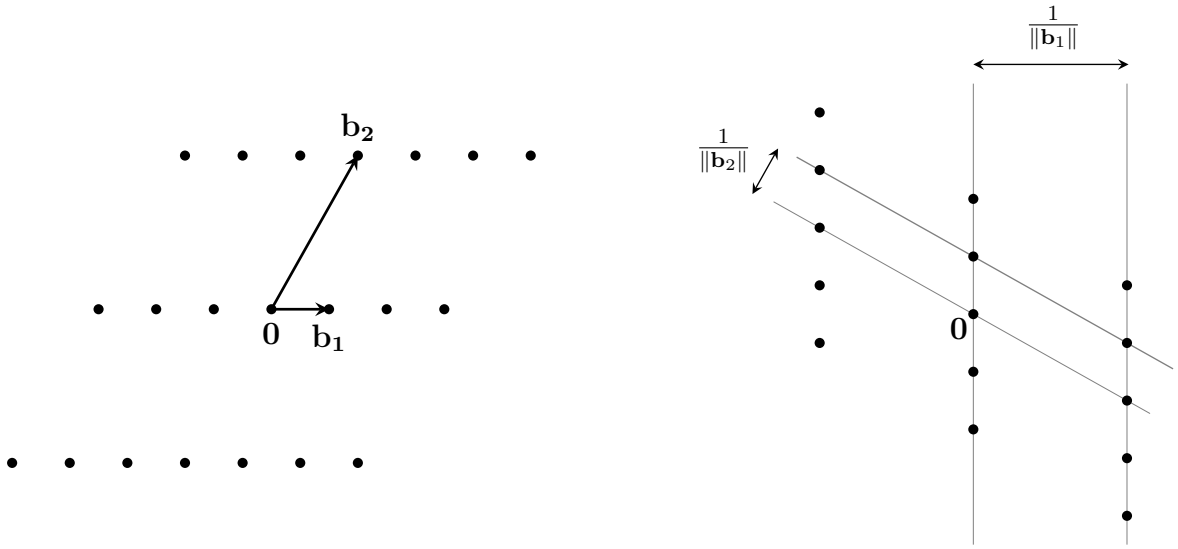
Based on the above definition, we can derive the following geometric interpretation of the dual lattice. For any vector $\mathbf{x}$, the set of points whose inner product with $\mathbf{x}$ is an integer, forms a series of hyperplanes, each perpendicular to $\mathbf{x}$ and spaced by a distance of $1/\|\mathbf{x}\|$. Therefore, any vector $\mathbf{x}$ in the lattice $\mathcal{L}$ imposes a condition that all points in the dual lattice $\mathcal{L}^*$ must lie on one of the hyperplanes defined by $\mathbf{x}$. See Figure 5 for an illustration.

**Definition 2.50** (Dual basis)**.** For a basis $\mathbf{B} = \begin{bmatrix} \mathbf{b}_1 & \cdots & \mathbf{b}_n \end{bmatrix} \in \mathbb{R}^{m \times n}$, define the dual basis $\mathbf{D} = \begin{bmatrix} \mathbf{d}_1 & \cdots & \mathbf{d}_n \end{bmatrix} \in \mathbb{R}^{m \times n}$ as the unique basis that satisfies

- $\text{span}(\mathbf{B}) = \text{span}(\mathbf{D})$
- $\mathbf{B}^T\mathbf{D} = \mathbf{I}$

The condition $\mathbf{B}^T\mathbf{D} = \mathbf{I}$ can be interpreted in terms of inner product as $\langle \mathbf{b}_i, \mathbf{d}_j \rangle = \delta_{i,j}$, where $\delta_{i,j}$ is the Kronecker delta function, i.e.,

$$\delta_{i,j} = \begin{cases} 1 & \text{if } \ i = j \\ 0 & \text{otherwise.} \end{cases}$$




(a) $\mathcal{L}$ (b) $\mathcal{L}^*$

Figure 5: A lattice and its dual.

In the case where $\mathbf{B}$ is full-rank, then $\mathbf{D} = \mathbf{B}^{-T}$. In the general case, $\mathbf{D} = \mathbf{B}\left(\mathbf{B}^T\mathbf{B}\right)^{-1}$

**Theorem 2.51.** *Let* $\mathbf{D}$ *be the dual basis of a basis* $\mathbf{B}$*. If* $\mathcal{L} = \mathcal{L}(\mathbf{B})$*, then* $\mathcal{L}^* = \mathcal{L}(\mathbf{D})$*.*

**Theorem 2.52.** *For any lattice* $\mathcal{L}$ *in* $\mathbb{R}^m$*,* $(\mathcal{L}^*)^* = \mathcal{L}$*.*

**Theorem 2.53.** *For any lattice* $\mathcal{L}$ *in* $\mathbb{R}^m$*,* $\det(\mathcal{L}^*) = 1/\det(\mathcal{L})$*.*

One basic parameter of a lattice is the length of its shortest nonzero vector.[8] This parameter is denoted by $\lambda_1$.

**Definition 2.54** (Minimum distance of a lattice)**.** The minimum distance of a lattice $\mathcal{L}$ is the length of a shortest nonzero lattice vector:

$$\lambda_1(\mathcal{L}) \triangleq \min_{\mathbf{v} \in \mathcal{L} \setminus \mathbf{0}} \|\mathbf{v}\|.$$

$\lambda_i(\mathcal{L})$ is the smallest $r$ such that $\mathcal{L}$ has $i$ linearly independent vectors of norm at most $r$.

[8] We specify a nonzero vector; otherwise, the zero vector would be a trivial solution.

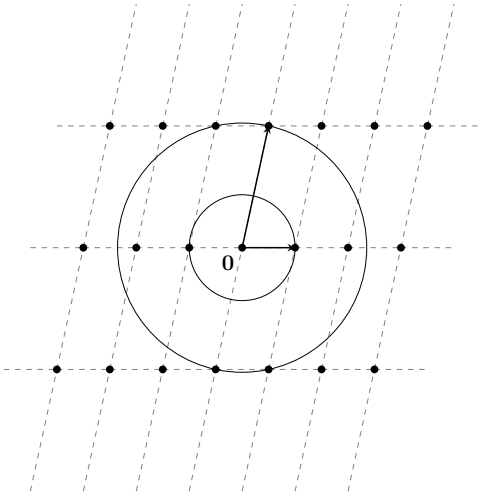



Figure 6: Lattice $\lambda_i$.

**Definition 2.55** ($i$-th successive minimum of a lattice)**.** Let $\mathcal{L}$ be a lattice of rank $n$. For $i \in \{1, \ldots, n\}$ we define the $i$-th successive minimum as

$$\lambda_i(\mathcal{L}) \triangleq \inf\{r \in \mathbb{R} \mid \dim(\mathrm{span}(\mathcal{L} \cap \overline{\mathcal{B}}(\mathbf{0}, r))) \geq i\}.$$

See Figure 6 for an illustration.

For a full-rank lattice $\mathcal{L}$, there is an upper bound on the length of the shortest nonzero vector of $\mathcal{L}$, which is an invariant of $\mathcal{L}$.

**Theorem 2.56** (Minkowski's first theorem)**.** *For any full-rank lattice $\mathcal{L}$ of rank $n$,*

$$\lambda_1(\mathcal{L}) \leq \sqrt{n}(\det(\mathcal{L}))^{1/n}.$$

Minkowski's first theorem provides a bound on $\lambda_1(\mathcal{L})$. Minkowski's second theorem strengthens that bound by considering the geometric mean of all $\lambda_i(\mathcal{L})$, instead of just $\lambda_1(\mathcal{L})$, which is clearly at least $\lambda_1(\mathcal{L})$.

**Theorem 2.57** (Minkowski's second theorem)**.** *For any full-rank lattice $\mathcal{L}$ of rank $n$,*

$$\left(\prod_{i=1}^{n} \lambda_i(\mathcal{L})\right)^{1/n} \leq \sqrt{n}(\det(\mathcal{L}))^{1/n}.$$

A last notion to mention in lattice theory that is used in the LWE problem that is described in a following chapter, is the concept of probabilistic noise over a lattice.

**Definition 2.58** (Gaussian function)**.** For any positive integer $n$ and real $s > 0$, which is taken to be $s = 1$ when omitted, define the Gaussian function $\rho_s : \mathbb{R}^n \to \mathbb{R}^{\geq 0}$ of parameter (or width) $s$ as

$$\rho_s(\mathbf{x}) \triangleq e^{-\pi \|\mathbf{x}\|^2 / s^2} = \rho(\mathbf{x}/s).$$

The normalization factor $\pi$ in the exponent is selected so that $\rho$ is its own Fourier transform.

Observe that $\rho_s$ is invariant under rotations of $\mathbb{R}^n$.

Furthermore, $\rho_s(\mathbf{x}) = \prod_{i=1}^n \rho_s(x_i)$.

The continuous Gaussian distribution $D_s$ over $\mathbb{R}^n$ is defined to have a probability density function $\rho_s$, but normalized, i.e., proportional to $\rho_s$ such that the probability of the entire sample space (in this case $\mathbb{R}^n$) is equal to 1:

$$f(\mathbf{x}) \triangleq \frac{\rho_s(\mathbf{x})}{\int_{\mathbb{R}^n} \rho_s(\mathbf{y})\, d\mathbf{y}} = \rho_s(\mathbf{x})/s^n.$$

Since a lattice $\mathcal{L}$ is an additive subgroup of $\mathbb{R}^n$, the quotient group $\mathbb{R}^n/\mathcal{L}$ of cosets is well-defined:

$$\mathbf{c} + \mathcal{L} = \{\mathbf{c} + \mathbf{v} : \mathbf{v} \in \mathcal{L}\}, \mathbf{c} \in \mathbb{R}^n,$$

with the usual induced addition operation $(\mathbf{c}_1 + \mathcal{L}) + (\mathbf{c}_2 + \mathcal{L}) = (\mathbf{c}_1 + \mathbf{c}_2) + \mathcal{L}$.

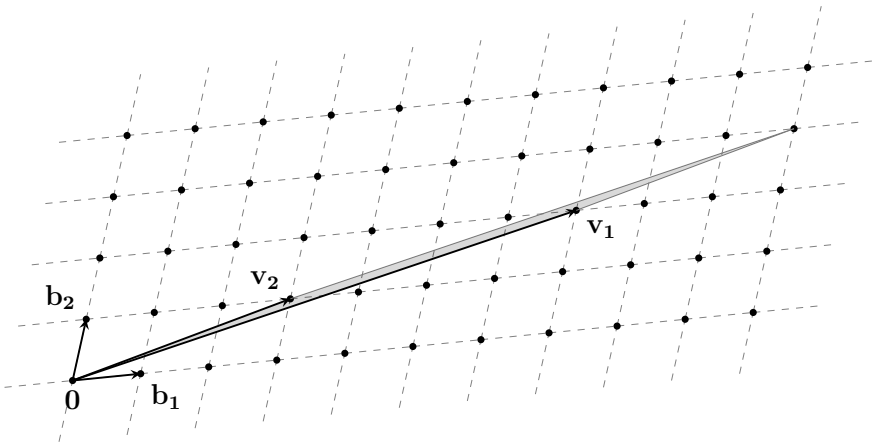
**Definition 2.59** (Discrete Gaussian distribution)**.** For a lattice coset $\mathbf{c} + \mathcal{L} \subset \mathbb{R}^n$ and parameter $s > 0$, the discrete Gaussian probability distribution $D_{\mathbf{c}+\mathcal{L},s}$ is simply the Gaussian distribution restricted to the coset:

$$D_{\mathbf{c}+\mathcal{L},s}(\mathbf{x}) \propto \begin{cases} \rho_s(\mathbf{x}) & \text{if } \mathbf{x} \in \mathbf{c} + \mathcal{L} \\ 0 & \text{otherwise} \end{cases}$$

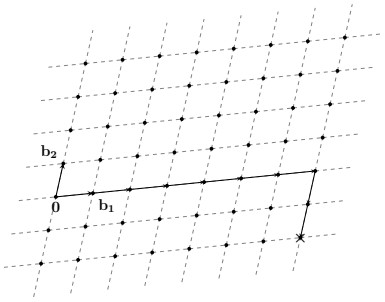
Of course, there is much more theory behind discrete Gaussians than what has been mentioned above.

### 2.3.2 Computational Problems

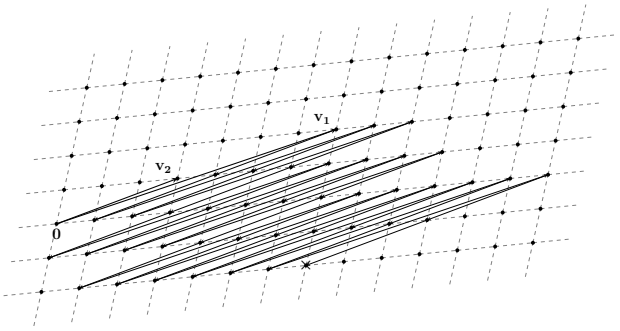
At this point, we introduce some computational problems on lattices. The most basic computational problem involving lattices is the Shortest Vector Problem, or SVP for




(a) A "good" and a "bad" basis for a lattice. The fundamental parallelepiped of the "bad" basis is also depicted to show that $\{\mathbf{v}_1, \mathbf{v}_2\}$ is indeed a basis of the lattice.




(b) A solution using the "good" basis.




(c) A solution using the "bad" basis.

Figure 7: "Good" vs "bad" lattice basis.

short. Minkowski's first theorem provides an upper bound for $\lambda_1(\mathcal{L})$, however, actually, it may be hard to find it if the given basis of $\mathcal{L}$ is a "bad" basis. There is no known efficient algorithm for finding such short vectors. A "bad" lattice basis can be thought as a basis that is far away from being orthogonal, e.g., long vectors "barely keeping the rank" of the lattice. See Figure 7 for an illustration of how it becomes harder to solve a system given a lattice point using the "bad" basis.

There are three variants of the SVP, in which a lattice basis is given as input:

- Search SVP: Find the shortest nonzero vector.
- Optimization SVP: Find the shortest nonzero vector length.
- Decisional SVP: Infer if the shortest nonzero vector is shorter than a given length.

**Definition 2.60** (Search SVP)**.** Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$, find $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $\|\mathbf{v}\| = \lambda_1(\mathcal{L}(\mathbf{B}))$.

**Definition 2.61** (Optimization SVP)**.** Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$, find $\lambda_1(\mathcal{L}(\mathbf{B}))$.

**Definition 2.62** (Decisional SVP)**.** Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$ and a rational $r \in \mathbb{Q}$, determine whether $\lambda_1(\mathcal{L}(\mathbf{B})) \leq r$ or not.

The lattice bases in the above problems are restricted to integer vectors to ensure the input can be represented in finitely many bits, allowing SVP to be treated as a standard computational problem. Allowing rational basis vectors instead of integer ones would result in an essentially equivalent definition, as scaling can convert all rational coordinates to integers.

Two easy relations among the three variants above are:

- Decision SVP is no harder than Optimization SVP.
- Optimization SVP is no harder than Search SVP.

In fact, it can be shown that the converse is also true:

- Optimization SVP is no harder than Decision SVP.
- Search SVP is no harder than Optimization SVP.

In summary, the three variants are essentially equivalent.

Additionally, there are the *approximation* variants of SVP which keep the basic concept, but, also, parameterized by some approximation factor parameter $\gamma \geq 1$; instead of finding the shortest vector, these approximation variants find a $\gamma$-approximation of it

**Definition 2.63** (Search $\mathsf{SVP}_\gamma$)**.** Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$, find $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $\mathbf{v} \neq 0$ and $\|\mathbf{v}\| \leq \gamma \cdot \lambda_1(\mathcal{L}(\mathbf{B}))$.

**Definition 2.64** (Optimization $\mathsf{SVP}_\gamma$)**.** Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$, find $d$ such that $d \leq \lambda_1(\mathcal{L}(\mathbf{B})) \leq \gamma d$.

**Definition 2.65** (Promise $\mathsf{SVP}_\gamma$)**.** An instance of the problem is given by a pair $(\mathbf{B}, r)$ where $\mathbf{B} \in \mathbb{Z}^{m \times n}$ is a lattice basis and $r \in \mathbb{Q}$. Infer which is the case:

- Case 1: $\lambda_1(\mathcal{L}(\mathbf{B})) \leq r$
- Case 2: $\lambda_1(\mathcal{L}(\mathbf{B})) > \gamma r$

In bibliography, the latter variant of the approximate-SVP, Promise $\mathsf{SVP}_\gamma$, is usually called GapSVP and denoted as $\mathsf{GapSVP}_\gamma$.

It is a promise problem, meaning that it is defined by two disjoint sets of inputs (in the above definition those are "Case 1" and "Case 2"). Unlike decision problems, the union of these sets does not have to contain all possible inputs, namely, there are illegal inputs on which the algorithm's behavior is undefined. The term "Promise" derives from the fact that it is promised the input not to be an illegal one.

As before, we have that for any $\gamma \geq 1$:

- Decision $\mathsf{SVP}_\gamma$ is no harder than Optimization $\mathsf{SVP}_\gamma$.
- Optimization $\mathsf{SVP}_\gamma$ is no harder than Search $\mathsf{SVP}_\gamma$.
- Optimization $\mathsf{SVP}_\gamma$ is no harder than Decision $\mathsf{SVP}_\gamma$.
- (Open question) Is Search $\mathsf{SVP}_\gamma$ no harder than Optimization $\mathsf{SVP}_\gamma$?

Decision $\mathsf{SVP}_\gamma$ and Optimization $\mathsf{SVP}_\gamma$ are equivalent.

One more problem similar to the approximate SVP is the approximate SIVP (Shortest Independent Vectors Problem). The difference is that the approximate SIVP asks for $n$ linearly independent lattice vectors, where the norm of each one is bounded by $\gamma \cdot \lambda_n(\mathcal{L})$.

**Definition 2.66** (Search $\mathsf{SIVP}_\gamma$)**.** Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$, find linearly independent $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathcal{L}(\mathbf{B})$ such that $\|\mathbf{v}_i\| \leq \gamma \cdot \lambda_n(\mathcal{L}(\mathbf{B}))$ for all $i \in \{1, \ldots, n\}$.

Another fundamental lattice problem is the Closest Vector Problem ($\mathsf{CVP}$). The goal is to find the lattice point closest to a given point in space. As with the previous problems, for any approximation factor $\gamma \geq 1$, we can define three variants.

**Definition 2.67** (Search $\mathsf{CVP}_\gamma$)**.** Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$ and a vector $\mathbf{t} \in Z^m$, find $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $\|\mathbf{v} - \mathbf{t}\| \leq \gamma \cdot \mathrm{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B}))$.

**Definition 2.68** (Optimization $\mathsf{CVP}_\gamma$)**.** Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$ and a vector $\mathbf{t} \in \mathbb{Z}^m$, find $d$ such that $d \leq \mathrm{dist}(\mathbf{t}, \mathbf{L}(\mathbf{B})) \leq \gamma d$.

**Definition 2.69** (Promise $\mathsf{CVP}_\gamma$)**.** An instance of the problem is given by a triple $(\mathbf{B}, \mathbf{t}, r)$ where $\mathbf{B} \in \mathbb{Z}^{m \times n}$ is a lattice basis, $\mathbf{t} \in \mathbb{Z}^m$, and $r \in \mathbb{Q}$. Infer which is the case:

- $\text{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B})) \leq r$
- $\text{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B})) > \gamma r$

SVP, SIVP and CVP are considered hard computational problems on lattices.

In the case of Search $\mathsf{CVP}_\gamma$, if the result vector is promised to be unique, the problem is named $\mathsf{BDD}_\gamma$ (Bounded Distance Decoding).

When referring to one of the aforementioned lattice problems without specifying the variant (search, optimization, decisional, promise), the search variant is implied.

# 3 The Concept of Homomorphic Encryption

Homomorphic encryption is a special cryptographic technique that enables computations to be performed on encrypted data without the need to decrypt it first. This property is especially helpful in a case where several privacy-preserving computations of data are to be done and one does not want to provide data in plaintext. Examples include cloud computing and secure multi-party computation. In these cases, homomorphic encryption enables operations to be performed on the encrypted data, hence avoiding exposure to sensitive information during computation and securing data privacy.

The technique is based on the mathematical (more specifically, algebraic) structure of various encryption schemes, whereby certain operations on ciphertext correspond to meaningful operations on the underlying plaintext. In this regard, homomorphic encryption can be either partially homomorphic, fully homomorphic, or somewhere in-between, that is, systems that can allow either addition or multiplication over the encrypted data to systems that support arbitrary operations.

The idea of homomorphic encryption, conceptually-speaking, goes back to the 1970s. However, it turned out that in practice it faced severe challenges due to inherent computational inefficiency. However, recent breakthroughs made fully homomorphic encryption effective; hence, applications such as privacy-preserving machine learning, secure data outsourcing, and encrypted databases opened up. In this chapter, we will discuss the basics of homomorphic encryption and the types of schemes.

At a high level, a homomorphic encryption scheme contains one more algorithm in its algorithm family $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$; this is the $\mathsf{Eval}$ algorithm (which may be randomized). $\mathsf{Eval}$ takes as input an extra kind of key (denoted in subscript) named the evaluation key (which can be shared publically), a list of ciphertexts (all encrypted under the same encryption key, public or secret key, depending on the kind of the cryptosystem) and the kind of evaluation that is going to be performed. Finally, it returns a ciphertext encrypting the result of this evaluation under the same key with the input ciphertexts.

Mathematically speaking, let $p_1, \ldots, p_n$ be some plaintexts, and let $c_1, \ldots, c_n$ be some ciphertexts encrypting the aforementioned plaintexts, respectively, under the same encryption key. More specifically, let (for a public-key cryptosystem) $\mathsf{pk}, \mathsf{sk}, \mathsf{ek}$ be the encryption key, the decryption key and the evaluation key, respectively. For the secret-key cryptosystem scenario, $\mathsf{pk} = \mathsf{sk}$ would be the case. So,

$$c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(p_i), \quad \text{for } i = 1, \ldots, n.$$

Let us say that one desires to compute $f(p_1, \ldots, p_n)$ for any desired function $f$ as long as $f$ can be efficiently computed. What the $\mathsf{Eval}$ algorithm does is that it takes as input this function $f$ and the ciphertexts encrypting the input plaintexts of $f$ and it returns a ciphertext that upon decryption will be equal to $f(p_1, \ldots, p_n)$, i.e.,

$$c \leftarrow \mathsf{Eval}_{\mathsf{ek}}(f, c_1, \ldots, c_n) \quad \text{such that} \quad f(p_1, \ldots, p_n) = \mathsf{Dec}_{\mathsf{sk}}(c).$$

One simple example of a homomorphic cryptosystem is RSA with multiplication. Let two plaintexts $p_1, p_2$ and their encryptions $c_1 = r_N[p_1^e], c_2 = r_N[p_2^e]$. Then,

$$r_N[c_1 c_2] = r_N[(p_1 p_2)^e].$$

Of course, it is quite easy to understand that a homomorphic cryptosystem is inherently not CCA-secure. The ability to perform homomorphic operations on ciphertexts gives to attackers a way to bypass the restriction of the CCA experiment that forbids the decryption of the challenge ciphertext.

The homomorphic encryption schemes can be categorized based on the capability of the encrypted operations that these can perform. Some basic categories are:

- **Partially Homomorphic Encryption**: Schemes that support the evaluation of circuits consisting of only one type of gate, e.g., addition or multiplication.
- **Somewhat Homomorphic Encryption**: Schemes that support the evaluation of circuits consisting of two types of gates, but only for a subset of circuits.
- **Leveled Fully Homomorphic Encryption**: Schemes that support the evaluation of arbitrary circuits composed of multiple types of gates of bounded (predetermined) depth.
- **Fully Homomorphic Encryption**: Schemes that support the evaluation of arbitrary circuits composed of multiple types of gates of unbounded depth.

The strongest notion of homomorphic encryption is the last category, Fully Homomorphic Encryption (FHE), and the first FHE scheme was introduced by Craig Gentry in his PhD thesis in 2009 [4].

Gentry introduced a technique called *bootstrapping*. Consider the following simple example in order to understand it. Let a homomorphic encryption scheme that can perform both encrypted addition and multiplication operations, but, the depth of those operations is finite (and known), which means that it may not be able to perform as many

consequent encrypted operations as the user may desire. Bootstrapping is a process that "resets" the depth consumed by the encrypted operations already performed on a ciphertext; not only that, everything remains encrypted the whole time of the bootstrapping procedure, the data encrypted in the ciphertext is never exposed.

Assume that the bootstrapping as a procedure needs depth $n$. Then, if the cryptosystem supports depth $n + m$ (for $m \geq 1$), then the user can perform $m$ encrypted operations on this ciphertext, then they perform bootstrapping. Now, the user has again depth $n + m$ available. Bootstrapping turned a finite-depth homomorphic encryption scheme to a infinite-depth one. In a nutshell, bootstrapping "refreshes" the available encrypted operation depth of a ciphertext while that ciphertext is encrypted the whole time during that procedure.

At a high level, to understand how bootstrapping works, the first thing to understand is that it acts on top of an encryption scheme that can evaluate its own decryption circuit. Using the terms introduced above, this means that the $\mathsf{Eval}$ algorithm of this scheme must be able to perform in its input ciphertexts the function $f = \mathsf{Dec}$.

To be more specific, a high level outline of how the bootstrapping procedure works is presented here. Let a somewhat homomorphic encryption scheme that can evaluate its own decryption circuit. Let a plaintext $p$. Assume, for simplicity but without loss of generality, that $\mathsf{Enc}$ is able of encrypting a ciphertext and a secret key; in the general case it may be needed that those are broken into bits and encoded somehow as plaintexts, so as they match the $\mathsf{Enc}$ specification.

- Generate keys: $(\mathsf{pk}, \mathsf{sk}, \mathsf{ek})$
- Encrypt $p$: $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(p)$
- Perform as many encrypted operations as permitted on $c$ using $\mathsf{Eval}$
- Generate new keys: $(\mathsf{pk}', \mathsf{sk}', \mathsf{ek}')$
- Add a second encryption layer to $c$ under the new keys: $c' \leftarrow \mathsf{Enc}_{\mathsf{pk}'}(c)$
- Encrypt the initial secret key under the new keys: $c_{\mathrm{sk}} \leftarrow \mathsf{Enc}_{\mathsf{pk}'}(\mathsf{sk})$
- Evaluate decryption on $c_{\mathrm{sk}}, c'$: $c'' \leftarrow \mathsf{Eval}_{\mathsf{ek}'}(\mathsf{Dec}, c_{\mathrm{sk}}, c')$
- $c''$ is an encryption of $p$ under the new keys

What $\mathsf{Eval}_{\mathsf{ek}'}(\mathsf{Dec}, c_{\mathrm{sk}}, c')$ do is that it removes the inner encryption layer, and this is possible thanks to the homomorphic encryption property of the scheme.

Let us verify the result:

$$\mathsf{Dec}_{\mathsf{sk}'}(c'') = \mathsf{Dec}_{\mathsf{sk}}(c) = p.$$

# 4 Lattice-Based Cryptography

This chapter is based on [5].

Lattice-based cryptography comprises all those varieties of cryptography based on computational problems that involve lattices in high-dimensional space. Basic problems are to find the most short vector (SVP) in a lattice and variants, including the Short Integer Solution (SIS) and the Learning With Errors (LWE) problem. These are thought to be resistant even against quantum attacks; therefore, such cryptosystems are considered a promising candidate for post-quantum cryptography. In this vein, several lattice-based schemes for a variety of cryptographic primitives have been suggested, among others, encryption schemes, digital signatures, and fully homomorphic encryption directly, by relying on worst-case hardness assumptions on lattices.

In this chapter, SIS and LWE are introduced, and it is shown how their hardness depends on the hardness of the more "primitive" lattice problems like approximate-SVP, approximate-SIVP. Being a slightly more specific, worst-case "primitive" lattice problems (like those mentioned above) are reducible to average-case SIS and LWE, (worst to average case reduction), which means that solving average-case SIS and LWE is at least as hard as solving worst-case "primitive" lattice problems, for appropriate parameters, of course.

After SIS and LWE, their ring variants are also introduced.

## 4.1 The Short Integer Solution (SIS) Problem

The Short Integer Solution (SIS) problem, initially introduced in Ajtai's influential work in 1996 [6], has formed the basis for one-way and collision-resistant hash functions, identification schemes, digital signatures, and other "minicrypt" primitives, though not for public-key encryption. Roughly, the reason for the last one is that Ajtai's function has "highly surjective" behavior, meaning that it generally "shrinks" the size of its input, i.e., the output size is smaller than the input size, implying non-invertibility due to possible collisions (pigeonhole principle).

Informally, the SIS problem can be described as follows: given many uniformly random elements of a certain large finite additive group, find a sufficiently "short" nontrivial integer combination of them that sums to zero.

**Definition 4.1** ($\mathsf{SIS}_{q,m,n,\beta}$)**.** Given $m$ uniformly random vectors $\mathbf{a}_i \in \mathbb{Z}_q^n$ , forming the columns of a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ , find a nonzero integer vector $\mathbf{x} \in \mathbb{Z}^m$ of norm $\|\mathbf{x}\| \leq \beta$

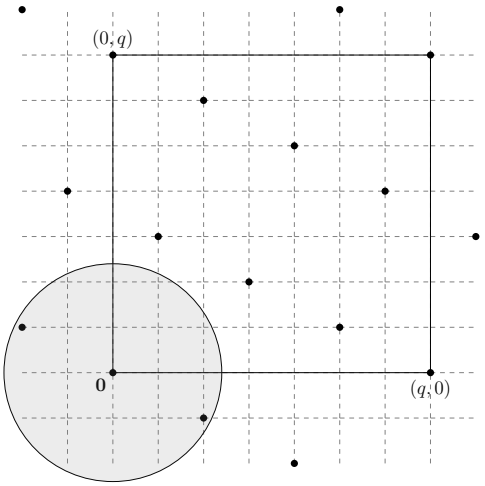



Figure 8: Average-case SIS problem viewed as an average-case SVP problem.

such that

$$f_{\mathbf{A}}(\mathbf{x}) \triangleq \mathbf{A}\mathbf{x} = \sum_{i=1}^{m} x_i \mathbf{a}_i = \mathbf{0} \in \mathbb{Z}_q^n,$$

where the operations are done modulo $q$.

Here, we emphasize a few simple yet valuable observations regarding the SIS problem:

- Without the constraint on $\|\mathbf{x}\|$, it is easy to find a solution via Gaussian elimination. Similarly, one must take $\beta < q$ because, otherwise, $\mathbf{x} = (q, 0, \ldots, 0) \in \mathbb{Z}^m$ would always be a legitimate (but trivial) solution.
- The norm bound $\beta$ and the number $m$ of vectors $\mathbf{a}_i$ must be large enough that a solution is guaranteed to exist. This is the case whenever $\beta \geq \sqrt{\lceil n \lg q \rceil}$ and $m \geq \lceil n \lg q \rceil$. Hence, a good setup for the problem would be $\sqrt{\lceil n \lg q \rceil} \leq \beta \ll q$.
- Any solution for a matrix $\mathbf{A}$ can trivially be converted to one for any extension $\begin{bmatrix}\mathbf{A} & \mathbf{A}'\end{bmatrix}$, simply by appending the solution vector with zeros, leaving the norm of the solution as is. In other words, one can ignore columns $\mathbf{a}_i$ as desired, so, the SIS problem *can only become easier as $m$ increases*. Similarly, it *can only become harder as $n$ increases*.

The SIS problem can be viewed as an average-case SVP problem on a specific class of $m$-dimensional integer lattices, called "$q$-ary" lattices, specifically the lattices

$$\mathcal{L}^{\perp}(\mathbf{A}) \triangleq \{\mathbf{x} \in \mathbb{Z}^m \mid \mathbf{A}\mathbf{x} \equiv \mathbf{0} \pmod{q}\} \supseteq q\mathbb{Z}^m. \tag{4.1}$$

See Figure 8 for an illustration.

An essentially equivalent (for typical parameters) variant of SIS, is the inhomogeneous version of it. Inhomogeneous SIS is almost the same, with the only difference being that the short integer solutions refer to the equation $\mathbf{Ax} = \mathbf{u} \in \mathbb{Z}_q^n$, where $\mathbf{A}, \mathbf{u}$ are uniformly distributed and independent.

The SIS problem allows for a small but significant optimization, known as the *Hermite normal form* (HNF). This technique reduces the size of the instance $\mathbf{A}$ by $n$ columns, without sacrificing any cryptographic functionality or hardness. Without loss of generality, we can assume that the leftmost $n$ columns $\mathbf{A}_1 \in \mathbb{Z}_q^{n \times n}$ of $\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 \end{bmatrix} \in \mathbb{Z}_q^{n \times m}$ form an invertible matrix over $\mathbb{Z}_q$. The reason for that is (as it has been observed above), one can ignore columns as desired. Then, replace $\mathbf{A}$ with

$$\mathbf{A}_1^{-1}\mathbf{A} = \begin{bmatrix} \mathbf{I}_n & \bar{\mathbf{A}} \end{bmatrix}, \quad \text{where} \quad \bar{\mathbf{A}} \triangleq \mathbf{A}_1^{-1}\mathbf{A}_2$$

and treat the identity $n \times n$ submatrix as implicit. Observe that $\bar{\mathbf{A}}$ is uniformly random, because $\mathbf{A}_2$ is uniform and independent of $\mathbf{A}_1$ . In addition to that, $\mathbf{A}$ and $[\mathbf{I}_n \mid \bar{\mathbf{A}}]$ have exactly the same set of (short) SIS solutions. Hence, SIS instances of the HNF form are at least as hard to solve as those of the initial form.

On the hardness of the SIS problem, the following theorem, from Ajtai's work in 1996 containing strengthening hardness improvements from a sequence of works that have followed, formulates that worst-case GapSVP and approximate-SIVP are no harder than average-case SIS.

**Theorem 4.2.** *For any $m = \text{poly}(n)$, any $\beta > 0$, and any sufficiently large $q \geq \beta n^{\varepsilon}$ for any constant $\varepsilon > 0$, solving $\mathsf{SIS}_{q,m,n,\beta}$ with non-negligible probability is at least as hard as solving $\mathsf{GapSVP}_{\gamma}$ and $\mathsf{SIVP}_{\gamma}$ (among others) on arbitrary $n$-dimensional lattices (i.e., in the worst case) with overwhelming probability (i.e., negligibly less than 1), for some $\gamma = \beta \cdot \tilde{O}(\sqrt{n})$.*

## 4.2 The Learning With Errors (LWE) Problem

The Learning With Errors (LWE) problem, initially introduced in Regev's influential work in 2005 [7], is the "encryption enabling" analogue of the SIS problem. In contrast to Ajtai's function, Regev's function is "highly injective", hence, increasing the size of its input and avoiding collisions.

Before we provide the definition for the LWE problem, we need to define the LWE distribution first.

**Definition 4.3** (LWE distribution)**.** For a vector $\mathbf{s} \in \mathbb{Z}_q^n$ called the secret, the LWE distribution $A_{\mathbf{s},\chi}$ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is sampled by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \leftarrow \chi$, and outputting $(\mathbf{a}, b = r_q[\langle \mathbf{a}, \mathbf{s} \rangle + e])$.

There are two important variants of the LWE problem: a search version, namely, to find the secret from the LWE samples, and a decision version that needs to differentiate the LWE samples from the uniformly random ones. These problems are parameterized by the parameters $q, m, n$ (same to the corresponding ones in SIS), and by an error distribution $\chi$ over $\mathbb{Z}$, usually taken to be a discrete Gaussian.

**Definition 4.4** (Search $\mathsf{LWE}_{q,m,n,\chi}$)**.** Given $m$ independent samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ drawn from $A_{\mathbf{s},\chi}$ for a uniformly random $\mathbf{s} \in \mathbb{Z}_q^n$ (fixed for all samples), find $\mathbf{s}$.

**Definition 4.5** (Decision $\mathsf{LWE}_{q,m,n,\chi}$)**.** Given $m$ independent samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ where every sample is distributed according to either:

- $A_{\mathbf{s},\chi}$ for a uniformly random $\mathbf{s} \in \mathbb{Z}_q^n$ (fixed for all samples), or
- the uniform distribution,

distinguish which is the case (with non-negligible advantage).

Here, we emphasize a few simple yet valuable observations regarding both the LWE problem variants:

- Without the error terms from $\chi$, both problems become simple to solve, as we can efficiently recover $\mathbf{s}$ from the LWE samples using Gaussian elimination. In the case of uniform decision-LWE, no solution $\mathbf{s}$ will likely exist with high probability.
- Combining the LWE samples into a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ (whose columns are the vectors $\mathbf{a}_i \in \mathbb{Z}_q^n$ ) and a vector $\mathbf{b} \in \mathbb{Z}_q^m$ (whose entries are the $\mathbf{b}_i \in \mathbb{Z}_q$), and the corresponding error terms into a vector $\mathbf{e}$, we get that

$$\mathbf{b} = \mathbf{A}^T\mathbf{s} + \mathbf{e} \in \mathbb{Z}_q^m \quad \text{(operations modulo } q\text{)}.$$

As in the case with SIS, Search $\mathsf{LWE}$ can be viewed as an average-case BDD problem on the family of $q$-ary $m$-dimensional integer lattices: for LWE samples, the vector $\mathbf{b}$ lies relatively close to a single vector in the LWE lattice

$$\mathcal{L}(\mathbf{A}) \triangleq \left\{ \mathbf{A}^T\mathbf{x} \;\middle|\; \mathbf{x} \in \mathbb{Z}_q^n \right\} + q\mathbb{Z}^m, \tag{4.2}$$

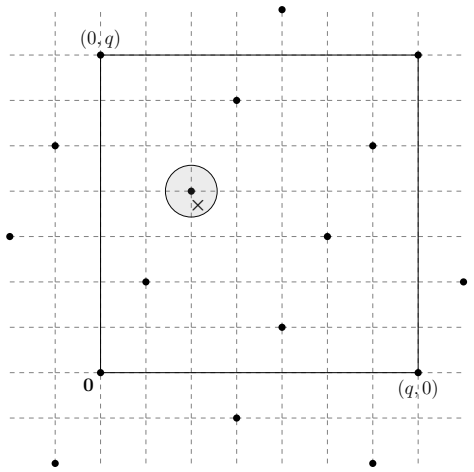



Figure 9: Average-case LWE problem viewed as an average-case BDD problem.

and the goal is to find that lattice vector. In the uniform case, $\mathbf{b}$ is far from all points in $\mathcal{L}(\mathbf{A})$ with very high probability.

See Figure 9 for an illustration.

It is easy to verify that the SIS and LWE problems as defined through $q$-ary lattices in equations (4.1) and (4.2) are dual to each other, up to a scaling factor of $q$, meaning that $\mathcal{L}(\mathcal{A}) = q \cdot \mathcal{L}^{\perp}(\mathbf{A})^*$.

Like the SIS problem, the LWE problem has a normal form, too, where the coordinates of the normal form secret are independently selected from the error distribution $\chi$ (modulo $q$).

For $\mathbf{A} = \begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2\end{bmatrix} \in \mathbb{Z}_q^{n \times m}$ with $\mathbf{A}_1 \in \mathbb{Z}_q^{n \times n}$ being invertible, we have

$$-\mathbf{A}_1^{-1}\mathbf{A} = \begin{bmatrix}-\mathbf{I}_n & \bar{\mathbf{A}}\end{bmatrix}, \quad \text{where} \quad \bar{\mathbf{A}} \triangleq -\mathbf{A}_1^{-1}\mathbf{A}_2.$$

As in SIS normal form, $\bar{\mathbf{A}}$ is uniformly random since $\mathbf{A}_2$ is independent of $\mathbf{A}_1$.

For $\mathbf{b} = \begin{bmatrix}\mathbf{b}_1 \\ \mathbf{b}_2\end{bmatrix} \in \mathbb{Z}_q^m$ with $\mathbf{b}_1 \in \mathbb{Z}_q^n$ (observe that $\mathbf{b}_i = \mathbf{A}_i^T\mathbf{s} + \mathbf{e}_i$ for $i = 1, 2$) we have

$$\begin{aligned}\bar{\mathbf{b}} &\triangleq \bar{\mathbf{A}}^T\mathbf{b}_1 + \mathbf{b}_2 \\ &= \bar{\mathbf{A}}^T\left(\mathbf{A}_1^T\mathbf{s} + \mathbf{e}_1\right) + \mathbf{A}_2^T\mathbf{s} + \mathbf{e}_2 \\ &= \bar{\mathbf{A}}^T\mathbf{e}_1 + \mathbf{e}_2.\end{aligned}$$

That is, the normal form LWE instance $(\bar{\mathbf{A}}, \bar{\mathbf{b}})$ comes from an LWE distribution, with secret $\mathbf{e}_1$, which is distributed according to the distribution $\chi$. Finding $\mathbf{e}_1$ yields the

original secret $\mathbf{s}$ through

$$\mathbf{s} = \mathbf{A}_1^{-T}(\mathbf{b}_1 - \mathbf{e}_1).$$

Moreover, for the uniform distribution case of the decision problem, when $\mathbf{b}$ is uniformly random and independent of $\mathbf{A}$, it immediately follows that $\bar{\mathbf{b}}$ is uniformly random and independent of $\bar{\mathbf{A}}$.

On the hardness of the LWE problem, the following theorem, from Regev's work in 2005 (stated strengthened from following works on it), (similarly to SIS, but using a quantum reduction) formulates that worst-case $\mathsf{GapSVP}_\gamma$ and $\mathsf{SIVP}_\gamma$ are no harder than average-case LWE.

**Theorem 4.6.** *For any $m = \mathrm{poly}(n)$, any modulus $q \leq 2^{\mathrm{poly}(n)}$ , and any (discretized) Gaussian error distribution $\chi$ of parameter $\alpha q \geq 2\sqrt{n}$, where $0 < \alpha < 1$, solving the decision $\mathsf{LWE}_{q,m,n,\chi}$ problem is at least as hard as quantumly solving $\mathsf{GapSVP}_\gamma$ and $\mathsf{SIVP}_\gamma$ on arbitrary n-dimensional lattices (i.e., in the worst case), for some $\gamma = \tilde{O}(n/\alpha)$.*

This theorem provides a quantum polynomial-time reduction for worst-case GapSVP and approximate-SIVP to average-case LWE, effectively turning any LWE-solving algorithm (classical or quantum) into a quantum algorithm for lattice problems. The quantum aspect or the reduction is relevant because, except from general quantum speedups, there are no known quantum algorithms for $\mathsf{GapSVP}_\gamma$ or $\mathsf{SIVP}_\gamma$ that significantly improve classical ones.

In addition to the above theorem, for any modulus $q$, decision LWE has been shown to be equivalent to search LWE (with a polynomial increase in the number of samples $m$), through a classical reduction.

Furthermore, a classical reduction, further strengthening the hardness of LWE, was introduced by Peikert in 2009. In the classical reduction, the factor $\gamma$ is the same as in the above quantum-reduction theorem, but only GapSVP is involved, not SIVP. Nonetheless, the classical reduction requires an exponentially large modulus $q \geq 2^{n/2}$ for the LWE problem, whereas the quantum reduction works for any modulus $q \geq 2\sqrt{n}/\alpha$. A large modulus increases the number of bits needed to represent LWE samples, leading to larger key sizes and less efficient cryptosystems.

## 4.3 The Ring Short Integer Solution (RSIS) Problem

On this section and on the following one, the ring variants of the SIS and LWE problems are introduced. The ring variants are neither a special case or a generalization of the non-ring ones. These are different problems, though, each ring variant is syntactically very similar to its corresponding non-ring one. These two sections will be briefer and the above ones, skipping the hardness results.

Beginning with the RSIS problem, there are some subtle changes to the parameters compared to SIS. Those are:

- $R$ is a ring, which is usually taken to be a polynomial quotient ring of degree $n$. More specifically, $R = \mathbb{Z}[x]/f(x)$, where $f(x) \in \mathbb{Z}[x]$ could be, for example $f(x) = x^n - 1$, or $f(x) = x^n + 1$ for a power-of-two $n$.
- $R$ is also endowed with a norm $\|\cdot\|$ which, for a vector $\vec{x} \in R$, is defined as $\|\vec{x}\| = \sqrt{\sum_i \|x_i\|^2}$, where $\|x_i\|$ could (though not necessarily) be the $L^2$ norm of the $x_i$ polynomial coefficients. There are different variants here.
- For a positive integer modulus $q$, we define $R_q \triangleq R/qR = Z_q[x]/f(x)$.

**Definition 4.7** ($\mathsf{RSIS}_{q,R,m,\beta}$)**.** Given $m$ uniformly random elements $a_i \in R_q$, defining a vector $\vec{a} \in R_q^m$, find a nonzero vector $\vec{x} \in R^m$ of norm $\|\vec{x}\| \leq \beta$ such that

$$f_{\vec{a}}(\vec{x}) \triangleq \langle \vec{a}, \vec{x} \rangle = \sum_{i=1}^{m} a_i x_i = 0 \in R_q.$$

An important point to highlight is that the number $m$ of elements $a_i \in R_q$ required to guarantee the existence of a sufficiently short solution, in the case of RSIS is only $m \approx \lg q$, in contrast to $m \approx n \lg q$ for SIS.

On the hardness of the RSIS problem, things are somewhat similar to the SIS, but not entirely. There is in this case a worst to average case reduction to "primitive" lattice problems, but, on a family of lattices called *ideal lattices*.

## 4.4 The Ring Learning With Errors (RLWE) Problem

For the RLWE, the new parameters are similar to RSIS, as explained previously: $R = \mathbb{Z}[x]/f(x)$, where $f(x) \in \mathbb{Z}[x]$ is a polynomial of degree $n$, and $R_q = R/qR = \mathbb{Z}_q[x]/f(x)$.

**Definition 4.8** (RLWE distribution)**.** For an $s \in R_q$ called the "secret", the RLWE distribution $A_{s,\chi}$ over $R_q \times R_q$ is sampled by choosing $a \in R_q$ uniformly at random, choosing $e \leftarrow \chi$, and outputting $(a, b = sa + e \bmod q)$.

**Definition 4.9** (Search $\mathsf{RLWE}_{q,R,m,\chi}$)**.** Given $m$ independent samples $(a_i, b_i) \in R_q \times R_q$ drawn from $A_{s,\chi}$ for a uniformly random $s \in R_q$ (fixed for all samples), find $s$.

**Definition 4.10** (Decision $\mathsf{RLWE}_{q,R,m,\chi}$)**.** Given $m$ independent samples $(a_i, b_i) \in R_q \times R_q$ where every sample is distributed according to either:

- $A_{s,\chi}$ for a uniformly random $s \in R_q$ (fixed for all samples), or
- the uniform distribution,

distinguish which is the case (with non-negligible advantage).

On the hardness of the RLWE problem (as mentioned in the RSIS section above), there is in this case, too, a worst to average case reduction to "primitive" lattice problems, but, on ideal lattices.

## 4.5 Why are RSIS and RLWE Preferred in Cryptography?

Short answer: due to smaller key sizes and faster multiplication.

RSIS and RLWE do not have the matrix that SIS and LWE had. Hence, RSIS and RLWE keys are roughly the square root of keys in SIS and LWE, respectively.

Although, if each polynomial has $n$ coefficients, and we have many polynomials, then it is like the case with the matrix. However, as noted above, we do not have many polynomials; quite the opposite, actually.

In the multiplication case, in a matrix-vector multiplication, we have a complexity of $O(n^2)$. However, in the case of multiplying polynomials, FFT-like techniques can be leveraged reducing the complexity to quasi-linear, $\tilde{O}(n)$. For example, in RSIS, compute each $x_i a_i \in R_q$ in quasi-linear $\tilde{O}(n)$ time, so the total time to compute $f_{\vec{a}}(\vec{x})$ is also quasi-linear for typical choices of $q$ and $m$.

FFT-like techniques usually are a variant of FFT that work on integral elements. Those techniques use a DFT-variant called Number Theoretic Transform (NTT). NTT substitutes the DFT complex exponential with an element with similar properties on integer

modular arithmetic. To be more specific, that element is an $n$-th root of unity

$$\omega^n \equiv 1 \pmod{q}.$$

and NTT is

$$\mathrm{NTT}(\mathbf{x}) = \left[\sum_{i=0}^{n-1} x_i \omega^{ij} \bmod q\right]_{j=[n]}.$$

The similar properties of $n$-th roots of unity to complex roots of unity allows similar factorizations with DFT of the above formula, leading to an algorithm for NTT that run in quasi-linear time, $\tilde{O}(n)$.

# 5 A Simple LWE-Based Cryptosystem

In this chapter, we will introduce an example cryptosystem based on the LWE problem, in order to understand the basics (encryption, decryption, homomorphic operations). Since this is just a silly toy example just for demonstration, we will keep things as simple as possible, ignoring smart techniques and optimizations. So, it may seem that there is a huge overhead. Indeed, homomorphic encryption introduces space and time complexity, but not as big as in our example; there have been significant optimizations over time.

## 5.1 LWoE-Cryptosystem

It will be easier and notationally cleaner to understand how encryption-decryption and the homomorphic operations work if we start by ignoring the error in the LWE problem. After grasping the core ideas, we will introduce the error and see how does it affect the computations.

### 5.1.1 Definition

So, let us start by introducing the Learning Without Errors public-key cryptosystem $\mathsf{LWoE-PKCS}_{q,m,n,\beta}$.

The parameters of the scheme are:

- $q \in \mathbb{Z}^{\geq 2}$: A number to create the finite ring $\mathbb{Z}_q$.
- $m, n \in \mathbb{Z}^{\geq 1}$: The dimensions of a tall ($m \gg n$) matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$.
- $\beta \in \mathbb{Z}_q \setminus \{0\}$: A fixed number used in the encryption of the bit.

Having those parameters fixed, we are ready to define the scheme.

Note: The subsequent operations in $\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}$ correspond to those defined within the ring $\mathbb{Z}_q$, addition and multiplication modulo $q$.

**Definition 5.1** ($\mathsf{LWoE-PKCS}_{q,m,n,\beta}$)**.**

- $\mathsf{Gen} : \varnothing \to (\mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m) \times \mathbb{Z}_q^n$,

$$\mathsf{Gen}() \triangleq ((\mathbf{A}, \mathbf{b}), \mathbf{s}),$$

  where $\mathbf{A} \xleftarrow{\mathsf{R}} \mathbb{Z}_q^{m \times n}$, $\mathbf{s} \xleftarrow{\mathsf{R}} \mathbb{Z}_q^n$, $\mathbf{b} = \mathbf{A}\mathbf{s} \in \mathbb{Z}_q^m$.

- $\mathsf{Enc}_{\mathsf{pk}} : \mathbb{Z}_2 \to \mathbb{Z}_q^n \times \mathbb{Z}_q$, with $\mathsf{pk} = (\mathbf{A}, \mathbf{b})$,[9]

$$\mathsf{Enc}_{\mathbf{A},\mathbf{b}}(p) \triangleq \Big(\mathbf{A}^T\mathbf{w}\ ,\ \mathbf{b}^T\mathbf{w} + \beta p\Big),$$

  where $\mathbf{w} \xleftarrow{\mathsf{R}} \mathbb{Z}_2^m$.

- $\mathsf{Dec}_{\mathsf{sk}} : \mathbb{Z}_q^n \times \mathbb{Z}_q \to \mathbb{Z}_2$,

$$\mathsf{Dec}_{\mathbf{s}}(\mathbf{c}, \sigma) \triangleq \mathsf{not}\big[\!\big[\mathbf{c}^T\mathbf{s} = \sigma\big]\!\big],$$

  where $[\![\cdot]\!]$ is the Iverson bracket.

**Definition 5.2** (Iverson bracket)**.** The Iverson bracket is defined as

$$[\![s \in S]\!] \triangleq \begin{cases} 1 & \text{if } \ s \in S \\ 0 & \text{if } \ s \notin S \end{cases}.$$

Observe that information is hidden in the last element of the ciphertext. The remaining ciphertext contains key-dependent data used in the decryption process.

Some notes on $\mathbf{w}$ used during encryption:

- $\mathbf{w}$ is *no longer needed* elsewhere after the encryption process, so, the party that encrypts can (and must) discard it after finishing the encryption.
- $\mathbf{w}$ must be kept *secret* and must *not be reused*. If the adversary finds the quantity $\mathbf{b}^T\mathbf{w}$, then they can find $p$. $\mathbf{b}$ is already available, as it is part of the public key.
  - If $\mathbf{w}$ is compromised, then $\mathbf{b}^T\mathbf{w}$ can be computed.
  - If $\mathbf{w}$ was reused, then $\mathbf{b}^T\mathbf{w}$ can be found by running $\mathsf{Enc}_{\mathsf{pk}}(0)$.
- If an adversary tries to run a brute-force attack on $\mathbf{w}$, since $\mathbf{w} \in \mathbb{Z}_2^m$, an exhaustive search on its values would need $2^m$ steps. Moreover, $\mathbf{w}$ was chosen uniformly, so, no value has any probability advantage over any other value.

Another important aspect to highlight regarding the security of this scheme is that in some corner cases the ciphertext may be transparent, meaning that the plaintext is obvious just from the ciphertext. E.g., if $\mathbf{A}$ was zero or if $\mathbf{w}$ was zero, then the ciphertext of the plaintext $p$ would be $(\mathbf{0}, \beta p)$; or, another more general example is if, for any reason, $\mathbf{b}$

[9] When $(\mathbf{A}, \mathbf{b})$ is used as index to $\mathsf{Enc}$, the surrounding parentheses are intentionally omitted for notational clarity.

was zero (which $b$ is known to everyone), then the ciphertext of the plaintext $p$ would be $(\mathbf{A}^T\mathbf{w}, \beta p)$. Normally, we would like to prevent such cases from happening by ruling out such possibilities from the distributions under the cryptosystem algorithms, but for now we will keep it as is for simplicity, as it does not affect the homomorphic properties that we are trying to demonstrate.

Since the plaintext that is being encrypted with this scheme is a bit, the operations that we will try to compute homomorphically are the addition and multiplication modulo 2, or, in binary gates, XOR and AND, respectively. XOR and AND gates can create the NAND gate, which is a universal gate; that means that if we can evaluate the XOR and AND gates homomorphically, then we can evaluate any binary circuit that we want.

### 5.1.2 Correctness & Security

Now, let us proceed with the correctness proof, by starting the algebraic steps in the decryption function: (Reminder: $\mathbf{As} = \mathbf{b}$ and $(\mathbf{c}, \sigma) = \left(\mathbf{A}^T\mathbf{w}, \mathbf{b}^T\mathbf{w} + \beta p\right)$)

$$\begin{aligned}\mathbf{c}^T\mathbf{s} = \mathbf{s}^T\mathbf{c} &= \mathbf{s}^T\mathbf{A}^T\mathbf{w} \\ &= \mathbf{b}^T\mathbf{w} = \underbrace{\mathbf{b}^T\mathbf{w} + \beta p}_{\sigma} - \beta p \\ &= \sigma - \beta p.\end{aligned}$$

$$\left.\begin{array}{l}\text{So, } \mathbf{c}^T\mathbf{s} = \sigma \iff p = 0\ , \\ \qquad \mathbf{c}^T\mathbf{s} \neq \sigma \iff p \neq 0 \iff p = 1\end{array}\right\} \iff p = \mathsf{not}\left[\!\left[\mathbf{c}^T\mathbf{s} = \sigma\right]\!\right].$$

Reminder: $\beta \in \mathbb{Z}_q \setminus \{0\}$.

Note that, since $\mathbf{w}$ is eliminated in the above operations, decryption works just fine even for $\mathbf{w} \in \mathbb{Z}_q^m$, i.e., for $\mathbf{w}$ with non-binary elements. The reason for trying to keep $\mathbf{w}$ small is that $\mathbf{w}$ contributes to error amplification after each homomorphic operation, as we will see when we introduce the error to the scheme.

Sometimes, for notational clarity, we will denote the whole ciphertext as $\bar{\mathbf{c}}$,

$$\bar{\mathbf{c}} = \begin{bmatrix}\mathbf{c} \\ \sigma\end{bmatrix},$$

and the "extended" secret key as $\bar{\mathbf{s}}$,

$$\bar{\mathbf{s}} = \begin{bmatrix}-\mathbf{s} \\ 1\end{bmatrix}.$$

So, we have

$$\begin{aligned}\bar{\mathbf{c}}^T\bar{\mathbf{s}} &= \begin{bmatrix}\mathbf{c}\\ \sigma\end{bmatrix}^T\begin{bmatrix}-\mathbf{s}\\ 1\end{bmatrix}\\ &= -\mathbf{c}^T\mathbf{s}+\sigma\\ &= -(\sigma-\beta p)+\sigma\\ &= \beta p.\end{aligned}$$

Based on that, the decryption function can be rewritten as

$$\mathsf{Dec}_{\bar{\mathbf{s}}}(\bar{\mathbf{c}}) \triangleq \mathsf{not}\left[\!\left[\bar{\mathbf{c}}^T\bar{\mathbf{s}} = 0\right]\!\right],$$

Let us now turn our attention to the security analysis of the cryptosystem.

From the point of view of an attacker, in order to "break" the cryptosystem, namely to extract the plaintext from the ciphertext, they need to find the $\mathbf{b}^T\mathbf{w}$ quantity that obfuscates the bit $p$. For this, there are two things that they can try: either find $\mathbf{w}$ (and then compute $\mathbf{b}^T\mathbf{w}$) or find some $\mathbf{s}' \in \mathbf{s} + \ker(\mathbf{A})$ (and then compute $\mathbf{b}^T\mathbf{w} = \mathbf{s}^T\mathbf{c}$).

For the first case (finding $\mathbf{w}$), we have mentioned that brute-force attack has high complexity, $O(2^m)$.

One other way would be to observe that the attacker knows $\mathbf{c}$ and $\mathbf{A}$, and also knows that $\mathbf{w}$ satisfies $\mathbf{A}^T\mathbf{w} = \mathbf{c}$. But, this is *only a necessary condition, not a sufficient one*; there are many solutions to the system $\mathbf{A}^T\mathbf{x} = \mathbf{c}$ since $\mathbf{A}^T$ is fat, hence the system is underdetermined. One more condition that is necessary on $\mathbf{w}$ that the attacker can assume (if there haven't any homomorphic operations taken place on this ciphertext)[10] to help him narrow down the solutions of $\mathbf{A}^T\mathbf{x} = \mathbf{c}$ is that $\mathbf{w} \in \mathbb{Z}_2^m$. But, even then, the problem is hard. Those conditions together form a problem to which Inhomogeneous SIS can be reduced; and still these are *just necessary but not sufficient conditions* for the true value of $\mathbf{w}$. Attacking on $\mathbf{w}$ is hard.

For the second case (finding $\mathbf{s}' \in \mathbf{s} + \ker(\mathbf{A})$), in most cases $\ker(\mathbf{A}) = \{\mathbf{0}\}$, since $\mathbf{A}$ is tall and randomly generated, hence, most probably $\mathbf{A}$ is full-column rank and the system $\mathbf{A}\mathbf{x} = \mathbf{b}$ will be overdeterminded. However, $\mathbf{s}$ was created such that $\mathbf{A}\mathbf{s} = \mathbf{b}$. According to those, $\mathbf{s}' = \mathbf{s}$ most probably is the unique solution. In order to find $\mathbf{s}' \in \mathbb{Z}_q^n$, the

[10] Below, we will see that if homomorphic operations have been performed on the ciphertext, then $\mathbf{w}$ may go beyond $\mathbb{Z}_2^m$.

attacker has to solve the system $\mathbf{Ax} = \mathbf{b}$. For this, the Gaussian elimination algorithm can be used, which has polynomial time complexity.

However, the Gaussian elimination algorithm is extremely brittle. Errors tend to be amplified when equations are combined.

The problem becomes hard to solve when we add some small error to the equations: $\mathbf{Ax} \approx \mathbf{b}$. This is why the error makes the problem extremely hard. For now, we have supposed that there is no error; we will deal with it afterward.

On the following two chapters that focus on the homomorphic operations of the cryptosystem defined here, we will consider, for simplicity, that $\beta = 1$, hence $\bar{\mathbf{c}}^T\bar{\mathbf{s}} = p$.

### 5.1.3 Homomorphic Addition

Let $(\mathsf{pk}, \mathsf{sk}) = ((\mathbf{A}, \mathbf{b}), \mathbf{s})$ and $(\mathbf{c}_i, \sigma_i) \leftarrow \mathsf{Enc}_{\mathbf{A},\mathbf{b}}(p_i)$, for $i = 1, 2$.

From the encryptor point of view, if those two ciphertexts were added, we have

$$\begin{aligned}(\mathbf{c}_3, \sigma_3) &= (\mathbf{c}_1 + \mathbf{c}_2, \sigma_1 + \sigma_2) \\ &= \left(\mathbf{A}^T\mathbf{w}_1 + \mathbf{A}^T\mathbf{w}_2, \mathbf{b}^T\mathbf{w}_1 + p_1 + \mathbf{b}^T\mathbf{w}_2 + p_2\right) \\ &= \left(\mathbf{A}^T(\mathbf{w}_1 + \mathbf{w}_2), \mathbf{b}^T(\mathbf{w}_1 + \mathbf{w}_2) + (p_1 + p_2)\right) \\ &= \left(\mathbf{A}^T\mathbf{w}_3, \mathbf{b}^T\mathbf{w}_3 + (p_1 + p_2)\right), \text{ where } \mathbf{w}_3 = \mathbf{w}_1 + \mathbf{w}_2.\end{aligned}$$

As we can see, the result ciphertext almost has the known format, with $\mathbf{w}_3 \in \mathbb{Z}_3$, and $\mathbf{b}^T\mathbf{w}_3$ obfuscating the value $p_1 + p_2 \in \mathbb{Z}_3$.[11]

From the decryptor point of view, we have

$$\begin{aligned}\bar{\mathbf{c}}_3^T\bar{\mathbf{s}} &= (\bar{\mathbf{c}}_1 + \bar{\mathbf{c}}_2)^T\bar{\mathbf{s}} \\ &= \mathbf{c}_1^T\bar{\mathbf{s}} + \mathbf{c}_2^T\bar{\mathbf{s}} \\ &= p_1 + p_2,\end{aligned}$$

but, the decryptor must return a result in $\mathbb{Z}_2$, so we can modify it to return the correct result, which is $p_1 \oplus p_2$, like that:

$$\mathsf{Dec}_{\bar{\mathbf{s}}}(\bar{\mathbf{c}}) \triangleq \mathsf{not}\left[\!\left[\bar{\mathbf{c}}^T\bar{\mathbf{s}} \equiv 0 \pmod{p}\right]\!\right].$$

[11] Note that + here is the addition modulo $q$.

### 5.1.4 Homomorphic Multiplication

Let again $(\mathsf{pk}, \mathsf{sk}) = ((\mathbf{A}, \mathbf{b}), \mathbf{s})$ and $(\mathbf{c}_i, \sigma_i) \leftarrow \mathsf{Enc}_{\mathbf{A},\mathbf{b}}(p_i)$, for $i = 1, 2$.

Homomorphic multiplication works with Kronecker product under an alternative "extended" secret key, which is the Kronecker product of the initial "extended" secret key with itself.

What that means is that $p_1 p_2 = \mathsf{Dec}_{\bar{\mathbf{s}}'}(\bar{\mathbf{c}}_1 \otimes \bar{\mathbf{c}}_2)$, with $\bar{\mathbf{s}}' = \bar{\mathbf{s}} \otimes \bar{\mathbf{s}}$.

In order to continue, we remind the Kronecker mixed-product property that homomorphic multiplication takes advantage of: $(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD})$. Using this and the Kronecker transpose property $(\mathbf{A} \otimes \mathbf{B})^T = \mathbf{A}^T \otimes \mathbf{B}^T$, for vectors we have

$$(\mathbf{a} \otimes \mathbf{b})^T (\mathbf{c} \otimes \mathbf{d}) = \Big(\mathbf{a}^T \mathbf{c}\Big)\Big(\mathbf{b}^T \mathbf{d}\Big).$$

We will analyze it just from the decryptor point of view:

$$\begin{aligned} \bar{\mathbf{c}}_3^T \bar{\mathbf{s}}' &= (\bar{\mathbf{c}}_1 \otimes \bar{\mathbf{c}}_2)^T (\bar{\mathbf{s}} \otimes \bar{\mathbf{s}}) \\ &= \Big(\bar{\mathbf{c}}_1^T \bar{\mathbf{s}}\Big)\Big(\bar{\mathbf{c}}_2^T \bar{\mathbf{s}}\Big) \\ &= p_1 p_2. \end{aligned}$$

The last thing to point out is that after an encrypted multiplication, the ciphertext length has grown. this ciphertext length expansion doesn't happen only in this toy example; it also happens in real-world homomorphic encryption schemes, but not with such expansion rate. to encounter this issue, a procedure called *relinearization* follows after an encrypted multiplication, in order to bring the ciphertext back to the initial size.

## 5.2 LWE-Cryptosystem

The problem $\mathbf{Ax} = \mathbf{b}$ (where $\mathbf{A}$ is tall, a solution $\mathbf{x}$ exists, $\mathbf{A}, \mathbf{x}$ are uniformly distributed with elements from $\mathbb{Z}_q$) becomes hard to solve when we add some small error to the equations: $\mathbf{Ax} \approx \mathbf{b}$.

The error term $\mathbf{e} \in \mathbb{Z}_q^m$ is random and small, from the error distribution $\chi$, which usually is some kind of discrete Gaussian.

**Notation**: Denote as $\mathbf{e} \leftarrow \chi$ the sampling procedure from the distribution $\chi$.

### 5.2.1 Definition

The definition of $\mathsf{LWE-PKCS}_{q,m,n,\beta,\chi}$ is almost identical to $\mathsf{LWoE-PKCS}_{q,m,n,\beta}$ above. The only thing that changed is that now we have an error distribution from which an

error value will be sampled during key generation, and that error will be added to $\mathbf{b}$. So, now, instead of sharing the actual $\mathbf{b}$ as part of the public key, we share an erroneous $\mathbf{b}$. Everything is exactly as in Definition 5.1, except $\mathsf{Gen}$ and $\mathsf{Dec}$.

$\mathsf{Gen}$ is defined as

$$\mathsf{Gen}() \triangleq ((\mathbf{A}, \mathbf{b}), \mathbf{s}),$$

where $\mathbf{A} \xleftarrow{\mathsf{R}} \mathbb{Z}_q^{m \times n}$, $\mathbf{s} \xleftarrow{\mathsf{R}} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \chi$, $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \in \mathbb{Z}_q^m$.

$\mathsf{Dec}$ can be defined in various ways based on $\beta$ and $\chi$.

If $\beta = \left\lfloor \frac{q}{2} \right\rceil$ and $\chi$ a distribution such that $\|\mathbf{e}\|_1 < \frac{q}{4}$ for all $\mathbf{e} \leftarrow \chi$, then $\mathsf{Dec}$ can be defined as

$$\mathsf{Dec}_{\mathbf{s}}(\mathbf{c}, \sigma) \triangleq \mathsf{not}\left[\!\left[\mathbf{c}^T\mathbf{s} \overset{q/4}{\approx} \sigma\right]\!\right],$$

where $a \overset{q/4}{\approx} b$ means that $a, b$ are at most $\frac{q}{4}$ close.

In order to define that formally, we first define the absolute value in $\mathbb{Z}_q$ as follows: for $a \in \mathbb{Z}_q$, define $|a|_{\mathbb{Z}_q} \triangleq \min\{a, q-a\}$. The property $|a|_{\mathbb{Z}_q} = |-a|_{\mathbb{Z}_q}$ holds.

For $a, b \in \mathbb{Z}_q, r \in \mathbb{R}$, define $\left(a \overset{r}{\approx}_{\mathbb{Z}_q} b\right) \triangleq \left(|a-b|_{\mathbb{Z}_q} \leq r\right)$.

The $\mathbb{Z}_q$ subscript will be omitted from the relation $\overset{r}{\approx}_{\mathbb{Z}_q}$ for notational clarity.

In our case, where $r = q/4$, / denotes the classical real numbers division.

If $\beta = 1$ and $\chi$ is a distribution over even numbers, then $\mathsf{Dec}$ can be defined as

$$\mathsf{Dec}_{\bar{\mathbf{s}}}(\bar{\mathbf{c}}) \triangleq \mathsf{not}\left[\!\left[\bar{\mathbf{c}}^T\bar{\mathbf{s}} \equiv 0 \pmod 2\right]\!\right].$$

### 5.2.2 Correctness & Security

We proceed as we did in the non-erroneous case:

$$\begin{aligned}\mathbf{c}^T\mathbf{s} = \mathbf{s}^T\mathbf{c} &= \mathbf{s}^T\mathbf{A}^T\mathbf{w} \\ &= (\mathbf{b}-\mathbf{e})^T\mathbf{w} = \underbrace{\mathbf{b}^T\mathbf{w} + \beta p}_{\sigma} - \beta p - \mathbf{e}^T\mathbf{w} \\ &= \sigma - \beta p - \mathbf{e}^T\mathbf{w}.\end{aligned}$$

$$\left.\begin{array}{l}\text{So, } \mathbf{c}^T\mathbf{s} = \sigma - \mathbf{e}^T\mathbf{w} \iff p = 0\ , \\ \quad\ \mathbf{c}^T\mathbf{s} \neq \sigma - \mathbf{e}^T\mathbf{w} \iff p \neq 0 \iff p = 1\end{array}\right\} \iff p = \mathsf{not}\left[\!\left[\mathbf{c}^T\mathbf{s} = \sigma - \mathbf{e}^T\mathbf{w}\right]\!\right].$$

For the first version of $\mathsf{Dec}$ (where $\beta = \left\lfloor \frac{q}{2} \right\rceil$ and $\|\mathbf{e}\|_1 < \frac{q}{4}$), we have that

$$\begin{aligned} \left|\mathbf{c}^T\mathbf{s} - \sigma\right|_{\mathbb{Z}_q} &= \left|-\beta p - \mathbf{e}^T\mathbf{w}\right|_{\mathbb{Z}_q} \\ &= \left|\beta p + \mathbf{e}^T\mathbf{w}\right|_{\mathbb{Z}_q}. \end{aligned}$$

$$\left.\begin{array}{l} \text{So, } \left|\mathbf{c}^T\mathbf{s} - \sigma\right|_{\mathbb{Z}_q} = \left|\mathbf{e}^T\mathbf{w}\right|_{\mathbb{Z}_q} \leq \|\mathbf{e}\|_1 < \frac{q}{4} \iff p = 0\ , \\ \quad\ \left|\mathbf{c}^T\mathbf{s} - \sigma\right|_{\mathbb{Z}_q} = \left|\beta + \mathbf{e}^T\mathbf{w}\right|_{\mathbb{Z}_q} > \frac{q}{4} \iff p = 1 \end{array}\right\} \iff p = \mathsf{not}\left[\!\left[\mathbf{c}^T\mathbf{s} \overset{q/4}{\approx} \sigma\right]\!\right].$$

For the second version of $\mathsf{Dec}$ (where $\beta = 1$ and $\mathbf{e} \in (\mathbb{Z}_q \cap 2\mathbb{Z})^m$), since $\mathbf{e}^T\mathbf{w} \equiv 0 \pmod 2$, we have that

$$\begin{aligned} \mathbf{c}^T\mathbf{s} &= \sigma - \beta p - \mathbf{e}^T\mathbf{w} \\ &\equiv \sigma - \beta p \pmod 2. \end{aligned}$$

$$\left.\begin{array}{l} \text{So, } \mathbf{c}^T\mathbf{s} \equiv \sigma \pmod 2 \iff p = 0\ , \\ \quad\ \mathbf{c}^T\mathbf{s} \not\equiv \sigma \pmod 2 \iff p = 1 \end{array}\right\} \iff p = \mathsf{not}\left[\!\left[\mathbf{c}^T\mathbf{s} \equiv \sigma \pmod 2\right]\!\right].$$

For the security analysis, we have covered everything in the corresponding section of the non-erroneous case. The attacker must solve the underlying LWE problem, which involves solving the erroneous linear equation system $\mathbf{A}\mathbf{x} \approx \mathbf{b}$.

On the following two chapters, we will consider again that $\beta = 1$, hence $\bar{\mathbf{c}}^T\bar{\mathbf{s}} = p + \mathbf{e}^T\mathbf{w}$. We use the second variant of $\mathsf{Dec}$. We will denote the error term $\mathbf{e}^T\mathbf{w}$ as $\bar{e}$, hence $\bar{\mathbf{c}}^T\bar{\mathbf{s}} = p + \bar{e}$.

### 5.2.3 Homomorphic Addition

We repeat the same algebra steps as in the non-erroneous case, and we observe how the error affects the computations.

Let $(\mathsf{pk}, \mathsf{sk}) = ((\mathbf{A}, \mathbf{b}), \mathbf{s})$ and $(\mathbf{c}_i, \sigma_i) \leftarrow \mathsf{Enc}_{\mathbf{A},\mathbf{b}}(p_i)$, for $i = 1, 2$.

From the encryptor point of view, nothing changes since the error is within $\mathbf{b}$. Everything is the same:

$$\begin{aligned} (\mathbf{c}_3, \sigma_3) &= (\mathbf{c}_1 + \mathbf{c}_2, \sigma_1 + \sigma_2) \\ &= \left(\mathbf{A}^T\mathbf{w}_1 + \mathbf{A}^T\mathbf{w}_2, \mathbf{b}^T\mathbf{w}_1 + p_1 + \mathbf{b}^T\mathbf{w}_2 + p_2\right) \\ &= \left(\mathbf{A}^T(\mathbf{w}_1 + \mathbf{w}_2), \mathbf{b}^T(\mathbf{w}_1 + \mathbf{w}_2) + (p_1 + p_2)\right) \\ &= \left(\mathbf{A}^T\mathbf{w}_3, \mathbf{b}^T\mathbf{w}_3 + (p_1 + p_2)\right), \text{ where } \mathbf{w}_3 = \mathbf{w}_1 + \mathbf{w}_2. \end{aligned}$$

From the decryptor point of view, though, we have

$$\begin{aligned}\bar{\mathbf{c}}_3^T\bar{\mathbf{s}} &= (\bar{\mathbf{c}}_1+\bar{\mathbf{c}}_2)^T\bar{\mathbf{s}}\\ &= \mathbf{c}_1^T\bar{\mathbf{s}}+\mathbf{c}_2^T\bar{\mathbf{s}}\\ &= p_1+\bar{e}_1+p_2+\bar{e}_2\\ &= p_1+p_2+\bar{e}_3,\ \text{where } \bar{e}_3=\bar{e}_1+\bar{e}_2.\end{aligned}$$

We can notice that the error accumulates after the encrypted addition.

At a first glance, that seems not to be a problem in our toy cryptosystem, since if $\bar{e}_1, \bar{e}_2$ are even, then $\bar{e}_3$ is even. But, if we perform consecutive encrypted additions, the error may overflow $q$ and mess with our information bit. One can think that this is going not happen if $q$ is even, and that seems to be correct. However, in real-world homomorphic encryption schemes, error accumulates with the encrypted operations performed, and if it exceeds a bound, then we may not be able to decrypt correctly.

In a nutshell, *the error limits the depth of operations that can be performed on the encrypted data.*

### 5.2.4 Homomorphic Multiplication

Let again $(\mathsf{pk},\mathsf{sk}) = ((\mathbf{A},\mathbf{b}),\mathbf{s})$ and $(\mathbf{c}_i,\sigma_i) \leftarrow \mathsf{Enc}_{\mathbf{A},\mathbf{b}}(p_i)$, for $i=1,2$.

We remind how encrypted multiplication works: $p_1p_2 = \mathsf{Dec}_{\bar{\mathbf{s}}'}(\bar{\mathbf{c}}_1\otimes\bar{\mathbf{c}}_2)$, with $\bar{\mathbf{s}}' = \bar{\mathbf{s}}\otimes\bar{\mathbf{s}}$.

From the decryptor point of view:

$$\begin{aligned}\bar{\mathbf{c}}_3^T\bar{\mathbf{s}}' &= (\bar{\mathbf{c}}_1\otimes\bar{\mathbf{c}}_2)^T(\bar{\mathbf{s}}\otimes\bar{\mathbf{s}})\\ &= \left(\bar{\mathbf{c}}_1^T\bar{\mathbf{s}}\right)\left(\bar{\mathbf{c}}_2^T\bar{\mathbf{s}}\right)\\ &= (p_1+\bar{e}_1)(p_2+\bar{e}_2)\\ &= p_1p_2+p_1\bar{e}_2+p_2\bar{e}_1+\bar{e}_1\bar{e}_2.\end{aligned}$$

Now, we notice that the error accumulates even faster after the encrypted multiplication. The same thing holds for real-world homomorphic encryption schemes.

*Encrypted multiplication accumulates the error much faster than encrypted addition.*

## 5.3 A Secret-Key Variant

A secret key variant of the above public-key cryptosystem merely changes the encryption procedure. The secret key is the same as above, and there is no public key. The ciphertext format does not change, i.e., it is still an element of $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

What changes in the public-key encryption function

$$\mathsf{Enc}_{\mathbf{A},\mathbf{b}}(p) \triangleq \Big(\mathbf{A}^T\mathbf{w}\ ,\ \mathbf{b}^T\mathbf{w} + \beta p\Big), \quad \text{where} \quad \mathbf{w} \xleftarrow{\mathsf{R}} \mathbb{Z}_2^m$$

is that instead of choosing a random subset sum of the rows of $\mathbf{A}$, we generate a random vector $\mathbf{a} \in \mathbb{Z}_q^n$, and instead of $\mathbf{b}^T\mathbf{w}$ $(= (\mathbf{A}\mathbf{s} + \mathbf{e})^T\mathbf{w})$ we have $\mathbf{a}^T\mathbf{s} + e$.

Mathematically expressed, the secret-key encryption function is

$$\mathsf{Enc}_{\mathbf{s}}(p) \triangleq \Big(\mathbf{a}\ ,\ \mathbf{a}^T\mathbf{s} + e + \beta p\Big), \quad \text{where} \quad \mathbf{a} \xleftarrow{\mathsf{R}} \mathbb{Z}_q^n \text{ and } e \leftarrow \chi',$$

for an appropriate $\chi'$ error distribution.

The decryption still works the same. Knowing the secret key $\mathbf{s}$ and using the first part of the ciphertext (namely, $\mathbf{a}$) it computes the term $\mathbf{a}^T\mathbf{s}$ that obfuscates the encrypted bit. Then, based on the variant, decryptor used the appropriate decision rule.

For $\beta = 1$ (and $e$ is even), if an attacker collects many ciphertexts

$$\begin{gathered}\Big(\mathbf{a}_1, \mathbf{a}_1^T\mathbf{s} + e_1 + p_1\Big) \\ \vdots \\ \Big(\mathbf{a}_k, \mathbf{a}_k^T\mathbf{s} + e_k + p_k\Big)\end{gathered}$$

then, they have created an LWE problem where the information bit is embedded in the error, i.e.,

$$\mathbf{A}\mathbf{x} = \mathbf{b} + \mathbf{e}$$

where

$$\mathbf{A} = \begin{bmatrix}\mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_k^T\end{bmatrix}\ ,\ \mathbf{b} = \begin{bmatrix}\mathbf{a}_1^T\mathbf{s} \\ \vdots \\ \mathbf{a}_k^T\mathbf{s}\end{bmatrix}\ ,\ \mathbf{e} = \begin{bmatrix}e_1 + p_1 \\ \vdots \\ e_k + p_k\end{bmatrix}.$$

# 6 Implementation of Algorithms for Secure Computation

Microsoft SEAL [8] is an open-source homomorphic encryption library that enables secure computation directly on encrypted data, without requiring decryption. Developed by Microsoft, it offers a variety of encryption schemes, including the BFV and CKKS schemes, for working with integers and real numbers, respectively. Developed by Microsoft, it offers a variety of encryption schemes, including the BFV, BGV, and CKKS schemes, for working with integers and real numbers, which are commonly used for secure computations.

In this thesis project, the most recent version of SEAL, version 4.1.2 (as of the time of completion), was used.

## 6.1 Introduction to Microsoft SEAL

We proceed introducing the basic notions and specifications that one needs to know in order to design and develop algorithms using SEAL.

Microsoft SEAL provides low-level primitives for encrypted arithmetic. Those are:

- `negate`
- `add`
- `sub`
- `multiply`
- `square`
- `exponentiate`
- `rotate_⟨rows|columns|vector⟩`
- `complex_conjugate`

All the functions mentioned above, have an `inplace` variant, where the result is stored in one of the input operands. The `inplace` variant of a function is a somewhat more performant than the non-`inplace` one (avoids a ciphertext copy). If such behavior is needed, it is suggested that we use the `inplace` variant.

`square` is for the multiplication of a ciphertext with itself once and `exponentiate` is for the same thing but for the given (integer non-encrypted) exponent. Those functions are more performant than using `multiply` and they should be prefered to the latter.

Additionally, there are the functions `add_many`, `multiply_many`, for more performant addition/multiplication on many ciphertexts. "Performance" in Homomorphic Encryption, except from time and space resources, also takes into account the depth of the encrypted computations. More on that later.

Furthermore, for `add`, `sub`, `multiply` functions, there are the `add_plain`, `sub_plain`, `multiply_plain` variants, where one of the operants is a plaintext. The former functions (i.e., the non-`plain` variants) are ciphertext-ciphertext operations, i.e., both their input operands are ciphertexts. On the other hand, the latter functions (i.e., the `plain` variants) are plaintext-ciphertext operations, i.e., one input operand is a plaintext while the other one is a ciphertext. The `plain` variants of the aforementioned functions are much more performant and, in the case of multiplication, no relinearization is needed afterward a plaintext-ciphertext multiplication, in contrast to a ciphertext-ciphertext multiplication. About relinearization, see below.

In addition to the above functions, there are also these operating on a single ciphertext (that also have an `inplace` variant):

- `relinearize`
- `rescale`
- `mod_switch`

`rescale` is used just in CKKS.

Those functions are used to regulate a ciphertext in some way. We will explain how are those used in the course of this chapter.

SEAL encrypts vectors of numbers, namely, many numbers at a time, and the encrypted operations offered by the library mentioned above (except `rotate` operations) operate in an SIMD (Single Instruction Multiple Data) manner on the encrypted data, i.e., they act elementwise on the vector of numbers encoded in each plaintext/ciphertext. `rotate` operation changes the order of the numbers within a ciphertext by rotating them.

SEAL offers a serialization functionality for the encrypted data and the different kind of keys, enabling easy network communication.

SEAL supports three leveled fully homomorphic encryption schemes:

- BFV, BGV: performing encrypted integer modular arithmetic
- CKKS: performing encrypted real or complex number arithmetic (approximate computations)

  - CKKS is much more performant than BFV, BGV

Using SEAL, shallow and wide computations are more feasible than deep computations.

For example, let $\mathbf{x}, \mathbf{y} \in \mathbb{C}^n$ be two $n$-dimensional complex vectors and $a_1, \ldots, a_{n+1} \in \mathbb{C}$ be $n+1$ complex numbers. Both the products $\mathbf{x} \odot \mathbf{y}$ and $\prod_{i=1}^{n+1} a_i$ perform $n$ multiplications, but computing the first one is more feasible.

In SEAL, before we start encrypting-computing-decrypting, some encryption parameters need to be set. These parameters configure various things as described below:

- `poly_modulus_degree`: Number of slots (data per plaintext)
  - Increasing it, increases security, but decreases computational performance.
- `coeff_modulus`: Configures the computational depth
  - Increasing it, increases the computational depth, but also increases the ciphertext size.
  - Its upper bound is determined by the `poly_modulus_degree`.
- `plain_modulus` (only in BFV, BGV): Plaintext data modulus
  - Increasing it, increases the number range, but also increases noise budget consumption and decreases initial noise budget.[12]

In this project, the CKKS cryptosystem was used the most, so, we describe here its technical details in order to understand what should be taken into consideration during the algorithmic implementation on top of it.

As already mentioned, SEAL encrypts vectors of numbers. A single CKKS plaintext encodes many floating point complex numbers. How many? `poly_modulus_degree`/2. Encrypting it, a single ciphertext emerges encrypting all those numbers. The order of those numbers is preserved between encryption and decryption. If we desire to encrypt just a single number, we can do it, but it would be a waste of space. The way the system works is on multiple numbers; encrypting just one number will not make the plaintext and ciphertext smaller. What is going to happen in this case is that the rest numbers will be set to zero.

All the encrypted operations operate in an SIMD manner, i.e., they act elementwise on the numbers encoded in each plaintext/ciphertext. “Rotate” operation changes the order

[12]Noise budget in BFV, BGV is a quantity that puts a depth limit in the encrypted computations. Every encrypted operation consumes noise budget. If there is no more noise budget, we can’t perform more encrypted operations and be sure about the decryption correctness.

of the numbers within a ciphertext by rotating them.

After an encrypted ciphertext-ciphertext multiplication, the size of the result ciphertext has increased. It can be brought back to the initial size by using `relinearize`. Be careful, though, because relinearization has a somewhat significant computational cost itself. It is not a rule that it should be done right after multiplication. Depending on the situation, it may be wise to do it after other operations, or (more rarely) not to do it at all.

"Encoding" is the procedure that converts a sequence of floating point complex numbers into a plaintext. Encoding involves an extra parameter in CKKS: the scale.

"Scale" is a number that is part of the plaintext (and, if encrypted, part of the ciphertext) that configures the accuracy in the results of the encrypted computations. The bigger the scale, the greater the accuracy. Scale has a similar role to the exponent in the IEEE 754 floating-point standard.

Encoding involves another parameter, the "modulus switching chain level". This "chain level" becomes part of the plaintext (and, if encrypted, part of the ciphertext), too. It is very important to understand very well those parameters in order to be able to implement algorithms on top of CKKS in SEAL.

The "modulus switching chain" consists of "modulus switching chain levels" (or, for conciseness, just "chain levels"). Each chain level contains a "coefficient modulus", which is a product of prime numbers. Those prime numbers are configured via the `coeff_modulus` encryption parameter mentioned above, which specifies the number of primes and the bit length of each one. If those primes are $q_0, \ldots, q_n$, in the $i$-th chain level, the coefficient modulus is $q_0 \cdots q_i$. Any plaintext and ciphertext has a modulus switching chain and a chain level as part of it.[13]

The highest chain level is the one with the coefficient modulus $q_0 \cdots q_n$. The highest chain level is used just for the keys, and it is called the "key level". The rest levels are used for plaintexts and ciphertexts. So, the highest "data level" is the one with the coefficient modulus $q_0 \cdots q_{n-1}$. A plaintext or ciphertext can only move down the chain, never up. "Moving down" means that the old coefficient modulus is divided by the last prime. The size of the ciphertext depends linearly on the number of primes in the product of its chain level.

[13]The modulus switching chain is not technically part of the plaintext or ciphertext, it is somewhere separately, but they are connected. The plaintext or ciphertext just keeps a hash determining the chain level.

The coefficient modulus in the current chain level plays the role of `plain_modulus` in CKKS. So, the scaled message must not get too close to it.

For any ciphertext-ciphertext or plaintext-ciphertext operation to be performed, the two operants must have the same "modulus switching chain", the same "chain level" and the same "scale".

There are two ways to move down to the modulus switching chain.[14] The first way is by using `mod_switch`, which just divides the coefficient modulus by the last prime number in this level. The second way is by using `rescale`, which does the same as above, but, additionally, scales down the ciphertext by that last prime; the result ciphertext scale is the previous scale divided by that last prime.

Now, let us observe how is the scale affected during operations through the following very simple examples.

Let $\bar{c}_i = c_i \cdot 2^{40}$ for $i = 1, 2$ be two scaled ciphertexts.

Adding them, we have

$$\bar{c}_1 + \bar{c}_2 = c_1 \cdot 2^{40} + c_2 \cdot 2^{40} = (c_1 + c_2) \cdot 2^{40}.$$

The scale did not change.

Multiplying them, we have

$$\bar{c}_1 \bar{c}_2 = c_1 \cdot 2^{40} \cdot c_2 \cdot 2^{40} = (c_1 c_2) \cdot 2^{80}.$$

The scale did change. It is squared.

In the case of multiplication, the result ciphertext with its new scale is valid, but if we continue using it as is in consecutive multiplications, the scale will be increased exponentially, and soon will be out of bounds. Remember that in any operation, both operants must have the same scale.

What we can do here, if we have set the initial `coeff_modulus` parameter correctly such that the primes $q_1, \ldots, q_{n-1}$ have almost the same bit length with our initial scale[15], then

[14] Here we are mentioning moving down by one step. Moving down many steps at once theoretically is the same as moving down that number of steps one-by-one.

[15] The bit length of the primes sometimes should be chosen very carefully, taking into account that if the following multiplication-rescale procedure is repeated many times, then the scale may increase or decrease each time at a faster rate, and, eventually, going out of bounds.

performing a `rescale` here will create a valid ciphertext with approximately equal to the initial one:

$$\mathsf{rescale}_{i \to i-1}(c_3 \cdot 2^{80}) = c_3' \cdot \frac{2^{80}}{q_i}$$

with $2^{80}/q_i \approx 2^{40}$.

However, we must be careful if we want to perform another encrypted operation on the rescaled ciphertext. As mentioned above, the scales of the operants must be equal; approximately equal does not work. There are two ways to face this issue:

1. Bring them to the same chain level via `mod_switch` and manually assign the scale value to either of those such that they are equal.
2. Bring them to the same chain level via multiplications and rescales.

The first way is faster (it avoids extra multiplications), but at the cost of accuracy. Messing up with the scale manually will affect the value upon decryption. If a slight loss of accuracy is acceptable, this option may be preferable.

The second way avoids the accuracy issues that the first way had by changing the scale, not manually, but through the operations provided by SEAL. This is the most accurate way but comes at the computational cost that extra dummy multiplications may have.

For example, assume that we want to compute this (ignoring relinearization): $c_1c_2c_3$ (all $c_i$ having the same chain level and scale). At first, compute $c_1c_2$ and rescale it. Now, $c_1c_2$ and $c_3$ have different chain levels and scales.

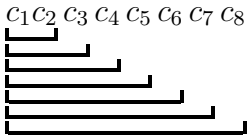
- For the first way, we perform `mod_switch` on $c_3$, then $c_1c_2$ and $c_3$ will have the same chain level, but their scales will be different (approximately equal, but not equal). Then we manually set either scale to be equal to the other one, and now we can proceed to the final multiplication.
- For the second way, we perform a dummy multiplication $1 \cdot c_3$, and then rescale it. Now, $c_1c_2$ and $1 \cdot c_3$ have same both the chain level and the scale, hence we can proceed to the final multiplication.

The same things would apply if the second operation was another one, e.g., $c_1c_2 + c_3$.

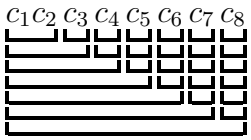
In this project, the second way was used.

Now, let us see the case where we have a series of multiplications to execute, $\prod_{n=1}^{8} c_n$. If we simply perform them one-by-one from left to right, then the multiplication depth will

be as much as the multiplications to be executed.

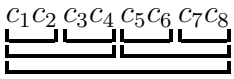



If we also consider the procedure to bring the multiplying ciphertexts to the same chain level and scale (using either of the aforementioned ways), then we have the following image:




If we were using the second way (doing dummy multiplications), this means that we have a lot more multiplications to perform, which would be time inefficient.

Even if we were using the first way (avoiding the dummy multiplications), there is a smarter way that reduces the multiplication depth from linear (to the number of multiplications) to logarithmic. What we do to accomplish that is to multiply pairs in the same level whenever possible.




The number of multiplications did not change; 7 multiplications. However, the depth reduced from 7 to 3.

In the Figure 10, a high level view of the whole procedure is illustrated.

The $\left(\mathbb{C}^{n/2}, \mathsf{scale}\right)$ case is for CKKS, while the $\mathbb{Z}_r^n$ case is for BFV and BGV.

The encode procedure is working a lot differently in CKKS, in contrast to BFV and BGV. However, the high level view is the same.

$n$ defines the batch size (in SEAL it is called "`poly_modulus_degree`"); it is a power of two, and, in SEAL, the possible values of it are $2^{10}, 2^{11}, 2^{12}, 2^{13}, 2^{14}, 2^{15}$.

$r$ is called the "plaintext modulus" (in SEAL, specifically in BFV and BGV, it is called "`plain_modulus`"). $r$ is a parameter only for BFV and BGV. For CKKS, $r$ shows up after encoding, nevertheless, it is configured internally.

$q$ is called the "ciphertext modulus" (in SEAL it is called "`coeff_modulus`"). As mentioned above, in the case of CKKS, $q$ is a product of primes. For CKKS, when dropping

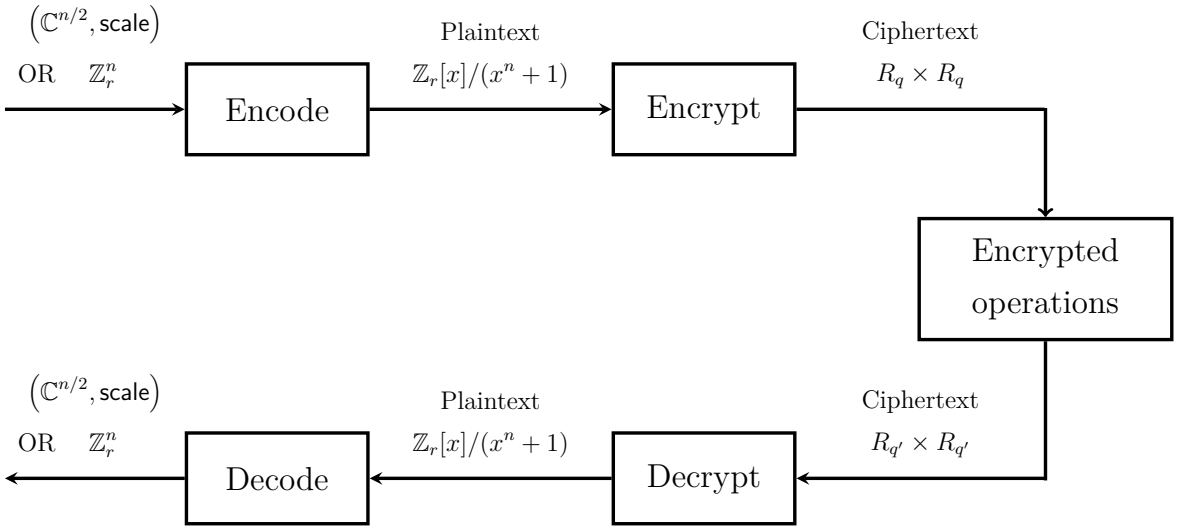



Figure 10: High level schematic.

one level in the chain through `rescale`, $q$ is divided by the last prime in the product, i.e., the last prime is removed from the product. That is why after encrypted operations, we may have a different "ciphertext modulus", $q'$. Nonetheless, in any case, $r \leq q' \leq q$ will hold.

$R_q$ is a polynomial ring; more specifically, $R_q = \mathbb{Z}_q/(x^n + 1)$.

## 6.2 Algorithms Implemented in Microsoft SEAL

We would like to compute some mathematically more sophisticated operations than the low-level primitives provided from SEAL (e.g., $x^{-1}, \sqrt{x}, |x|$). To accomplish that, we select some "basic" mathematical functions and create iterative algorithms that approximate their result. Other mathematical functions used in this project are computed by making use of the "basic" functions. The iterative algorithms behind the "basic" functions use exclusively the low-level primitives that we have at our disposal (roughly speaking: addition, subtraction, multiplication).

### 6.2.1 Computing the Functions $x \mapsto x^{-1}$ and $x \mapsto 1/\sqrt{2x}$

About the "basic" functions, the way to create such algorithms that was used here is the Newton-Raphson method. Namely, we create a mathematical function $f$ that its zero is the desirable result, then run the Newton-Raphson iterations to approximate that that zero. The important thing here is that the resulting Newton-Raphson iteration involves only the low-level primitives of the library mentioned above; hence, it is crucial that $f$ is defined in such a way that the result involves only the available operations. For example,

for $x \mapsto x^{-1}$, if we define $f(y) = y - x^{-1}$, then this is not going to work because the resulting Newton-Raphson iterations will contain $x^{-1}$ in it.

Each algorithm works as follows:

- The user gives an encrypted input $x$ which is the operand of the desired mathematical operation.
- The user gives an initial value $y_0$ (either encrypted or decrypted).
  - The value $y_0$ is a prediction of the final result. The less accurate this prediction is, the slower the algorithm will converge.
  - The value $y_0$ must adhere to some constraints. If it does not, then the algorithm will diverge.
- The user gives the number of iterations to be performed.
- The algorithm runs and returns the computed result.

We create the basis functions $x \mapsto x^{-1}$ and $x \mapsto 1/\sqrt{2x}$. The rest of functions used here are computed using just $x \mapsto 1/\sqrt{2x}$.

The reasons those functions were chosen are:

- $x \mapsto x^{-1}$ can be computed using the result of $x \mapsto 1/\sqrt{2x}$ like that: $x^{-1} = 2\left(1/\sqrt{2x}\right)^2$. However, it is far more efficient if it has its own algorithm, than using $x \mapsto 1/\sqrt{2x}$. In addition, $x \mapsto x^{-1}$ can be computed in an even more performant way than Newton-Raphson by using its Taylor expansion series, because its Taylor expansion series has some symmetry that lets us factorize it nicely, saving almost half the multiplication depth consumed compared to the Newton-Raphson approach, for the same accuracy. Nevertheless, we will show here the Newton-Raphson way, too.
- For the $x \mapsto 1/\sqrt{2x}$ now, the reason for choosing this operation is that the initial purpose was to compute $x \mapsto \sqrt{x}$, but the Newton-Raphson iteration for this contained operations beyond the SEAL primitives (addition, subtraction, multiplication). On the other hand, for its multiplicative inverse $x \mapsto 1/\sqrt{x}$, the Newton-Raphson iteration was appropriate; and it is very easy to obtain $\sqrt{x}$ from $1/\sqrt{x}$: just multiply the latter by $x$. The coefficient 2 on $x$ is used to save one multiplication in the Newton-Raphson iteration process.

Evaluate $x \mapsto x^{-1}$ for $x \neq 0$:

$$y_{n+1} = 2y_n - xy_n^2 \quad \text{converges for} \quad \begin{Bmatrix} 0 < y_0 < \frac{2}{x} & \text{if} \;\; x > 0 \\ \frac{2}{x} < y_0 < 0 & \text{if} \;\; x < 0 \end{Bmatrix}.$$

*Proof.* Let $f(y) = y^{-1} - x$. Then $f'(y) = -y^{-2}$.

$$\begin{aligned} y_{n+1} &= y_n - \frac{f(y_n)}{f'(y_n)} \\ &= y_n - \frac{y_n^{-1} - x}{-y_n^{-2}} \\ &= 2y_n - xy_n^2 \end{aligned}$$

Convergence:

- For $x > 0$ : $2y_0 - xy_0^2 > 0 \iff y_0\left(\frac{2}{x} - y_0\right) > 0 \iff 0 < y_0 < \frac{2}{x}$
- For $x < 0$ : $2y_0 - xy_0^2 < 0 \iff y_0\left(\frac{2}{x} - y_0\right) > 0 \iff \frac{2}{x} < y_0 < 0$

□

Evaluate $x \mapsto \frac{1}{\sqrt{2x}}$ for $x > 0$:

$$y_{n+1} = \frac{3}{2}y_n - xy_n^3 \quad \text{converges for} \;\; 0 < y_0 < \sqrt{\frac{3}{2x}}.$$

*Proof.* Let $f(y) = y^{-2} - 2x$. Then $f'(y) = -2y^{-3}$.

$$\begin{aligned} y_{n+1} &= y_n - \frac{f(y_n)}{f'(y_n)} \\ &= y_n - \frac{y_n^{-2} - 2x}{-2y_n^{-3}} \\ &= \frac{3}{2}y_n - xy_n^3 \end{aligned}$$

Convergence:

- For $x > 0$ : $\frac{3}{2}y_0 - xy_0^3 > 0 \iff y_0\left(\frac{3}{2x} - y_0^2\right) > 0 \iff 0 < y_0 < \sqrt{\frac{3}{2x}}$

□

Or, for the more generalized case $x \mapsto \frac{1}{\sqrt[k]{kx}} = (kx)^{-\frac{1}{k}}$ for $k \in \mathbb{Z}^{\geq 1}$ and $x > 0$ if $k$ is even, else $x \neq 0$:

$$y_{n+1} = \left(1 + \frac{1}{k}\right) y_n - x y_n^{k+1} \quad \text{converges for} \begin{cases} 0 < y_0 < \sqrt[k]{\dfrac{1+1/k}{x}} & \text{if } x > 0 \\ \sqrt[k]{\dfrac{1+1/k}{x}} < y_0 < 0 & \text{if } k \text{ odd and } x < 0 \end{cases}.$$

*Proof.* Let $f(y) = y^{-k} - kx$. Then $f'(y) = -ky^{-k-1}$.

$$\begin{aligned} y_{n+1} &= y_n - \frac{f(y_n)}{f'(y_n)} \\ &= y_n - \frac{y_n^{-k} - kx}{-k y_n^{-k-1}} \\ &= \left(1 + \frac{1}{k}\right) y_n - x y_n^{k+1} \end{aligned}$$

Convergence:

- For $k$ even or odd and $x > 0$ : $\left(1 + \frac{1}{k}\right) y_0 - x y_0^{k+1} > 0 \iff y_0\left(\frac{1+1/k}{x} - y_0^k\right) > 0 \iff 0 < y_0 < \sqrt[k]{\frac{1+1/k}{x}}$
- For $k$ odd and $x < 0$ : $\left(1 + \frac{1}{k}\right) y_0 - x y_0^{k+1} < 0 \iff y_0\left(\frac{1+1/k}{x} - y_0^k\right) > 0 \iff \sqrt[k]{\frac{1+1/k}{x}} < y_0 < 0$

□

Let us now see what is the multiplication depth for each of those methods. For $k$ multiplication terms (or, equivalently, for $k-1$ multiplications), the depth is $\lceil \lg(k+1) \rceil$.

Multiplications in the $x \mapsto x^{-1}$ Newton-Raphson iteration:

$$y_{n+1} = \underbrace{2 y_n} - \underbrace{x\, \underbrace{y_n^2}}$$

As we can see, the second term is the bottleneck since it involves the most multiplications; 2, to be specific. Each iteration consumes $\lceil \lg 3 \rceil = 2$ chain levels.

Multiplications in the $x \mapsto \frac{1}{\sqrt{2x}}$ Newton-Raphson iteration:

$$y_{n+1} = \underbrace{\frac{3}{2} y_n} - \underbrace{x y_n\, \underbrace{y_n^2}}$$

Here, each iteration consumes $\lceil \lg 4 \rceil = 2$ chain levels.

Multiplications in the $x \mapsto \frac{1}{\sqrt[k]{kx}} = (kx)^{-\frac{1}{k}}$ Newton-Raphson iteration:

$$y_{n+1} = \left(1 + \frac{1}{k}\right) y_n - x y_n^{k+1}$$

Here, each iteration consumes $\lceil \lg(k+2) \rceil$ chain levels.

Now, let us present a more multiplication-depth-friendly way to evaluate $x \mapsto x^{-1}$.

For $x \mapsto \frac{1}{x}$, we can use the known formula

$$\frac{1}{1-x} \overset{|x|<1}{=} \sum_{n=0}^{\infty} x^n$$

to create an algorithm, like that:

$$\frac{1}{x} = \frac{1}{1-(1-x)} \overset{|1-x|<1}{=} \sum_{n=0}^{\infty} (1-x)^n.$$

But, the above formula converges only for $|x-1| < 1 \iff 0 < x < 2$. One way to expand the input range for the user is to insert a normalization parameter $a$, which will also be part of the input.

The new formula will be

$$\frac{1}{x} = a\frac{1}{ax} \overset{|1-ax|<1}{=} a \sum_{n=0}^{\infty} (1-ax)^n \tag{6.1}$$

which converges for $|ax-1| < 1 \iff 0 < ax < 2$.

Observe that the above series is the Taylor expansion of $\frac{1}{x}$ centered at $\frac{1}{a}$.

$a$ has the same role here as $y_0$ in the Newton-Raphson method; it is a prediction of the result. The closer $a$ is to $x^{-1}$, the faster the convergence will be. In the case of the perfect prediction $a = x^{-1}$, the formula has converged from the first sum term, since the rest terms are 0.

Furthermore, observe that the convergence region is the same as in Newton-Raphson method; for an input $x$, the series converges for

$$\begin{aligned} 0 < a < \frac{2}{x} &\quad \text{if } x > 0, \\ \frac{2}{x} < a < 0 &\quad \text{if } x < 0. \end{aligned}$$

An algorithm implementing the above formula, would approximate the result by summing a finite number of terms, let them be $N$.

Assuming that $N$ is a power of two, we have

$$\begin{aligned}
\sum_{n=0}^{N-1} x^n &= \sum_{n=0}^{N/2-1} x^n + \sum_{n=N/2}^{N-1} x^n \\
&= \sum_{n=0}^{N/2-1} x^n + \sum_{n=N/2}^{N-1} x^{N/2} x^{n-N/2} \\
&= \sum_{n=0}^{N/2-1} x^n + x^{N/2} \sum_{n=0}^{N/2-1} x^n \\
&= \left(1 + x^{N/2}\right) \sum_{n=0}^{N/2-1} x^n \\
&\overset{(*)}{=} \left(1 + x^{N/2}\right)\left(1 + x^{N/4}\right) \cdots (1 + x) \\
&= \prod_{n=1}^{\lg N} \left(1 + x^{N/2^n}\right) \\
&= \prod_{n=1}^{\lg N} \left(1 + x^{2^{\lg N - n}}\right) \\
&= \prod_{n=0}^{\lg N - 1} \left(1 + x^{2^n}\right),
\end{aligned}$$

where in $(*)$ we are recursively applying the above result.

This reduces the number of terms from $N$ to $\lg N$. Of course the latter are multiplication terms, but we will see below that this product can be computed algorithmically very conveniently with respect to the chain levels. As well, we should not forget that the sum terms in the sum formula had powers to be computed; those were multiplications, too.

Using the Taylor expansion result from (6.1) with the above factorization for a power-of-two $N$, we get

$$a \sum_{n=0}^{N-1} (1 - ax)^n = a \prod_{n=0}^{\lg N - 1} \left[1 + (1 - ax)^{2^n}\right].$$

Implementing it, we observe that each iteration consumes only one chain level (except for the first one that consumes two):

$$a \cdot \underbrace{\left[1 + (1 - ax)\right]}_{\sqcup} \cdot \underbrace{\left[1 + (1 - ax)^2\right]}_{\sqcup} \cdot \underbrace{\left[1 + \left((1 - ax)^2\right)^2\right]}_{\sqcup} \cdot \underbrace{\left[1 + \left(\left((1 - ax)^2\right)^2\right)^2\right]}_{\sqcup} \cdots$$

This happens because the result of each iteration (very conveniently) has the same chain level with the next term that is going to be multiplied with. Additionally, every iteration performs only one multiplication, squaring the term $(1-ax)^{2^i}$ from the previous iteration.

If $a$ is given non-encrypted, then we can save the first dummy multiplication by isolating the product term, thus saving one chain level:

$$a\sum_{n=0}^{N-1}(1-ax)^n = \left(2a-a^2x\right)\prod_{n=1}^{\lg N-1}\left[1+(ax-1)^{2^n}\right].$$

Since $a$ is not encrypted, computing $a^2$ will consume no chain level.

An important thing to note here is that the constraints for this algorithm

$$\begin{cases} 0 < a < \dfrac{2}{x} & \text{if } \ x > 0 \\ \dfrac{2}{x} < a < 0 & \text{if } \ x < 0 \end{cases}$$

force us to know in advance the sign of the input $x$ in order to provide the algorithm with a same sign value for $a$. However, when computing on encrypted data, we may know nothing about the sign of $x$. If this is the case, a way to address that is to compute $x^2 \mapsto x^{-2}$ using the above algorithm (now, the sign is definitely positive) with prediction $a^2$, and then multiply the result with $x$.

Generally, the closer to 0 is $a$, the greater the range of the possible inputs is, but, also, the slower the convergence will be. Additionally, there may be arithmetic precision issues.

### 6.2.2 Computing Other Mathematical Functions

Before introducing the algorithms for $x \mapsto x^{-1}$ and $x \mapsto 1/\sqrt{2x}$, it was mentioned that the rest functions used in this project are computed using just $x \mapsto 1/\sqrt{2x}$.

Define $\varrho(x) \triangleq \frac{1}{\sqrt{2x}}$ for any $x > 0$, and let us now see how are those functions computed using $\varrho$.

Evaluate $x \mapsto \frac{1}{\sqrt{x}}$ for $x > 0$:

$$\frac{1}{\sqrt{x}} = \varrho(x/2) = \sqrt{2}\varrho(x).$$

Evaluate $x \mapsto \sqrt{x}$ for $x > 0$:

$$\sqrt{x} = x\varrho(x/2) = x\sqrt{2}\varrho(x).$$

Evaluate $x \mapsto \frac{1}{|x|}$ for $x \neq 0$:

$$\frac{1}{|x|} = \varrho(x^2/2) = \sqrt{2}\varrho(x^2).$$

Evaluate $x \mapsto |x|$ for $x \neq 0$:

$$|x| = x^2 \varrho(x^2/2) = x^2 \sqrt{2} \varrho(x^2).$$

Evaluate $x \mapsto \operatorname{sgn}(x)$ for $x \neq 0$:

$$\operatorname{sgn}(x) = x \varrho(x^2/2) = x \sqrt{2} \varrho(x^2).$$

Evaluate $(x_1, x_2) \mapsto \max(x_1, x_2)$ for $x_1 \neq x_2$:

$$\max(x_1, x_2) = \frac{x_1 + x_2}{2} + \frac{(x_1 - x_2)^2}{\sqrt{2}} \varrho\Big((x_1 - x_2)^2\Big).$$

Evaluate $(x_1, x_2) \mapsto \min(x_1, x_2)$ for $x_1 \neq x_2$:

$$\min(x_1, x_2) = \frac{x_1 + x_2}{2} - \frac{(x_1 - x_2)^2}{\sqrt{2}} \varrho\Big((x_1 - x_2)^2\Big).$$

In the above functions, when $\varrho$ is involved in multiplications, it is better to use the variant where the constant is not in the argument of $\varrho$. For example, in $x \mapsto \sqrt{x}$, it is better to use $x\sqrt{2}\varrho(x)$ instead of $x\varrho(x/2)$, and prioritize the multiplications like this

$$\Big(x\sqrt{2}\Big)\varrho(x)$$

instead of this

$$x\Big(\sqrt{2}\varrho(x)\Big),$$

since the computation of $\varrho(x)$ is expected to consume some chain levels, so we try not to burden it more.

Now, we proceed to the computation of an encrypted if-else of the following form:

`if` $a < b$ :
    $r = r_1$
`else if` $a > b$ :
    $r = r_2$

or, mathematically expressed:

$$r = \begin{cases} r_1 & \text{if } a < b \\ r_2 & \text{if } a > b \end{cases}.$$

This is equivalent to

$$r = \frac{r_1 + r_2}{2} + \frac{(b - a)(r_1 - r_2)}{\sqrt{2}} \varrho\Big((b - a)^2\Big)$$

*Proof.* For $a \neq b$, let

$$s \triangleq \mathbb{I}(a < b) = \begin{cases} 1 & \text{if } a < b \\ 0 & \text{if } a > b \end{cases}.$$

Then

$$\begin{aligned} r &= sr_1 + (1-s)r_2 \\ &= r_2 + s(r_1 - r_2) \end{aligned} \tag{6.2}$$

Now, let

$$s' \triangleq \begin{cases} 1 & \text{if } a < b \\ -1 & \text{if } a > b \end{cases} = \frac{b-a}{|b-a|}.$$

Observe that

$$s' = 2s - 1.$$

Now, let

$$s'' \triangleq \varrho\Big((b-a)^2\Big) = \frac{1}{\sqrt{2(b-a)^2}} = \frac{1}{\sqrt{2}|b-a|}.$$

Observe that

$$\begin{aligned} s'' &= \frac{1}{\sqrt{2}(b-a)} \frac{b-a}{|b-a|} \\ &= \frac{1}{\sqrt{2}(b-a)} s' \\ &= \frac{1}{\sqrt{2}(b-a)} (2s-1) \\ &= \frac{\sqrt{2}}{b-a} s - \frac{1}{\sqrt{2}(b-a)} \\ \iff \quad s &= \frac{b-a}{\sqrt{2}} s'' + \frac{1}{2}. \end{aligned}$$

Hence, substituting this into (6.2), we have

$$\begin{aligned} r &= r_2 + \left(\frac{b-a}{\sqrt{2}} s'' + \frac{1}{2}\right)(r_1 - r_2) \\ &= r_2 + \frac{b-a}{\sqrt{2}} s''(r_1 - r_2) + \frac{1}{2}(r_1 - r_2) \\ &= \frac{r_1 + r_2}{2} + \frac{b-a}{\sqrt{2}} s''(r_1 - r_2) \\ &= \frac{r_1 + r_2}{2} + \frac{(b-a)(r_1 - r_2)}{\sqrt{2}} \varrho\Big((b-a)^2\Big). \end{aligned}$$

□

This is how the min and max formulas emerged:

- For min: $r_1 = a, r_2 = b$.
- For max: $r_1 = b, r_2 = a$.

Another simple application of the above could be an encrypted binary query to a server. The server has two numbers encrypted, and the client wants to get one of them, but without the server knowing which one. The client sends an encrypted number to the server, indicating whether it wants to receive the first or second number from the server. Let us assume that the comparison with zero determines which one. Hence, if the client sends a positive number (which the server cannot know since it is encrypted), the server performs the above encrypted if-else procedure that returns the correct result between the two. The same thing is applied for the negative number. The server will never know the result.

### 6.2.3 Linear Algebra Operations

In this section and the following ones, we present some higher level mathematical operations that operate on encrypted data, implemented using SEAL. Those include linear algebra operations on linear algebra objects (vectors and matrices) with different methods of batching their inner elements and the FFT algorithm.

We begin with the very basic linear algebra operations:

- Elementwise operations between matrices (e.g., addition, subtraction, negation, Hadamard multiplication)
- Matrix multiplication

In the case where a matrix is a 2-dimensional structure of ciphertexts (i.e., each matrix element is a ciphertext, no batching used), it is quite easy to understand that the above operations involve only the SEAL operations that we have at our disposal.

In this ciphertext matrix representation, since each matrix element is a ciphertext, and a ciphertext contains many numbers batched, every operation on those ciphertext matrices compute at once the same operation for many batched matrices.

Elementwise operations just perform the low-level SEAL corresponding operation on each matrix element.

Matrix multiplication is just

$$(\mathbf{AB})_{i,j} = \sum_k a_{i,k} b_{k,j},$$

involving only additions and multiplications.

For square matrices, computing a matrix power is easy, it is just a special case of the matrix multiplication. But, there are repetitions that can lead to algorithmic optimizations.

For example, if we have to compute $\mathbf{A}^4$, we observe that by computing $\mathbf{A}^2$ in the first step, this result is repeated afterward ($\mathbf{A}^4 = \mathbf{A}^2\mathbf{A}^2$). This pattern leads to an algorithm like the following one, whose complexity is logarithmic to the power that the matrix is raised to.

**Algorithm 1** Compute the power of a square matrix.

**Input:** A square matrix $\mathbf{A}$, a positive integer power $n$.
**Output:** $\mathbf{A}^n$.

$N \leftarrow \lceil \lg(n+1) \rceil$ # The bit length of $n$
Let $n_0, \ldots, n_{N-1}$ be the binary representation of $n$, with $n_0$ being the LSB.
$\mathbf{R} \leftarrow \mathbf{I}$ # The result matrix
**for** $i \leftarrow 0, \ldots, N-1$ **do**
    **if** $n_i = 1$ **then**
        $\mathbf{R} \leftarrow \mathbf{RA}$
    **end if**
    $\mathbf{A} \leftarrow \mathbf{A}^2$
**end for**
**return R**

For the above algorithm, when operating on homomorphically encrypted ciphertexts, proper relinearization and rescale operations are needed.

Another useful representation of linear algebra objects (vectors and matrices) that has been implemented and used in this project is to batch many elements in a single ciphertext, taking advantage of the fact that a ciphertext encrypts many numbers. This representation can be used to save space.

Let a single ciphertext encrypting an entire vector of complex numbers. The linear algebra elementwise operations can be implemented using immediately the corresponding operations from SEAL, which act in an SIMD way, as stated before.

But, more operations that are extremely useful can be implemented if we make use of the ciphertext rotation operation from SEAL.

For example, one useful operation that has been implemented is to compute the sum of the batched elements that are within the ciphertext; in vector notation, $\mathbf{x} \mapsto \sum_i x_i$. The result of the summation, which will be a ciphertext of course, is not going to be smaller in size than the initial ciphertext. In fact, it is going to be the same size, but containing the result as its first element; the rest batched elements of this ciphertext will inevitably be there, but not used.

This batched-element sum can be used to compute norms or more complicated calculations, like this one $\sum_i x_i y_i$ for two ciphertexts $\mathbf{x}, \mathbf{y}$, which involves a ciphertext-ciphertext multiplication first.

A not so smart way to implement it is to rotate the ciphertext by 1 step and adding it to the result each time, until all the batched elements are added.

**Algorithm 2** Compute the sum of the batched elements within a ciphertext.

**Input:** A ciphertext $c$; $n$: the number of the batched elements within $c$.

**Output:** A ciphertext $c_{\text{summed}}$ containing in its first element the sum of the batched elements of $c$.

$c_{\text{summed}} \leftarrow c$
**for** $i \leftarrow 1, \ldots, n-1$ **do**
    $c_{\text{summed}} \leftarrow c_{\text{summed}} + \text{rot}_i(c)$  # Encrypted addition and ciphertext rotation
**end for**
**return** $c_{\text{summed}}$

The above algorithm used the input $n$, since the user may not have used all the batching size that a ciphertext has. The rest batched elements are there, but their values are ignored. Of course these could be 0 (an encryption of 0 to be precise), and summing them wouldn't alter the result, but these extra uneeded sums would take extra time to finish.

The above algorithm performs $n-1$ additions and $n-1$ rotations. It is linear to $n$.

A smarter way to implement it is to sum the most batched elements possible in every sum step. Let $n$ be a power of two. If we rotate $c$ by $n/2$ steps and then add it to the initial $c$ we would have summed $n/2$ elements at once.

Let us visualize this in vector notation, where for each $i$, $x_i$ is a single complex number:

$$\begin{bmatrix} x_1 \\ \vdots \\ x_{n/2} \\ x_{n/2+1} \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} x_{n/2+1} \\ \vdots \\ x_n \\ x_1 \\ \vdots \\ x_{n/2} \end{bmatrix} = \begin{bmatrix} x_1 + x_{n/2+1} \\ \vdots \\ x_{n/2} + x_n \\ \times \\ \vdots \\ \times \end{bmatrix}.$$

We do not care about the last $n/2$ elements

The next iteration will be done with $n/2$ instead of $n$.

Repeating this procedure over and over again until all the batched elements are summed, we end up having $\lg n$ iterations, i.e., $\lg n$ additions and $\lg n$ rotations.

**Algorithm 3** Smarter way for computing the sum of the batched elements within a ciphertext.

**Input:** A ciphertext $c$; $n$ (power of two): the number of the batched elements within $c$.

**Output:** A ciphertext $c_{\text{summed}}$ containing in its first batched element the sum of the batched elements of $c$.

$c_{\text{summed}} \leftarrow c$
$c_{\text{rot}} \leftarrow c$
**for** $i \leftarrow \lg n - 1, \ldots, 0$ **do**
    $c_{\text{rot}} \leftarrow \text{rot}_{2^i}(c_{\text{rot}})$
    $c_{\text{summed}} \leftarrow c_{\text{summed}} + c_{\text{rot}}$
**end for**
**return** $c_{\text{summed}}$

Of course, implementing it in practice, the above algorithm may become longer in code due to technical optimizations that can take place; that holds for almost every algorithm presented. The core idea, though, is going to be the same.

However, what if $n$ is not a power of two? Then, we run the smart algorithm described above for every power of two composing $n$ (i.e., in its binary representation), and sum those results

For example, let $n = 6 = 2^2+2^1$. Then, we run the smart algorithm for every power-of-two length subvector:

$$\mathbf{x} = \begin{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ \begin{bmatrix} x_3 \\ \vdots \\ x_6 \end{bmatrix} \end{bmatrix},$$

and then sum the intermediate results. More specifically, in this example we run the smart algorithm twice; the first time with input $(\mathbf{x}, 2)$ and store the result, the second time with input $(\text{rot}_2(\mathbf{x}), 4)$ and add the result to the previous one.

The same thinking can be applied to compute the product of the batched elements of a ciphertext, $\mathbf{x} \mapsto \prod_i x_i$.

A nice simple application (of the above batched-element sum) that has been implemented is the solution to the 2D Least Squares Linear Regression using as input two ciphertexts $\mathbf{x}, \mathbf{y}$, each containing the training data of the corresponding dimension.

Let $y = ax+b$ be our prediction model, where $a, b$ are computed using the given ciphertext data $\mathbf{x}, \mathbf{y}$ using the known formulas

$$a = \frac{n\sum_i x_i y_i - \sum_i x_i \sum_i y_i}{n\sum_i x_i^2 - \left(\sum_i x_i\right)^2},$$

$$b = \frac{\sum_i x_i^2 \sum_i y_i - \sum_i x_i \sum_i x_i y_i}{n\sum_i x_i^2 - \left(\sum_i x_i\right)^2}.$$

The server is running the above encrypted computations using the data $\mathbf{x}, \mathbf{y}$ sent from the client. After the server has computed the parameters $a, b$ (which of course are encrypted), then it can send them back to the client to decrypt them and use them however it wishes, or the server can keep them and run the encrypted model itself producing a prediction $y$ for every input $x$ via $y = ax + b$.

A very important implementation detail of the above application is that the denominator is always nonnegative (almost certainly positive in most cases).

*Proof.* From the norm isomorphism, we have that there exists $c \in \mathbb{R}$ such that for all $\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\|_1 \leq c\|\mathbf{x}\|_2$. We prove that $c = \sqrt{n}$ is one such value. Since $\|\mathbf{x}\|_1 = \mathbf{1}^T|\mathbf{x}|$, from the Cauchy-Schwartz inequality, we have

$$\left|\mathbf{1}^T|\mathbf{x}|\right| \leq \|\mathbf{1}\|_2\||\mathbf{x}|\|_2$$
$$\iff \|\mathbf{x}\|_1 \leq \sqrt{n}\|\mathbf{x}\|_2.$$

So,

$$\left(\sum_i x_i\right)^2 \leq \left(\sum_i |x_i|\right)^2 = \|\mathbf{x}\|_1^2 \leq n\|\mathbf{x}\|_2^2 = n\sum_i x_i^2$$
$$\implies n\sum_i x_i^2 - \left(\sum_i x_i\right)^2 \geq 0.$$

□

Knowing that the denominator is nonnegative can be taken advantage of, in order to compute more efficiently its inverse using immediately the inversion algorithm described above $x \mapsto x^{-1}$ with initial value $0 < a < \frac{2}{x}$. Since $x$ is not known, we can use as initial value a positive number close to 0 enough to cover a good range of possible values of the input. For example, if $a = 10^{-3}$, then the input must be $< 2 \cdot 10^3$. If one desires to be sure, then the data should be sent to the server normalized by a factor.

Another interesting representation of a linear algebra object using batching is the one of a matrix.

Whichever way we use to batch the elements of a matrix, the elementwise operations are going to be easy.

Until now we have mentioned two methods of batching the elements of linear algebra objects: for matrices (and therefore vectors that are just special cases of matrices) we have mentioned the no-batching method (every element of the matrix is a ciphertext) and for vectors we have mentioned the method of batching all its elements in a single ciphertext (if the batch size of the ciphertext is not greater than the number of the vector elements). In the last case, we also introduced an algorithm for computing the sum and the product of the first $n$ batched elements of a ciphertext (where $n$ is part of the input).

Now we are going to introduce two more batching methods for matrices and then develop algorithms for each method that perform matrix multiplication for some cases.

The first batching method that comes to mind is to batch each column or each row of a matrix. We proceed with the column batching. The row batching is very similar, and it is connected with the column batching through the transpose operation. So, the analysis for the row batching can be easily produced from the column batching analysis that follows.

Again, the elementwise operations are trivial to implement, no matter how we choose to batch the matrix elements. The hard part is the matrix multiplication. So, we have to select a batching method that enables matrix multiplication somehow.

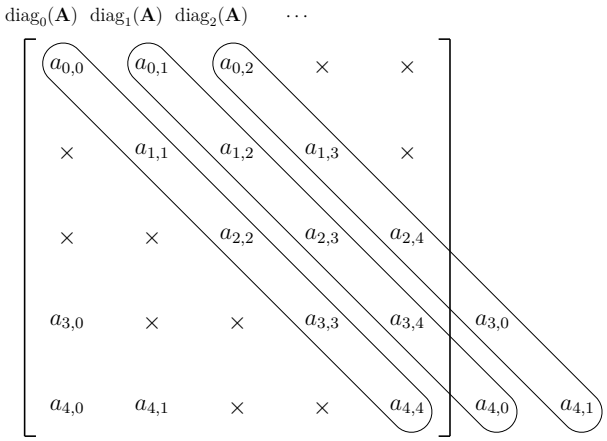



Figure 11: The diagonals of a matrix.

For the following analysis, we will use zero-indexed indices. Moreover, when an index exceeds its corresponding dimension, then it is taken modulo that dimension.

We introduce the two batching methods for a matrix. Those two batching methods both work together for the matrix multiplication.

The first batching method is the column-batching method. This method batches all the elements of each column.

The second batching method is the diagonal-batching method. This method batches all the elements of each "column diagonal". To define what a "column diagonal" is, we introduce the following operator.

**Definition 6.1.** Let $\mathbf{A} \in \mathbb{C}^{m \times m}$ and for $i, j \in \mathbb{Z}_m$ let $a_{i,j}$ be the $(i,j)$-th element of $\mathbf{A}$. For $i \in \mathbb{Z}_m$ define

$$\mathrm{diag}_i(\mathbf{A}) \triangleq \begin{bmatrix} a_{0,i} \\ a_{1,1+i} \\ \vdots \\ a_{m-1,m-1+i} \end{bmatrix}.$$

$\mathrm{diag}_i(\cdot)$ is defined only for square matrices here. The number of the diagonals of an $m \times m$ matrix is equal to $m$.

The first element of each diagonal is the first element of each column, that is why it was mentioned as "column diagonal" before.

To understand this definition better, see the example in Figure 11.

So, the diagonal-batching method batches all the elements of each diagonal, for diagonals as defined above.

For some $m, n \in \mathbb{Z}^{\geq 1}$, define:

- The set $C_{m\times n}$ that contains all the column-batched matrices in $\mathbb{C}^{m\times n}$.
- The set $C^T_{m\times n}$ that contains the transposed elements of $C$ (i.e., the $n \times m$ row-batched complex matrices).
- The set $D_{m\times m}$ that contains all the diagonal-batched matrices in $\mathbb{C}^{m\times m}$.

For notational clarity, we drop the indices in the above sets. $m, n$ will be implied implicitly as defined above.

The two matrix multiplication methods that are going to be presented have the following specifications:

- $C \times C^T \to D$
- $D \times C \to C$

If the matrix multiplications to be computed follow the above specifications, the following methods can be used.

To proceed, we introduce the following notation. For a finite sequence of elements $x_i$ for $i \in \mathbb{Z}_n$, for some $n \in \mathbb{Z}^{\geq 0}$, we denote as $[x_i]_{i=[n]}$ the vector with elements $x_0, \ldots, x_{n-1}$, in that order:

$$[x_i]_{i=[n]} \triangleq \begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix}.$$

Observe that:

- For an $m \times m$ matrix $\mathbf{A}$, $\text{diag}_k(\mathbf{A}) = [a_{i,i+k}]_{i=[m]}$.
- For an $m$-dimensional vector $\mathbf{x}$, $\text{rot}_k(\mathbf{x}) = [x_{i+k}]_{i=[m]}$.

Let us begin with the $D \times C \to C$ case.

The following lemma [9] provides a way to multiply a matrix $\mathbf{A}$ with a vector $\mathbf{x}$ using only the diagonals of $\mathbf{A}$, rotations of $\mathbf{x}$, elementwise multiplications and additions.

**Lemma 6.2.** *Let* $\mathbf{A} \in \mathbb{C}^{m\times m}$, $\mathbf{x} \in \mathbb{C}^m$. *Then*

$$\mathbf{Ax} = \sum_{k=0}^{m-1} \mathrm{diag}_k(\mathbf{A}) \odot \mathrm{rot}_k(\mathbf{x})$$

*Proof.* For $i, k \in \mathbb{Z}_m$ let $a_{i,k}$ be the $(i,k)$-th element of $\mathbf{A}$ and $x_k$ be the $k$-th element of $\mathbf{x}$. Then, we have

$$\begin{aligned}
\mathbf{Ax} &= \left[\sum_{k=0}^{m-1} a_{i,k}x_k\right]_{i=[m]} \\
&= \left[\sum_{k=0}^{m-1} a_{i,i+k}x_{i+k}\right]_{i=[m]} \qquad \text{(just a term reordering)} \\
&= \sum_{k=0}^{m-1} [a_{i,i+k}]_{i=[m]} \odot [x_{i+k}]_{i=[m]} \\
&= \sum_{k=0}^{m-1} \mathrm{diag}_k(\mathbf{A}) \odot \mathrm{rot}_k(\mathbf{x})
\end{aligned}$$

□

Now, using the above lemma, it is easy to express the $D \times C \to C$ matrix multiplication.

Let $\mathbf{A} \in D_{m\times m}$ and $\mathbf{B} \in C_{m\times n}$, with $\mathbf{b}_i$ for $i \in \mathbb{Z}_n$ being the $i$-th column of $\mathbf{B}$. Then

$$\mathbf{AB} = \begin{bmatrix}\mathbf{Ab}_0 & \cdots & \mathbf{Ab}_{n-1}\end{bmatrix},$$

where, for the $i$-th column of the result matrix, we have

$$\mathbf{Ab}_i = \sum_{k=0}^{m-1} \mathrm{diag}_k(\mathbf{A}) \odot \mathrm{rot}_k(\mathbf{b}_i).$$

Finally, for the $C \times C^T \to D$ case, we just use the known matrix multiplication formula.

Let $\mathbf{A}, \mathbf{B} \in C_{m\times n}$. This means that $\mathbf{B}^T \in C^T_{m\times n}$.

For $i, j \in \mathbb{Z}_m$, $k \in \mathbb{Z}_n$ and

$$\mathbf{C} = \mathbf{AB}^T,$$

let $c_{i,j}$ be the $(i,j)$-th element of $\mathbf{C}$, $a_{i,k}$ be the $(i,k)$-th element of $\mathbf{A}$ and $b_{j,k}$ be the $(j,k)$-th element of $\mathbf{B}$. Then,

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k}b_{j,k}.$$

For $\ell \in \mathbb{Z}_m$, let $j = i + \ell$, so, we have

$$\begin{aligned}\mathrm{diag}_\ell(\mathbf{C}) &= [c_{i,i+\ell}]_{i=[m]} \\ &= \left[\sum_{k=0}^{n-1} a_{i,k} b_{i+\ell,k}\right]_{i=[m]} \\ &= \sum_{k=0}^{n-1} [a_{i,k}]_{i=[m]} \odot [b_{i+\ell,k}]_{i=[m]} \\ &= \sum_{k=0}^{n-1} \mathbf{a}_k \odot \mathrm{rot}_\ell(\mathbf{b}_k),\end{aligned}$$

where $\mathbf{a}_k$, $\mathbf{b}_k$ are the $k$-th columns of $\mathbf{A}, \mathbf{B}$, respectively.

To sum up,

- For the case $D \times C \to C$, for $\mathbf{C} = \mathbf{AB}$, $i \in \mathbb{Z}_n$:

$$\mathbf{c}_i = \sum_{k=0}^{m-1} \mathrm{diag}_k(\mathbf{A}) \odot \mathrm{rot}_k(\mathbf{b}_i).$$

- For the case $C \times C^T \to D$, for $\mathbf{C} = \mathbf{AB}^T$, $i \in \mathbb{Z}_m$:

$$\mathrm{diag}_i(\mathbf{C}) = \sum_{k=0}^{n-1} \mathbf{a}_k \odot \mathrm{rot}_i(\mathbf{b}_k).$$

Thus, we end up with the following incredibly simple algorithms.

**Algorithm 4** $D \times C \to C$ encrypted matrix multiplication

**Input:** A ciphertext list $(a_0, \ldots, a_{m-1})$, with each ciphertext encrypting a diagonal of the left operand matrix $\mathbf{A}$; a ciphertext list $(b_0, \ldots, b_{n-1})$, with each ciphertext encrypting a column of the right operand matrix $\mathbf{B}$.

**Output:** A ciphertext list $(c_0, \ldots, c_{n-1})$, with each ciphertext encrypting a column of the result matrix $\mathbf{C} = \mathbf{AB}$.

**for** $i \leftarrow 0, \ldots, n-1$ **do**
  $c_i \leftarrow a_0 \cdot b_i$
  **for** $k \leftarrow 1, \ldots, m-1$ **do**
    $c_i \leftarrow c_i + a_k \cdot \mathrm{rot}_k(b_i)$
  **end for**
  # Relinearize and rescale $c_i$ here.
**end for**
**return** $(c_0, \ldots, c_{n-1})$

**Algorithm 5** $C \times C^T \to D$ encrypted matrix multiplication

**Input:** A ciphertext list $(a_0, \ldots, a_{n-1})$, with each ciphertext encrypting a column of the left operand matrix $\mathbf{A}$; a ciphertext list $(b_0, \ldots, b_{n-1})$, with each ciphertext encrypting a column of the right operand matrix $\mathbf{B}$.

**Output:** A ciphertext list $(c_0, \ldots, c_{m-1})$, with each ciphertext encrypting a diagonal of the result matrix $\mathbf{C} = \mathbf{A}\mathbf{B}^T$.

**for** $i \leftarrow 0, \ldots, m-1$ **do**
  $c_i \leftarrow a_0 \cdot \mathrm{rot}_i(b_0)$
  **for** $k \leftarrow 1, \ldots, n-1$ **do**
    $c_i \leftarrow c_i + a_k \cdot \mathrm{rot}_i(b_k)$
  **end for**
  # Relinearize and rescale $c_i$ here.
**end for**
**return** $(c_0, \ldots, c_{n-1})$

The above algorithms perform ciphertext-ciphertext multiplications, so, both relinearization and rescaling are needed. Usually, those operations are done right after the multiplication takes place. However, since relinearization has a computational cost that should be taken into account, doing it outside the inner for loop on the final sum result will make the above algorithm a lot more performant.

The batch size of a ciphertext is configured at the beginning during the cryptosystem parameter setting. Thus, most probably there will be cases where the number of elements that one may desire to batch may be greater or less than the batch size. In the case that the batch size is bigger, one can pad with zeros until it fits the batch size.

For example, in the $C \times C^T \to D$ case, let $\mathbf{A}, \mathbf{B} \in C_{m' \times n}$, where $m' < m$, with $m$ being the batch size. Then, one can encrypt the matrices (let $m'' = m - m'$)

$$\mathbf{A}' = \begin{bmatrix} \mathbf{A} \\ \mathbf{0}_{m'' \times n} \end{bmatrix} \quad , \quad \mathbf{B}' = \begin{bmatrix} \mathbf{B} \\ \mathbf{0}_{m'' \times n} \end{bmatrix}$$

where $\mathbf{A}', \mathbf{B}' \in C_{m \times n}$, and

$$\begin{aligned} \mathbf{A}'\mathbf{B}'^T &= \begin{bmatrix} \mathbf{A} \\ \mathbf{0}_{m'' \times n} \end{bmatrix} \begin{bmatrix} \mathbf{B}^T & (\mathbf{0}_{m'' \times n})^T \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{A}\mathbf{B}^T & \mathbf{0}_{m' \times m''} \\ \mathbf{0}_{m'' \times m'} & \mathbf{0}_{m'' \times m''} \end{bmatrix} \end{aligned}$$

where the result matrix is $m \times m$, but the result that one is interested in is in the upper left $m' \times m'$ submatrix, while the rest elements are just 0.

One more thing is the following one. What if, in the $D \times C \to C$ case, one did not want the left operand matrix to be a square matrix? Then one can apply the same zero-padding technique until it becomes a square matrix and meet the specifications, and then find the desired result in a submatrix.

In the opposite case, now, where the batch size if smaller, then one may need more ciphertexts to store the elements that would be in a single batch, and then use block matrices with the previously described batching methods to achieve the desired result.

Of course, more operations can be enabled on batched elements if we use the masking multiplication and rotations. Let $\mathbf{x}$ be a vector of elements and assume that we would like to isolate the $i$-th element of it to perform an operation. We can compute the Hadamard product with the appropriate mask, in this case $\mathbf{e}_i$, which is a vector with 1 in its $i$-th position and 0 elsewhere. Then, if we want it to be the first element of the resulting vector, we can rotate left by $i$ steps the last result. Algebraically expressed, $\text{rot}_i(\mathbf{x} \odot \mathbf{e}_i)$. However, this technique comes with a greater performance cost if used extensively.

### 6.2.4 Fast Fourier Transform (FFT)

Another interesting idea to implement using homomorphic encryption is the DFT over encrypted data.

Using zero-indexed indices.

For a finite sequence of $n$ complex numbers $x_0, \ldots, x_{n-1} \in \mathbb{C}$, its DFT is the finite sequence $y_0, \ldots, y_{n-1} \in \mathbb{C}$ of the same length defined as

$$y_m \triangleq \sum_{k=0}^{n-1} x_k \omega_n^{km}, \text{ for } m \in \mathbb{Z}_n,$$

where $\omega_n \triangleq e^{-i2\pi/n}$.

For the above formula, representing the above sequences using the vectors $\mathbf{x}, \mathbf{y}$, the above formula can be expressed via the DFT operator this way: $\mathbf{y} = \text{DFT}_n(\mathbf{x}) = \text{DFT}(\mathbf{x})$ (dropping the length index $n$ since it can be inferred by the input vector).

For $n$ being a power of two, the radix-2 Cooley-Tukey FFT algorithm emerges as follows:

$$\begin{aligned} y_m &= \sum_{k=0}^{n-1} x_k \omega_n^{km} \qquad \text{for } m \in \mathbb{Z}_n \\ &= \sum_{k=0}^{n/2-1} x_{2k} \omega_n^{2km} + \sum_{k=0}^{n/2-1} x_{2k+1} \omega_n^{(2k+1)m} \qquad \text{for } m \in \mathbb{Z}_n \\ &= \sum_{k=0}^{n/2-1} x_{2k} \underbrace{\omega_n^{2km}}_{\omega_{n/2}^{km}} + \omega_n^m \sum_{k=0}^{n/2-1} x_{2k+1} \underbrace{\omega_n^{2km}}_{\omega_{n/2}^{km}} \qquad \text{for } m \in \mathbb{Z}_n \end{aligned}$$

$$\iff \mathbf{y} = \mathrm{DFT}(\mathbf{x}) = \begin{bmatrix} \mathrm{DFT}(\mathbf{x}_\mathrm{e}) \\ \mathrm{DFT}(\mathbf{x}_\mathrm{e}) \end{bmatrix} + \boldsymbol{\omega}_n \odot \begin{bmatrix} \mathrm{DFT}(\mathbf{x}_\mathrm{o}) \\ \mathrm{DFT}(\mathbf{x}_\mathrm{o}) \end{bmatrix}, \tag{6.3}$$

where $\boldsymbol{\omega}_n = \begin{bmatrix} \omega_n^0 & \cdots & \omega_n^{n-1} \end{bmatrix}^T$, $\mathbf{x}_\mathrm{e}$ vector contains the $x_{2k}$ values (even indices) and $\mathbf{x}_\mathrm{o}$ contains the $x_{2k+1}$ values (odd indices).

For the above vectors that repeat their first half elements, this happens since for all $k \in \mathbb{Z}_n$ the $\omega_{n/2}^{km}$ values are repeated for $n/2 \leq m \leq n-1$.

In addition, the fact that the last $n/2$ elements of $\boldsymbol{\omega}_n$ are just the negation of the first $n/2$ elements of it can be leveraged.

It is very clean that it can be implemented using our homomorphic encryption library SEAL: it uses just additions and multiplications.

The twiddle factors $\boldsymbol{\omega}_n$ don't have to be encrypted, making some multiplications operate between plaintext and ciphertexts, which are faster than ciphertext-ciphertext multiplications.

If we represent every element of the input sequence as a ciphertext (i.e., without batching the input elements), we can implement the recursive FFT algorithm as described above.

**Algorithm 6** Radix-2 FFT

**Input:** Ciphertexts $c_0, \ldots, c_{n-1}$.

**Output:** DFT of the input sequence.

**if** $n = 1$ **then**
  **return** $c_0$
**end if**
$\left(\tilde{c}_0^{(\mathrm{e})}, \ldots, \tilde{c}_{n/2-1}^{(\mathrm{e})}\right) \leftarrow$ Recursive call with input $(c_0, c_2, \ldots, c_{n-2})$
$\left(\tilde{c}_0^{(\mathrm{o})}, \ldots, \tilde{c}_{n/2-1}^{(\mathrm{o})}\right) \leftarrow$ Recursive call with input $(c_1, c_3, \ldots, c_{n-1})$
**for** $k \leftarrow 0, \ldots, n/2-1$ **do**
  $\tilde{c}_k \leftarrow \tilde{c}_k^{(\mathrm{e})} + \omega_n^k \cdot \tilde{c}_k^{(\mathrm{o})}$
  $\tilde{c}_{k+n/2} \leftarrow \tilde{c}_k^{(\mathrm{e})} + \omega_n^k \cdot \tilde{c}_k^{(\mathrm{o})}$
**end for**
**return** $(\tilde{c}_0, \ldots, \tilde{c}_{n-1})$

The above algorithm, for clarity, omits the appropriate rescale operations needed after encrypted multiplications and extra dummy operations for chain level matching between the terms that are added, but these should be done. Since, only plaintext-ciphertext multiplications are involved, no relinearization is needed.

The above algorithm was a simple FFT implementation for computing the DFT of a sequence of encrypted numbers, each number represented by a ciphertext (i.e., no batched input). Of course, a ciphertext can contain many batched elements, so, the above algorithm can perform ciphertext-batch-size FFTs in a single run.

Now, what if one desires to batch the input in a single ciphertext in order to save space? Can we leverage the ciphertext rotation operation efficiently to make it work in this case?

DFT can be represented in matrix form:

$$\mathbf{y} = \mathbf{F}_n \mathbf{x}$$

where $\mathbf{F}_n \in \mathbb{C}^{n \times n}$,

$$\mathbf{F}_n = \left[\omega_n^{km}\right]_{\substack{k=0,\ldots,n-1 \\ m=0,\ldots,n-1}} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \omega_n^3 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \cdots & \omega_n^{2(n-1)} \\ 1 & \omega_n^3 & \omega_n^6 & \omega_n^9 & \cdots & \omega_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \omega_n^{3(n-1)} & \cdots & \omega_n^{(n-1)(n-1)}. \end{bmatrix}$$

Revisiting the formula (6.3), we have

$$\begin{aligned}
\mathrm{DFT}(\mathbf{x}) &= \begin{bmatrix}\mathrm{DFT}(\mathbf{x}_\mathrm{e})\\ \mathrm{DFT}(\mathbf{x}_\mathrm{e})\end{bmatrix} + \boldsymbol{\omega}_n \odot \begin{bmatrix}\mathrm{DFT}(\mathbf{x}_\mathrm{o})\\ \mathrm{DFT}(\mathbf{x}_\mathrm{o})\end{bmatrix} \\
\iff \quad \mathbf{F}_n\mathbf{x} &= \begin{bmatrix}\mathbf{F}_{n/2}\mathbf{x}_\mathrm{e}\\ \mathbf{F}_{n/2}\mathbf{x}_\mathrm{e}\end{bmatrix} + \boldsymbol{\omega}_n \odot \begin{bmatrix}\mathbf{F}_{n/2}\mathbf{x}_\mathrm{o}\\ \mathbf{F}_{n/2}\mathbf{x}_\mathrm{o}\end{bmatrix} \\
&= \begin{bmatrix}\mathbf{F}_{n/2}\mathbf{x}_\mathrm{e}\\ \mathbf{F}_{n/2}\mathbf{x}_\mathrm{e}\end{bmatrix} + \begin{bmatrix}\omega_n^0 & & \\ & \ddots & \\ & & \omega_n^{n-1}\end{bmatrix}\begin{bmatrix}\mathbf{F}_{n/2}\mathbf{x}_\mathrm{o}\\ \mathbf{F}_{n/2}\mathbf{x}_\mathrm{o}\end{bmatrix} \\
&= \begin{bmatrix}\mathbf{F}_{n/2}\\ \mathbf{F}_{n/2}\end{bmatrix}\mathbf{x}_\mathrm{e} + \begin{bmatrix}\mathbf{W}_{n/2} & \mathbf{0}_{n/2}\\ \mathbf{0}_{n/2} & -\mathbf{W}_{n/2}\end{bmatrix}\begin{bmatrix}\mathbf{F}_{n/2}\\ \mathbf{F}_{n/2}\end{bmatrix}\mathbf{x}_\mathrm{o} \\
&= \begin{bmatrix}\mathbf{F}_{n/2}\\ \mathbf{F}_{n/2}\end{bmatrix}\mathbf{x}_\mathrm{e} + \begin{bmatrix}\mathbf{W}_{n/2}\mathbf{F}_{n/2}\\ -\mathbf{W}_{n/2}\mathbf{F}_{n/2}\end{bmatrix}\mathbf{x}_\mathrm{o} \\
&= \begin{bmatrix}\mathbf{F}_{n/2} & \mathbf{W}_{n/2}\mathbf{F}_{n/2}\\ \mathbf{F}_{n/2} & -\mathbf{W}_{n/2}\mathbf{F}_{n/2}\end{bmatrix}\begin{bmatrix}\mathbf{x}_\mathrm{e}\\ \mathbf{x}_\mathrm{o}\end{bmatrix} \\
&= \begin{bmatrix}\mathbf{I}_{n/2} & \mathbf{W}_{n/2}\\ \mathbf{I}_{n/2} & -\mathbf{W}_{n/2}\end{bmatrix}\begin{bmatrix}\mathbf{F}_{n/2} & \mathbf{0}_{n/2}\\ \mathbf{0}_{n/2} & \mathbf{F}_{n/2}\end{bmatrix}\begin{bmatrix}\mathbf{x}_\mathrm{e}\\ \mathbf{x}_\mathrm{o}\end{bmatrix},
\end{aligned}$$

where $\mathbf{0}_{n/2}$ is the $(n/2)\times(n/2)$ zero matrix, and $\mathbf{W}_{n/2}$ is the $(n/2)\times(n/2)$ matrix with the consequent powers of $\omega_n$ in its diagonals and the rest elements 0, i.e.,

$$\mathbf{W}_{n/2} = \begin{bmatrix}\omega_n^0 & & \\ & \ddots & \\ & & \omega_n^{n/2-1}\end{bmatrix}.$$

Due to the way that divide-and-conquer works in the DFT formula, it is observed that the elements of the vector $\mathbf{x}$ are reordered. Continuing the factorization in the last matrix equation, we get that the elements of the vector $\mathbf{x}$ will be ordered in the bit-reversed order.

The following algorithm was introduced in [9].

Denoting as $\mathbf{F}_n^{\mathrm{BR}}$ the $n\times n$ DFT matrix with input length $n$ and bit-reversed output, the following factorization of $\mathbf{F}_n^{\mathrm{BR}}$ emerges:

$$\mathbf{F}_n^{\mathsf{BR}} = \begin{bmatrix}\mathbf{F}_{n/2}^{\mathsf{BR}} & \mathbf{F}_{n/2}^{\mathsf{BR}}\\ \mathbf{F}_{n/2}^{\mathsf{BR}}\mathbf{W}_{n/2} & -\mathbf{F}_{n/2}^{\mathsf{BR}}\mathbf{W}_{n/2}\end{bmatrix} = \begin{bmatrix}\mathbf{F}_{n/2}^{\mathsf{BR}} & \mathbf{0}\\ \mathbf{0} & \mathbf{F}_{n/2}^{\mathsf{BR}}\end{bmatrix}\begin{bmatrix}\mathbf{I}_{n/2} & \mathbf{I}_{n/2}\\ \mathbf{W}_{n/2} & -\mathbf{W}_{n/2}\end{bmatrix},$$

and, if we keep going, we get

$$\mathbf{F}_n^{\mathsf{BR}} = \mathbf{D}_n^{(n)}\mathbf{D}_{n/2}^{(n)}\cdots\mathbf{D}_2^{(n)},$$

where, for $k \in \{n, n/2, \ldots, 2\}$:

$$\mathbf{D}_k^{(n)} = \begin{bmatrix} \begin{bmatrix} \mathbf{I}_{n/k} & \mathbf{I}_{n/k} \\ \mathbf{W}_{n/k} & -\mathbf{W}_{n/k} \end{bmatrix} & \mathbf{0}_{2n/k} & \cdots & \mathbf{0}_{2n/k} \\ \mathbf{0}_{2n/k} & \begin{bmatrix} \mathbf{I}_{n/k} & \mathbf{I}_{n/k} \\ \mathbf{W}_{n/k} & -\mathbf{W}_{n/k} \end{bmatrix} & \cdots & \mathbf{0}_{2n/k} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_{2n/k} & \mathbf{0}_{2n/k} & \cdots & \begin{bmatrix} \mathbf{I}_{n/k} & \mathbf{I}_{n/k} \\ \mathbf{W}_{n/k} & -\mathbf{W}_{n/k} \end{bmatrix} \end{bmatrix} \in \mathbb{C}^{n \times n},$$

with the number of the diagonal submatrices being $k/2$.

From the above structure of $\mathbf{D}_k^{(n)}$ it can be understood that for $i \in \{1, \ldots, \lg n\}$ the vector $\mathrm{diag}_k\left(\mathbf{D}_{2^i}^{(n)}\right)$ is nonzero only for $k \in \{0, n/2^i, n - n/2^i\}$.

Using the Lemma 6.2 and the above observation, we have that for a vector $\mathbf{v} \in \mathbb{C}^n$:

$$\mathbf{D}_{2^i}^{(n)}\mathbf{v} = \sum_{k \in \{0, n/2^i, n-n/2^i\}} \mathrm{diag}_k\left(\mathbf{D}_{2^i}^{(n)}\right) \odot \mathrm{rot}_k(\mathbf{v})$$

For $i = 1$ (since $n/2 = n - n/2$ the sum has only two terms) we have

$$\mathbf{D}_2^{(n)}\mathbf{v} = \mathrm{diag}_0\left(\mathbf{D}_2^{(n)}\right) \odot \mathbf{v} + \mathrm{diag}_{n/2}\left(\mathbf{D}_2^{(n)}\right) \odot \mathrm{rot}_{n/2}(\mathbf{v}),$$

and for $i \neq 1$ we have

$$\begin{aligned} \mathbf{D}_{2^i}^{(n)}\mathbf{v} = \mathrm{diag}_0\left(\mathbf{D}_{2^i}^{(n)}\right) \odot \mathbf{v} &+ \mathrm{diag}_{n/2^i}\left(\mathbf{D}_{2^i}^{(n)}\right) \odot \mathrm{rot}_{n/2^i}(\mathbf{v}) \\ &+ \mathrm{diag}_{n-n/2^i}\left(\mathbf{D}_{2^i}^{(n)}\right) \odot \mathrm{rot}_{n-n/2^i}(\mathbf{v}). \end{aligned}$$

Making use just those last formulas, the algorithm for the encrypted DFT with batched input can be implemented.

**Algorithm 7** Encrypted batched-input DFT

**Input:** Ciphertext $c$; the power-of-two length $n$ of the batched elements.

**Output:** DFT of the batched-element sequence in $c$.

**for** $i \leftarrow 1, \ldots, \lg n$ **do**
  $c_0 \leftarrow c \cdot \mathrm{diag}_0\left(\mathbf{D}_{2^i}^{(n)}\right)$
  $c_1 \leftarrow \mathrm{rot}_{n/2^i}(c)$
  $c_1 \leftarrow c_1 \cdot \mathrm{diag}_{n/2^i}\left(\mathbf{D}_{2^i}^{(n)}\right)$
  **if** $i = 1$ **then**
    $c \leftarrow c_0 + c_1$
  **else**
    $c_2 \leftarrow \mathrm{rot}_{-n/2^i}(c)$
    $c_2 \leftarrow c_2 \cdot \mathrm{diag}_{n-n/2^i}\left(\mathbf{D}_{2^i}^{(n)}\right)$
    $c \leftarrow c_0 + c_1 + c_2$
  **end if**
**end for**
**return** $c$

Again, the $\mathbf{D}_k^{(n)}$ matrices do not need to be encrypted, so, the multiplications involving them are just plaintext-ciphertext multiplications. Hence, no relinearization is needed.

And, again, the above algorithm omits the appropriate rescale operations needed after the encrypted multiplications.

The input length $n$ could be smaller than the ciphertext batch size. In that case, the batched elements must be stored in a ciphertext in a way that the rotations (especially the right rotations) work seamlessly. For example, the elements could be repeated until they reach the batch size; this works without any elements be left over, since both the ciphertext batch size and $n$ are powers of two.

An interesting application of the above encrypted DFT algorithms is the encrypted frequency filtering. A simple thought is the following. Having an encrypted Fourier transform of a sequence, one can compute the complex conjugate of its elements and then multiply with the initial data. Multiplying a complex number with its conjugate results to its absolute value squared. Hence, we just computed the power spectral density, and it is still encrypted. using the encrypted if-else presented above, one can filter out the undesired frequencies of it.

### 6.2.5 Benchmarks

Tested on Intel Core i7-7700HQ.

The batch size is equal to $n/2$, where $n$ is the polynomial degree, as shown in the high-level schematic in Figure 10.

The following are the results of operation timings for various parameters of the CKKS cryptosystem.

Note the scale factor in y-axis sometimes.

The following measurements change the batch size, keeping the ciphertext coefficient modulus constant (and low).

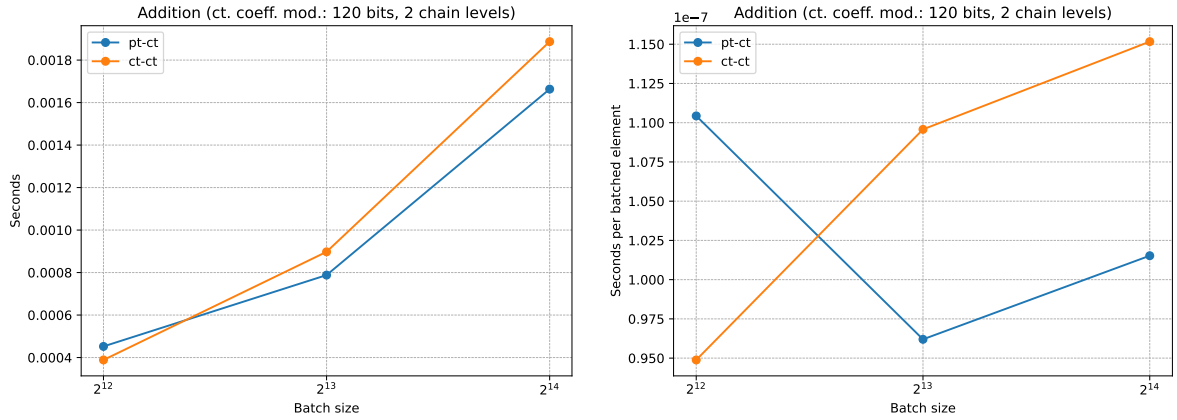



Figure 12: Addition on different batch sizes.

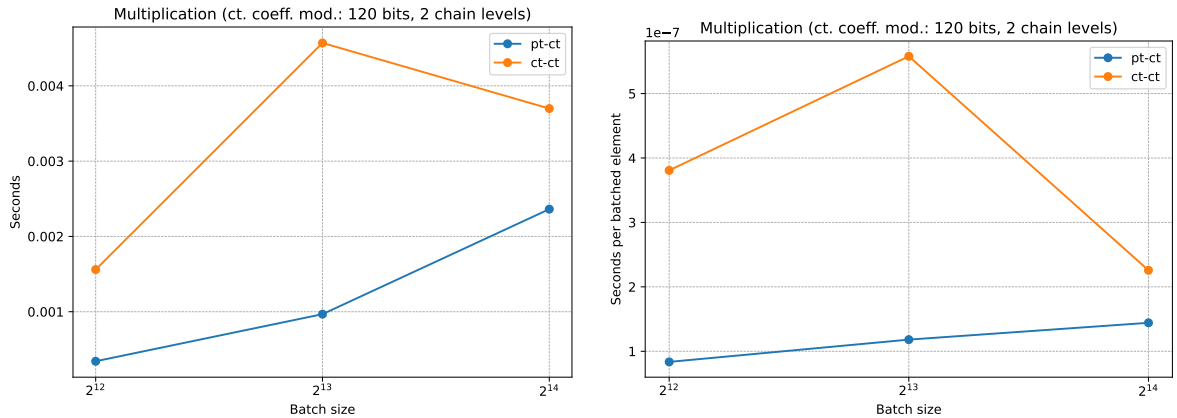



Figure 13: Multiplication on different batch sizes.

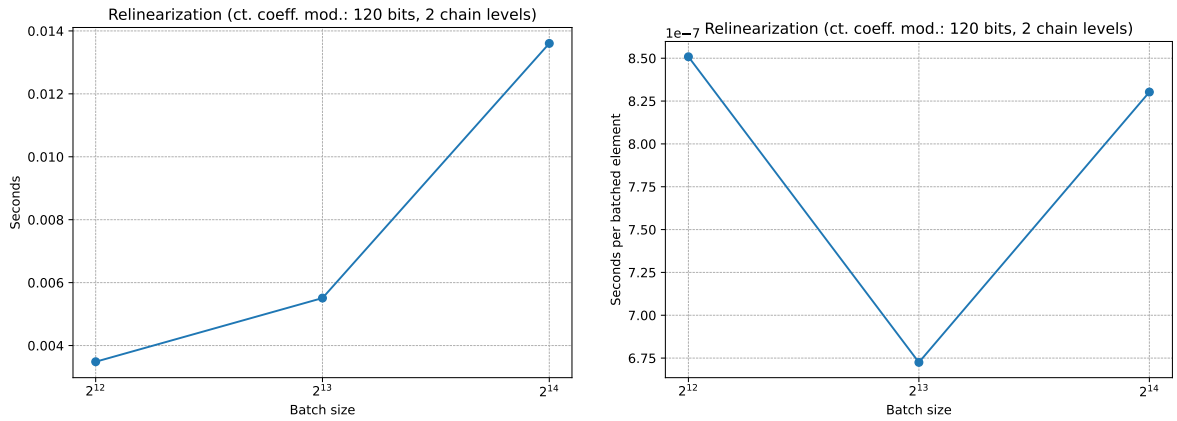



Figure 14: Relinearization on different batch sizes.

Generally, the ciphertext coefficient modulus is low. Only 2 chain levels are there.

Addition and multiplication are pretty fast. If we also can leverage batching, then we have even better results looking the per batched element time.

Notice how relinearization is somewhat computationally costly compared to the above timings. So, it should be used smartly whenever possible.

The following measurements change ciphertext coefficient modulus, keeping the batch size constant.

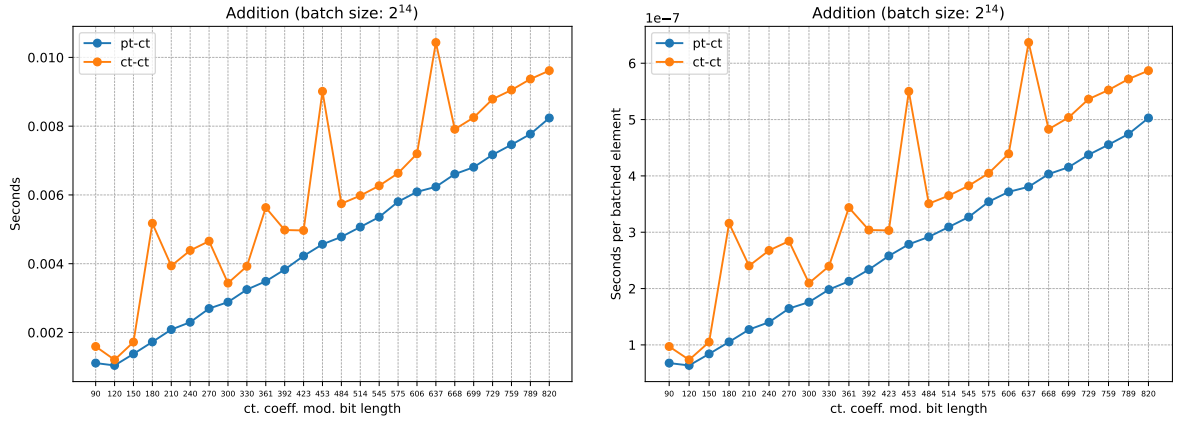



Figure 15: Addition on different number of chain levels.

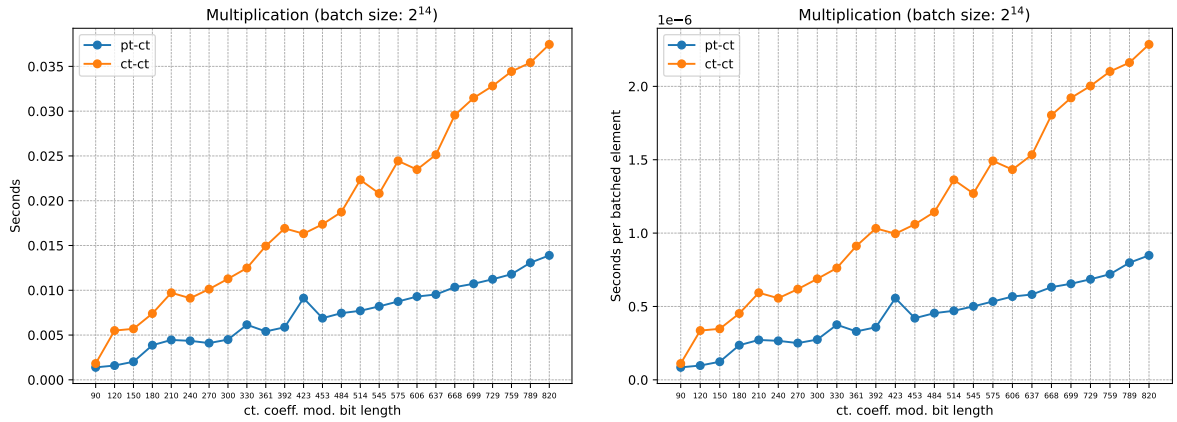



Figure 16: Multiplication on different number of chain levels.

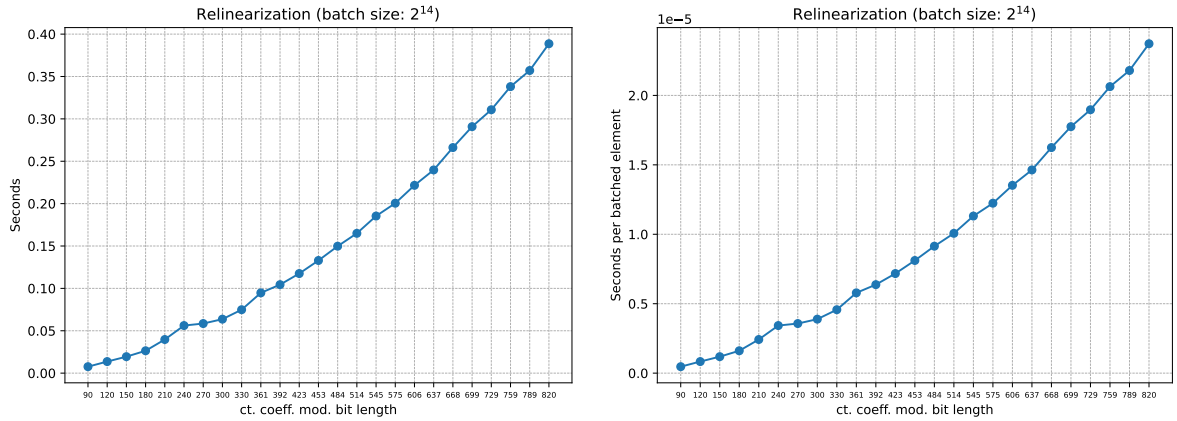



Figure 17: Relinearization on different number of chain levels.

Again, batching can be leveraged. And, again, it is evident the computational cost of relinearization.

Next, we present measurements taken for the FFT. Ignore the cryptosystem parameters. The purpose here is to compare the FFT non-batched vs. batched versions, and show how much can things be improved if we leverage batching.

Encrypted FFT time (32 elements): 3.00907 s

Encrypted batched FFT time (32 elements): 0.542291 s

Encrypted FFT time (64 elements): 6.70999 s

Encrypted batched FFT time (64 elements): 0.625865 s

Encrypted FFT time (128 elements): 14.7536 s

Encrypted batched FFT time (128 elements): 0.681758 s

Similar are the results between non-batched and batched matrices.

One more thing to highlight, is to be careful in the implementation of matrix multiplication (in any variant, batched or non-batched), is not to perform relinearization in every internal multiplication, but perform the sum of the products and then perform the relinearization on the result. The time difference is huge.

# 7 References

# References

[1] J. Katz and Y. Lindell, *Introduction to Modern Cryptography: Principles and Protocols*, 2nd. Boca Raton, FL: CRC Press, 2014.

[2] O. Regev, *Introduction*, Accessed: 2024-9-14, 2009. [Online]. Available: https://cims.nyu.edu/~regev/teaching/lattices_fall_2004/ln/introduction.pdf.

[3] O. Regev, *Dual lattices*, Accessed: 2024-9-14, 2009. [Online]. Available: https://cims.nyu.edu/~regev/teaching/lattices_fall_2004/ln/DualLattice.pdf.

[4] C. Gentry, "A fully homomorphic encryption scheme," PhD Thesis, Ph.D. dissertation, Stanford University, 2009. [Online]. Available: https://crypto.stanford.edu/craig/craig-thesis.pdf.

[5] C. Peikert, *A decade of lattice cryptography*, Cryptology ePrint Archive, Paper 2015/939, 2015. [Online]. Available: https://eprint.iacr.org/2015/939.

[6] M. Ajtai, "Generating hard instances of lattice problems," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing (STOC)*, ACM, 1996, pp. 99–108. DOI: 10.1145/237814.237838.

[7] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM*, 2009. DOI: 10.1145/1568318.1568324.

[8] *Microsoft SEAL (release 4.1)*, https://github.com/Microsoft/SEAL, Microsoft Research, Redmond, WA., Jan. 2023.

[9] J. H. Cheon, K. Han, and M. Hhan, *Faster homomorphic discrete fourier transforms and improved FHE bootstrapping*, Cryptology ePrint Archive, Paper 2018/1073, 2018. [Online]. Available: https://eprint.iacr.org/2018/1073.