On the Post-Quantum Understanding of Lattice Problems

Furkan Kuz

MSc Thesis
May 2025

DEPARTMENT OF MATHEMATICS AND STATISTICS

Supervisor: Dr. Mika Hirvensalo
Examiner: Dr. Ville Junnila

UNIVERSITY OF TURKU
Department of Mathematics and Statistics

Furkan Kuz: On the Post-Quantum Understanding of Lattice Problems

MSc Thesis 97 Pages, 32 appendix pages.
Mathematics
May 2025

This thesis explores key literature on lattice-based post-quantum cryptography and its elements.

Chapter 2 introduces key computational concepts, including complexity theory and various models of computation, to establish the theoretical basis for cryptographic security.

Chapter 3 provides an overview of classical cryptographic techniques, setting the stage for the transition to quantum-resistant approaches.

Chapters 4 and 5 investigate lattice theory and its associated hard problems.

Chapter 6 focuses on classical algorithms for lattice reduction that are relevant to both analyzing and solving lattice-based cryptographic problems.

Appendices A and B supplement the main content with relevant mathematical background and introductory concepts in quantum computing.

Keywords: lattice cryptography, computational complexity, post-quantum cryptography, quantum computation.

Contents

1 Introduction

Throughout the existence of humankind, the process of creating and preserving information has been indispensable across all epochs. Although some sources claim that the earliest instances of obscured or enigmatic writings can be traced back to Egypt two millennia prior to the Common Era [1] appearing as hieroglyphic substitution[1], investigating and tracing these approaches pose significant difficulties. In the present chapter, an analysis will be pursued on versions of these approaches that can be denoted in mathematical thinking.

There is no doubt that no field of science is limited solely to its scope. However, cryptography consistently unveils its crucial nature whenever significant events unfold across various forms such as religion, art, love, death and war. Prior to exploring all of these, we will make an effort to understand a few central terms.

Cryptography: The science and art of techniques to protect the security when transferring information. Effective information transfer must prioritize confidentiality to restrict access to sensitive data, integrity to ensure that the data remain unaltered and trustworthy and availability to provide authorized users with timely access to the information. Etymology of the word comes from Greek as a combination of the words, *cryptos* and *graphein* which somehow can be translated as secret-writing [2].

Encryption: Process of transforming plaintext (*m:* message) into unintelligible format (*c:* ciphertext) by applying mathematical functions or mappings with special encryption key(E_k).

$$c = E_k(m)$$

Decryption: Reverse process of encryption where D_k is decryption key to converse ciphertext to message.

$$m = D_k(c)$$

Cipher: Algorithm or steps listed to execute encryption and decryption. It is a word of Arabic origin (*sifr*) that exactly corresponds to number 0 where it was to be translated by Fibonacci later on. Due to its ambiguous and mysterious existence for nothingness and being a concept introduced at later stages of mathematical history [3], it was employed to label the unintelligibility. Occasionally, it is used as a parallel term to encryption, while *deciphering* is employed as its counterpart for decryption. Since these terms are explained briefly, we can now explore how they caused significant changes in human history.

Scytale

In Ancient Greece, Scytale was used as form of secret messaging commonly adapted between soldiers and messengers of the Greek Army. For encryption purposes, messengers were wrapping their belts around a cylindrical object and writing the message in horizontal form which will result in meaningless vertical text. The object

[1] Regardless of its lack of cryptographic intentions

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Figure 1: Numerical Encodings (in alphabetical order) of Letters [4]

used in this process is also called the Scytale E_k, a non-unique physical key where $E_k = D_k$.

Atbash Cipher

One of the first ciphers that has been used with Hebrew Alphabet and handed down to generations through the Old Testament from 5th century BCE. Atbash is an example for one of the simplest monoalphabethic substitution ciphers, that is also known as reverse alphabet where E_k is given below. Since we can not use mathematical operations on letters, we need their numerical order encoding.

We will be using English alphabet, which comprises 26 characters, with numbers encoding. For the sake of simplicity, within this framework, $n = 26$ refers to the total quantity of distinct characters constituting the alphabet.

$$E(x) = (26 - (x - 1)) \mod 26$$

By its naive nature Atbash Cipher uses the same function for decryption. Its security is extremely limited due to small key size.

$$D(x) = E(x)$$

The Caesar Cipher

Another important occurrence of cryptography during the 1st century BCE originates from Ancient Rome, and it has been utilized for military purposes. The Caesar Cipher also known as **Shift Cipher**. While key of the Caesar Cipher can be any element of the given alphabet $\mathbb{Z}_{26}$, it is believed that Julius Caesar utilized this cryptographic technique with a key value of $E_k = D_k = k = 3$. Hence, it is attributed to Roman statesman Julius Caesar. Table one shows plaintext and corresponding ciphertext.

$$E(x) = (x + k) \pmod{26}$$

$$D(y) = (y - k) \pmod{26}$$

U	N	I	V	E	R	S	I	T	Y	O	F	T	U	R	K	U
X	Q	L	Y	H	U	V	L	W	B	R	I	W	X	U	N	X

Table 1: Message encrypted by Caesar Cipher

Letter Frequency and Golden Ages

In the mid-ninth century, within an intellectually vibrant period of the Islamic world characterized by state-supported pursuits in science and philosophy, al-Kindi introduced the earliest documented method of statistical cryptanalysis [3]. In his book, *A Manuscript on Deciphering Cryptographic Messages*, he explains how to use cryptanalysis on an encrypted message, if another plaintext from the same language is available.

This approach revealed the major weakness of the substitution ciphers against to cryptanalysis based on letter frequency, where each letter of plaintext always maps to the same letter in ciphertext. Centuries later, *Golden Age of England* with long period of peace and prosperity, the fate of England were about to experience the impact of the letter frequencies and cryptanalysis. During her imprisonment Mary, Queen of Scots was using a method for nomenclature and substitution cipher for secret communications. Queen Mary thought her messages were safe, while Thomas Phelippes was, in fact, intercepting and copying her correspondence. Due to nature of this weak encryption, Queen Mary expressed her desires openly and freely in writing, according to her will. This narrative culminated in a cryptanalyst deciphering these writings, leading to their conviction and subsequent execution, including even Queen Mary herself.

Enigma vs. Bombe

In the aftermath of World War I Arthur Scherbius invented a machine that allows secure communication with the aid of cryptographic techniques [5]. Physical composition of the device was similar to a typewriter. Upon the user's keystroke, the other keyboard was illuminating the corresponding cyphertext at the other pair with respect to predetermined keys. Despite its commercial origins, the primary beneficiary of the invention turned out to be German Army. Since keys were based on rotor positions, Polish researchers succeeded in finding a way to break the Enigma. After reporting what they have found to the British Intelligence, Alan Turing and Gordon Welchman came up with the Bombe [6] as one of the greatest achievements of Bletchley Park, the code-breaking center of England. This machine was using electric circuits and by abusing a simple design flaw on Enigma, where plaintext letter consistently corresponds to a distinct character, i.e. *no letter remains unchanged in the encryption process.* Later in the war, the Lorenz Cipher presented an even greater challenge. Tommy Flowers led the development of Colossus, the world's first electronic, digital, programmable computer without memory, built to break this complex cipher at Bletchley Park. [7]

Computers

The incorporation of computers into information and security systems led to the swift development of new frameworks, outpacing and replacing older models. The rise in computing power and speed, coupled with the simultaneous reduction in computational costs, has enabled the exponential proliferation of highly sophisticated

cryptographic protocols and algorithms. During the post-war period, the power and importance of cryptography was regarded in a similar manner with Nuclear Power. [8] Initially, cryptography was primarily utilized for military intents, fostering the notion that its sole purpose was related to military affairs. Shortly after World War II and Cold War, nations started to see cryptographic applications as a matter of state secrecy. This approach later on caused severe issues when cryptography techniques were intended to be used for personal privacy and secrecy of networked systems.

Early ciphers employed symmetric key cryptography, where a single key is used for both encryption and decryption. To achieve cryptographic goals using symmetric keys, the key must be securely transferred between communicating parties. However, since encryption was specifically intended to secure communications over insecure channels, transferring private keys over these same channels posed a critical vulnerability, as eavesdroppers could easily intercept and read all messages. Nonetheless, if we consider that encryption was utilized due to the insecure nature of the communication channel, these communicating parties could not use this channel for transferring the private keys; otherwise an eavesdropper would be able to read all messages effortlessly.

This dilemma was resolved by the groundbreaking work of Whitfield Diffie and Martin Hellman in their seminal 1976 paper [9], which introduced the foundational concept of public key cryptography. Their proposal enabled secure key exchange over insecure channels, eliminating the need to transmit private keys and thereby addressing a long-standing vulnerability in classical cryptographic systems. Ron Rivest, Adi Shamir, and Leonard Adleman subsequently concretized this idea by developing the RSA algorithm, the first practical public-key cryptosystem based on the computational difficulty of factoring large integers into primes. RSA rapidly became a standard for encryption and digital signatures, effectively resolving long-standing vulnerabilities inherent in classical cryptographic systems.

In the early 1980s, Peter Shor was a young and ambitious student at Caltech, where he was profoundly influenced by the intellectual environment and the notable physicist Richard Feynman. A pivotal moment in Shor's career occurred in 1992 when Umesh Vazirani presented the Bernstein-Vazirani problem at Bell Labs [10]. This presentation underscored the superior efficiency of quantum computers over classical ones, as demonstrated using the Quantum Turing Machine model. Inspired by these insights, Shor delved deeper into the field of quantum computation. He built on Dan Simon's work on period-finding using the Fourier transform, recognizing its profound potential for cryptographic applications. Shor ingeniously adapted Simon's algorithm to tackle the discrete logarithm and factoring problems, which are fundamental to cryptographic systems like RSA [11]. Shor's groundbreaking work demonstrated that quantum computers could factor large numbers exponentially faster than classical algorithms, posing a significant threat to current cryptographic security.

Later, Miklós Ajtai published a groundbreaking paper demonstrating the NP-hardness of certain lattice problems and proposed their application to cryptography [12]. This

marked the advent of lattice-based cryptography, distinguished by robust worst-case hardness guarantees, in contrast to RSA's average-case computational assumptions, which were already threatened by Shor's algorithm.

In contrast, lattice problems are known to be computationally hard for classical computers and are believed to remain intractable even for quantum computers, making them a strong foundation for post-quantum cryptography [12]. Unlike RSA and ECC, no known quantum algorithm can efficiently solve lattice problems including Shor's [13]. This positions lattice-based cryptography as a strong and promising foundation for secure communication in the post-quantum era [14].

2 Elements of Computation

Cryptography, the practice of securing communication in the presence of third parties, relies heavily on the principles of computational complexity. The robustness of cryptographic systems' security is contingent on the inherent difficulty of selected mathematical problems. The cryptographic system can be broken if these problems can be solved efficiently. Conversely, if solving these problems requires an impractical amount of resources (time, computational power), the system remains secure. This delicate balance underscores the necessity of computational complexity in cryptography. Moreover, computational complexity enables the construction of cryptographic proofs. These proofs assure that a cryptographic system is secure under specified assumptions. They often involve demonstrating that breaking the cryptographic scheme would require solving a problem known to be computationally infeasible. This intersection of computational complexity and cryptographic proofs is essential for building trust in cryptographic protocols.

"... the enemy knows the system [15] ..."
— **Shannon**

The goal is to create an architecture that sustains its robustness even when all its details are known to potential threats. The solution rests on the deployment of sophisticated mathematical strategies and methodologies, irrespective of the concept's simplicity.

1. **Assume every detail about the system is disclosed.**
2. **Ensure problem complexity remains unsolvable despite this transparency.**
3. **Yet, for whosoever possesses the key, solving the problem of the system should be a straightforward task.**

In this chapter, we will explore the foundational concepts and background information that will lay the groundwork for subsequent discussions. One of the fundamental principles in cryptography, as highlighted by Claude Shannon, is the idea that a system should remain secure under any circumstances, even if every single detail about the system is captured except the key. This concept, known as Shannon's maxim, emphasizes the importance of designing cryptographic algorithms that do not rely on the secrecy of the algorithm itself but on the secrecy of the key. To achieve this, cryptographic systems are built on problems that are computationally hard to solve. The hardness of these problems ensures that even with full knowledge of the system, an adversary cannot break the encryption without possessing the secret key. Thus, we come to the crucial role of computational complexity in cryptography. Cryptographic proofs are formal methods used to demonstrate the security of cryptographic algorithms. These proofs typically show that breaking the algorithm would require solving a problem known to be hard, as per computational complexity theory. For instance, a proof might show that any algorithm capable of breaking a cryptographic system can be used to solve a problem that is considered hard to solve, thus implying that breaking the cryptographic system is as hard as solving the

problem. This connection provides a strong assurance of security. In summary, the connection between computational complexity and cryptography is fundamental to the design and analysis of secure systems. Understanding this connection is crucial for appreciating the robustness of cryptographic protocols and the assurances provided by cryptographic proofs. This section synthesizes insights from standard texts in automata theory and computational complexity, including works by Sipser [16], Hopcroft et al. [17], Arora and Barak [18], and Talbot and Welsh [19].

2.1 Problem Complexity

In the effort to comprehend the complexity of a task or problem from a computational viewpoint, we might encounter a perceptible difference between the theory and its practical manifestation. Owing to its insightful portrayal of the complexity notion and requirement for *efficient algorithms*[2] at hand, I hereby refer to precise formulation of The Travelling Salesman from "Complexity and Cryptography: An Introduction" [19]. While more advanced techniques to solve this problem exist, we will focus on the brute-force approach.

Example 1. (The Travelling Salesman) Given that a set of cities $\{c_1, c_2, ..., c_n\}$ and a distance matrix $D_{n\times n}$, where the $(i, j)^{th}$ entry in the matrix represents the distance from city c_i to city c_j:

$$D_{ij} = \text{distance}(c_i, c_j), \quad \forall i, j \in \{1, ..., n\}.$$

Identify the most efficient short-length route that encompasses each city at exactly one time, considering that the traveling salesman returns to the city of origin as a final destination.

A basic approach would be to request the device to traverse all possible permutations of the cities to identify the shortest route. Although the problem is solvable, the real question is how much time and resources it would take, especially when the number of cities C_n is large. By adopting the same variables from the authors of the book, let us postulate that your hypothetical route comprises 300 cities. Now we need to compare 300! different routes and $D_{300\times 300}$ since $n = 300$. To address this problem, we will employ two computers for comparison: a 2021 model MacBook Pro M1[3], which possesses a computational power of 10.4 teraflops and a hypothetical computer designed for illustrative purposes. Let us assume that we have brought this computer to the birth of the Earth. Disregarding practical limitations, let us assume that this device has been operating continuously until the present time to solve the aforementioned Travelling Salesman Problem.

- Age of the Earth in seconds $\leq 4.1 \times 10^{17}$
- Operations performed each second by the device: 10.4 TFLOPS $\leq 10^{13}$
- Total number of operations until this day: $\leq 4.1 \times 10^{30}$
- Number of operations needed for solving TSP with brute-force: $2^{300} \approx 10^{90}$

[2] See 17

[3] According to commercial papers and claims of Apple Inc

In a hypothetical scenario where a computer utilizes all the atoms in the Earth (approximately 3.6×10^{51} atoms) and each atom performs 10^{10} operations per second, the total number of operations this supermassive computer could have accomplished until today would be approximately 1.5×10^{79}.

$$\left(4.1 \times 10^{17}\right) \times \left(10^{10}\right) \times \left(3.6 \times 10^{51}\right) = 1.5 \times 10^{79}$$

However, the number of calculations performed to solve the problem is still insufficient. Based on the elucidation of the Travelling Salesman Problem, can we assert its inherent difficulty in finding a solution? Furthermore, when considering an alternate scenario involving another salesman and a distinct set of potential routes under comparable conditions, can we discern which query poses a lesser challenge compared to the other in terms of hardness? Cryptographers often enjoy complex mathematical problems, but this one is unlikely to interest them despite its difficulty. We will address the key questions in the following sections.

2.2 Turing Machines for Complexity Analysis

In the mid-20th century, the concept of modern computers as we know them today had not yet been realized. However, visionaries began to conceptualize machines capable of performing calculations and solving complex problems. Among the early innovators was Alan Turing, who introduced an essential model of computation known as the Turing Machine. Turing's profound insight was that the essence of computation did not require a complex apparatus; rather, a simple, abstract machine could suffice to execute any computation, provided it had sufficient time and resources. It is important to remember that Turing's work preceded the invention of computers. Before computers were invented, the term *computer* was referred to a person who computes' [20]. To comprehend the significance of Turing machines, we must consider the desired attributes of a computational device that may be considered equivalent to human computers. A key requirement is universality, the device should be capable of performing any calculation that any other computational device can undertake. Essentially, it should have the capability to simulate all other machines. Additionally, the model must be as simple as possible to allow rigorous analysis and understanding of its fundamental properties. A few paragraphs earlier, we utilized one of the well-known problems as an example: the *Travelling Salesman Problem*. We sought to explain why solving this problem is particularly challenging. Before we dive into the details of *Turing machines*, it is important to first clarify how *problems*, *algorithms*, and *languages* relate to one another. We will touch on key ideas from *Formal Language Theory*, but only to the extent needed to support our discussion, without getting too deep into its complexities. When discussing the concept of an *input alphabet*, we refer to a finite set of symbols or characters available for use. A *string* or *word* is defined as a sequence of symbols derived from this alphabet. A *language*, in this context, encompasses the set of all possible words composed of symbols from the given alphabet. An *algorithm* is a sequential list of steps designed to solve the given problem. If we can formalize a *problem* we wish to solve, such as *binary addition*, by expressing the problem in a way a Turing machine can understand, we can enable the machine to perform operations on the problem.

Remark 1. A Turing machine processes input strings and may either accept them, reject them, or run indefinitely if the strings are not in the language recognized by the machine.

Remark 2. A language qualifies as recursively enumerable if there is a Turing machine capable of accepting it.

Remark 3. A language is recursive if there exists a Turing machine that accepts it and halts on any input string.

This abstract machine, named after Alan Turing, is also a crucial concept for theoretical computational complexity. This theoretical computing device compromises an infinite tape divided into small cells where each cell accommodates a symbol from a pre-established alphabet. Machine operates under a predefined set of rules and transitions through states while simultaneously engaging in reading, writing, and navigating operations across the tape.

The component of the Turing machine responsible for horizontal navigation and read-write operations is also called tape-head. The fundamental lesson that we need to extract from the following explanations can be summarized as follows: Any computational task that is capable of being computed can be computed using Turing machines. The answer to the question of programmability can be understood through the final state of a Turing machine. If a halting state is achieved by the machine, it indicates that the computations for the given inputs have been completed.

Conversely, if the machine enters an infinite loop, it suggests either a programming error or the presence of a problem for a Turing machine cannot decide. Before introducing the formal definition of a Turing machine, it is essential to define the following possible outcomes:

- **Machine halts:** The Turing machine reaches either an accepting state or a rejecting state.

- **Machine does not halt:** The Turing machine runs indefinitely, entering an infinite loop.

Understanding these outcomes is fundamental in the study of computability and the limits of what can be computed. Let us take into account, for instance, the problem of integer addition. More formally, given two non-negative integers $x, y \in \mathbb{Z}$ where $x, y \geq 0$, the task is to compute $x + y$. It is imperative to provide an algorithmic solution for this issue. Another crucial aspect to highlight is the encoding of the input provided to the Turing machine, as this can significantly impact the running time. A well-documented example that demonstrates the impact of encoding involves the encoding of inputs in unary and binary forms for an identical problem while the desired output remains unchanged. This example is elaborated in detail by Talbot [19].

2.3 Deterministic Turing Machine

Deterministic Turing Machines are generally used for creating the theoretical model for computational complexity, classifying problems into complexity classes with respect to required time and space to solve it.

Definition 1 (Deterministic Turing Machine)**.** A deterministic Turing machine is defined as a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

where:

$$\begin{aligned}
Q &: \text{A set comprising a finite number of distinct states} \\
\Sigma &: \text{A finite set, input alphabet (excluding the blank-empty symbol } \sqcup) \\
\Gamma &: \text{A finite set, tape alphabet (with } \sqcup \in \Gamma \text{ and } \Sigma \subseteq \Gamma) \\
\delta &: Q \times \Gamma \to Q \times \Gamma \times \{L, R\} \\
q_0 &: \text{initial state} \\
q_{\text{accept}} &: \text{accepting state} \\
q_{\text{reject}} &: \text{rejecting state}
\end{aligned}$$

For each state $q \in Q$ and symbol $\gamma \in \Gamma$, $\delta(q, \gamma) = (q', \gamma', D)$ denotes the transition from state q to state q', where tape-head reacts based on the program configuration and current visible cell by replacing γ with γ', and moving the tape head in direction D, again according to the instructions, to the left-hand side (L) or to the right-hand side (R). The transition function δ is defined for every combination of states and tape symbols that can be seen. A deterministic Turing Machine is an infinite tape including blank symbols $\sqcup$ in tape alphabet Γ. The machine's ability to interpret other symbols and detect the end of the input is based on the transition function. The machine starts from the initial state or *starting state* and operates until it reaches an accepting state q_{accept} or a rejecting state q_{reject}. By the working principles of the machine, it is possible that it remains in an infinite loop (*non-terminating computations*) if it fails to reach a final state.

Remark 4. The Turing machine defined above is referred to as "deterministic" due to the nature of its transition function. This function specifies a unique action for every state and associated symbol.

The Deterministic Turing Machine dictates the next move based on the current state and the symbol being read. This deterministic nature ensures that for any given state and input symbol, there is a precisely defined action, which includes writing a symbol, moving the tape head, and transitioning to a subsequent state. Consequently, the behavior of the deterministic Turing machine is entirely predictable and repeatable for the same input and initial configuration.

Remark 5. The running time or the execution time of an algorithm is evaluated based on the total number of operations executed by a Turing Machine for a specific task.

When we assess an algorithm's running time using a Turing Machine, we count the number of fundamental operations (such as reading a symbol, writing a symbol, and moving the tape head) required to complete the algorithm for a given input size. This count provides an asymptotic measure of the algorithm's efficiency to differentiate and compare whether an algorithm is considered better or faster for the given problem. However, it is important to understand that there are types of problems that deterministic Turing Machines are not capable of solving.

Let us examine a specific type of problem. Suppose we want to determine whether a particular number is a prime number. This problem has only two possible outcomes, making it very straightforward: yes or no. A number is either prime or it is not. Here, we see that the algorithm solving this problem makes a binary decision. Such problems are typically referred to as decision problems.

Definition 2. A *decision problem* is simply a yes-or-no question concerning an infinite set of inputs, meaning it has only two possible outputs (*yes* or *no*) for any given input. [21]

Definition 3 (Church-Turing Thesis)**.** Every computation deemed effective might be performed by using a Turing machine. Likewise, any computation performed by a Turing machine is achievable through an effective computational process, even if it takes a relatively long time.

One of the most crucial aspects that we need to learn from the Church-Turing thesis is the existence of undecidable problems.

Definition 4 (Decidable Problem)**.** Given that A is a problem and M is a Turing machine, a problem is a Decidable Problem if and only if there exists an M to solve the problem for any input in a finite amount of time.

One of the most famous undecidable problems was also introduced by Turing in 1936 [22] as an analysis of decision problems with the original German name *Entscheidungsproblem*. Later on, the problem was named after usage of the term *halting* in [23]

Definition 5 (Palindrome)**.** Given the alphabet Σ, a palindrome refers to any word x in Σ^* that exhibits symmetrical characteristics, being readable in the same manner when read forward and backward. For example, 010 and 1001 are palindromes, whereas 10 is not.

Definition 6 (Decider)**.** Let M denote a Turing machine that halts on all inputs, i.e., never enters an infinite loop. Such Turing machines are defined as **deciders**.

Definition 7 (Verifier)**.** Let V denote a Turing machine that halts on all inputs. A verifier for a language A is a V takes an input pair (w, c_i), where w is the main input string, and c_i is a certificate.

$$A = \{w \mid V \text{ accepts } (w, c_i) \text{ for some string } c_i\}$$

As such, the verifier acts as a proof-checker, helping to ascertain the membership of

w in the language A based on the information contained in the certificate c_i.

Example 2 (Turing Machine for Palindromes)**.** Given that a Deterministic Turing Machine M computes binary string x as follows.

$$M(x) = \begin{cases} Accept, & \text{if } x \text{ is a palindrome} \\ Reject, & \text{otherwise} \end{cases}$$

Formally, M is defined with tape alphabet $\Gamma : \{0, 1, b, \text{start}\}$, the state set $Q : \{q_0, q_A, q_R, q_{\text{have0}}, q_{\text{match0}}, q_{\text{have1}}, q_{\text{match1}}, q_{\text{reverse}}\}$, accepting state $q_{accept} = q_A$, and rejecting state $q_{reject} = q_R$. We define transition function δ as follows:

$$\begin{aligned}
&\delta(q_0, \sqcup) = (q_A, 1, L).\\
&\delta(q_0, 0) = (q_{\text{have0}}, \sqcup, R).\\
&\delta(q_0, 1) = (q_{\text{have1}}, \sqcup, R).\\
&\delta(q_{\text{have0}}, 1) = (q_{\text{have0}}, 1, R).\\
&\delta(q_{\text{have0}}, 0) = (q_{\text{have0}}, 0, R).\\
&\delta(q_{\text{have0}}, \sqcup) = (q_{\text{match0}}, \sqcup, L).\\
&\delta(q_{\text{match0}}, 1) = (q_R, \sqcup, L).\\
&\delta(q_{\text{match0}}, 0) = (q_{\text{rev}}, \sqcup, L).\\
&\delta(q_{\text{match0}}, \sqcup) = (q_{\text{A}}, \sqcup, L).\\
&\delta(q_{\text{reverse}}, 0) = (q_{\text{reverse}}, 0, L).\\
&\delta(q_{\text{reverse}}, 1) = (q_{\text{reverse}}, 1, L).\\
&\delta(q_{\text{reverse}}, \sqcup) = (q_0, \sqcup, R).\\
&\delta(q_{\text{have1}}, 1) = (q_{\text{have1}}, 1, R).\\
&\delta(q_{\text{have1}}, 0) = (q_{\text{have1}}, 0, R).\\
&\delta(q_{\text{have1}}, \sqcup) = (q_{\text{match1}}, \sqcup, L).\\
&\delta(q_{\text{match1}}, 0) = (q_R, \sqcup, L).\\
&\delta(q_{\text{match1}}, 1) = (q_{\text{reverse}}, \sqcup, L).\\
&\delta(q_{\text{match1}}, \sqcup) = (q_A, \sqcup, L).
\end{aligned}$$

The transition function δ of a deterministic Turing machine M for recognizing palindromes over a binary alphabet is designed to systematically compare characters from both ends of the input string. The machine starts at the leftmost symbol and transitions into different states to locate and match corresponding characters from the right end. Specifically, M uses states such as $q0$, $qhave0$, $qhave1$, $qmatch0$, $qmatch1$, and $qreverse$ to identify and validate matching pairs. If M successfully matches all pairs and reaches the blank symbol $\sqcup$, it transitions to the accepting state qA. Conversely, if a mismatch is detected, M transitions to the rejecting state qR. δ formally maps each state and symbol pair to a new state, symbol, and head

movement, ensuring the proper verification process for palindromes. For instance, $\delta(q0, \sqcup) = (qA, 1, L)$, $\delta(q0, 0) = (qhave0, \sqcup, R)$, and $\delta(qmatch0, 0) = (qrev, \sqcup, L)$, among others, illustrate the specific transitions necessary for palindrome validation. A commonly employed method to represent a Turing machine is through the utilization of a transition diagram. This diagram takes the form of a directed graph, with distinct vertices representing different states in the set Q. Occasionally, nodes without any incoming or outgoing edges may also appear in the graph. The edges within the graph signify each transition triggered by distinct input symbols as defined in the transition function. In order to assist in comprehension, each edge is labeled with the letter read, the letter to be written, and the tape-head direction, while each vertex is labeled with the corresponding state. By utilizing this graphical representation and arrowheads to indicate the direction of the edge, behavior and state transitions of the Turing machine can be visually depicted in a clear and intuitive manner.

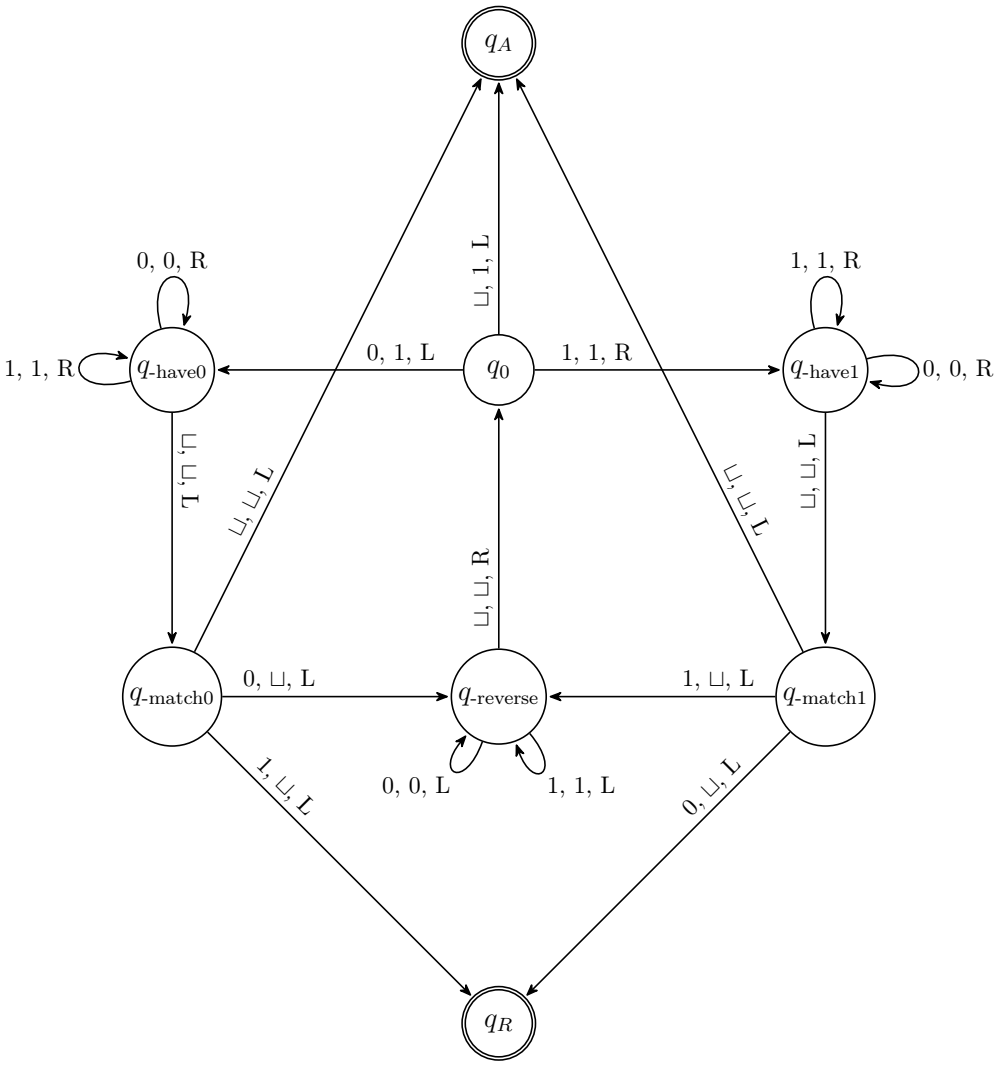

Figure 2: State Diagram of the Example Turing Machine

By employing the machine on a tape that includes binary palindrome symbols such as "100001" or "010010", we ascertain that the execution concludes upon encountering an accepting state.

There exist problems for which the fixed sequence of operations of a Deterministic Turing Machine (DTM) is insufficient to provide an efficient solution. For instance, problems like the Boolean satisfiability problem and certain optimization problems require exploring a vast number of potential solutions. A DTM can only follow one computational path at a time, making it inefficient for such tasks.

To address these limitations, other variants of Turing machines have been conceptualized. These machines differ fundamentally in that their transition functions allow multiple possible actions for a given state and input symbol pair. For example, a Non-deterministic Turing Machine (NDTM) can have several possible transitions, enabling the machine to explore many computational paths simultaneously. Another variant is the probabilistic Turing machine, which utilizes randomness to explore alternative computational paths. This capability allows NDTMs to solve certain problems more efficiently in theory, as they can essentially *guess* a solution path and verify it in polynomial time. Despite this theoretical advantage, it remains an open question whether NDTMs are more powerful than DTMs in terms of computational complexity.

2.4 Non-deterministic Turing Machine

Non-deterministic Turing Machines are theoretical models that extend deterministic Turing machines by allowing multiple possible transitions for a given state and tape symbol. They are used to explore the boundaries of what can be computed and to classify problems into complexity classes such as NP.

Definition 8. A non-deterministic Turing machine is defined as a 7-tuple $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where:

- Q: A set comprising a finite number of distinct states
- Σ: A finite set, corresponding to the input alphabet (excluding the blank-empty symbol $\sqcup$)
- Γ: A finite set, the tape alphabet (with $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$)
- δ: $Q \times \Gamma \to P(Q \times \Gamma \times \{L, R\})$
- q_0
- q_{accept}
- q_{reject}

Here, $\mathcal{P}(Q \times \Gamma \times \{L, R\})$ denotes the power set of $Q \times \Gamma \times \{L, R\}$, allowing the transition function to return a set of possible next moves rather than a single one. This non-determinism allows the machine to explore many computational paths simultaneously.

2.5 Probabilistic Turing Machine

A Probabilistic Turing Machine (PTM) is a variant of the classical Turing Machine that incorporates randomness into its computation process. This additional feature allows the PTM to explore multiple computational paths based on probabilistic transitions, making it a powerful tool for solving certain types of problems more efficiently than deterministic machines.

Definition 9. (Probabilistic Turing Machine) A Probabilistic Turing Machine is formally defined as a 7-tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where:

- Q is a finite set of states.
- Σ is a finite set called the input alphabet, which does not include the blank symbol $\sqcup$.
- Γ is a finite set called the tape alphabet, where $\Sigma \subseteq \Gamma$ and $\sqcup \in \Gamma$.
- δ is a transition function defined as $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$, where $\mathcal{P}(Q \times \Gamma \times \{L, R\})$ denotes a set of possible moves. Each move can be assigned a probability.
- q_0 is the initial state.
- q_{accept} is the accepting state.
- q_{reject} is the rejecting state.

PTM operates similarly to a Deterministic Turing Machine but with the key distinction that its transition function includes probabilistic choices. At each computational step, the PTM may have several possible moves, each associated with a certain probability. The machine utilizes a random number generator to decide which move to make based on these assigned probabilities, introducing an element of randomness into the computation process.

Remark 6. Probabilistic Turing Machine is a variant of Non-deterministic Turing Machine.

Consider a PTM designed to decide a simple language

$$L = \{w \mid w \text{ contains an even number of } 1s\}.$$

This machine can employ randomness to determine its next move when it encounters a symbol. For example, let the PTM be defined as $P = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where $Q = \{q_0, q_1, q_{\text{accept}}, q_{\text{reject}}\}$, $\Sigma = \{0, 1\}$, and $\Gamma = \{0, 1, \sqcup\}$. The initial state is q_0, the accepting state is q_{accept}, and the rejecting state is q_{reject}. The transition function δ includes probabilistic transitions such as $\delta(q_0, 1) = \{(q_1, 1, R, 0.5), (q_0, 1, R, 0.5)\}$. This means that when the machine is in state q_0 and reads a 1, it has two possible transitions: with probability 0.5, it transitions to state q_1, writes a 1, and moves right; with the other probability of 0.5, it remains in state q_0, writes a 1, and moves right. Other transitions in the machine might be deterministic, such as $\delta(q_1, 1) = \{(q_0, 1, R, 1.0)\}$, meaning the machine will certainly transition to state q_0 if it reads a 1 while in state q_1.

The introduction of randomness allows the PTM to potentially follow different computational paths, providing a mechanism to explore various solutions or behaviors. This is particularly advantageous in solving certain problems more efficiently than Deterministic Turing Machines. For example, PTMs are useful in probabilistic algorithms where solutions can be found in expected polynomial time even if the worst-case deterministic time is much higher. This makes PTMs a valuable tool in fields such as cryptography, randomized algorithms, and complexity theory, where the ability to handle randomness can lead to significant performance improvements and innovative solution strategies.

When discussing probabilistic Turing machines and the concept of randomness, we encounter several challenges. Typically, when considering a fair die, we expect the outcome of a roll to be random. However, from a philosophical perspective, one might argue that this perceived randomness is merely a result of our lack of information about the die and the rolling process, thereby creating an illusion of randomness. This invites a deeper exploration into the distinction between what is considered random and what is truly random or pseudorandom. Efforts grounded in classical physics fall short in adequately explaining the randomness inherent in quantum mechanics, which operates under its own unique principles. The *Quantum Circuit Model* provides a concrete realization of this true randomness, harnessing quantum phenomena to drive computations that defy classical predictability. For those intrigued by the nuances of quantum randomness and its implementation, further details can be explored in the appendix. The Quantum Circuit Model is computationally equivalent to *the Quantum Turing Machine*.

Definition 10. (Oracle) An oracle or *blackbox* is a Turing machine that can solve **any** problem within only a single operation.

Oracles serve as a crucial means to explore and understand the limits of computational power and efficiency. In classical computation, an oracle allows us to bypass the conventional step-by-step processing of algorithms by providing immediate solutions to complex problems. This theoretical construct helps researchers to conceptualize and analyze problems that are otherwise intractable, offering insights into the nature of computational complexity and the boundaries of what can be computed efficiently.

2.6 Time and Space Complexity

Before presenting formal definitions of complexity and hardness, we briefly interpret a Turing machine as a model of an algorithm. The running time of a Turing machine is measured by the number of steps it performs before halting, expressed as a function of the input size $|n|$. A computational problem can often be solved by multiple algorithms, each producing the same output for a given input. Informally, one may view these as different sets of instructions achieving the same goal. However, the efficiency and execution speed of these algorithms can vary significantly, depending on their design and underlying computational principles.

Definition 11 (Running Time)**.** Let M be a DTM, given that halts in all inputs.

A function $T : \mathbb{N} \to \mathbb{N}$ is used to represent the time bound for M. A DTM M is said to run in time $T(n)$ if computations take (n) steps where $n = |x|$ is input length.

Definition 12 (Running Space)**.** Let M be DTM that halts in all inputs. A function $S : \mathbb{N} \to \mathbb{N}$ is used to represent the nonempty space bound for M. A DTM M is said to use space $S(n)$ if computations require $S(n)$ tape cells.

In pursuit of designing efficient algorithms, it becomes imperative to conduct comparative analyses of diverse functions. Thus, we shall employ the method of asymptotic order of growth, a widely accepted approach, for conducting such comparisons. Asymptotic analysis is a mathematical way of expressing the limiting behavior of a function when the argument shifts towards a certain value or tends to infinity. Consider a function $f(n)$ which shall be established as $f(n) = n^{100} + n$. As n increases, the lower-order term n is insignificant compared to the dominant term n^{100} and thus can be safely ignored in asymptotic analysis. The following definition is a formal expression of the most commonly used worst-case notation referred to as *Big-O Notation.*

Definition 13 (Big-O)**.** Let $f, g : \mathbb{N} \to \mathbb{R}^+$ be functions. We say

$$f(n) = O\big(g(n)\big)$$

if and only if there exist constants $c > 0$ and $n_0 \in \mathbb{N}$ such that, for all $n \geq n_0$,

$$0 \ \leq \ f(n) \ \leq \ c \cdot g(n).$$

Example 3. Let $f, g : \mathbb{N} \to \mathbb{R}^+$ be functions defined as $f(n) = 1500n^3$, and $g(n) = n^3$. As defined earlier, it can be shown that $f(n) = O(g(n))$. Recall that $f(n) \geq 0$ for all n. To be able to use the definition, we need a positive constant $c > 0$ such that $f(n) \leq g(n)$ for all $n \geq n_0$. We have

$$1500n^3 \leq c \cdot n^3 \Leftrightarrow 1500 \leq c, \quad \text{where } n > 0 \ .$$

Hence, we may choose any c such that $c \geq 1500$. Let us choose $c = 1500$ for simplicity. The inequality $c \geq 1500$ is satisfied for all $n \geq 1$, so we may choose $n_0 = 1$ and equality holds for all $n \geq n_0$. Hence, we have established that $f(n) \leq g(n)$.

While complexity analysis encompasses not only Big O notation for the worst case scenario, there exist other indicators for specifying other scenarios for the computation performance such as best-case or average-case. Rather than providing definitions for all of them one by one, we will utilize a table to describe other members of the asymptotic notation.

Big-O notation provides an upper limit on the growth rate of a function. It reflects the worst-case performance of an algorithm. For example, if an algorithm has a time complexity of $O(n^2)$, it means that in the worst case, the algorithm's running time will grow no faster than n^2 as the input size n increases. Omega notation gives a lower limit on the growth rate of a function. It reflects the best-case performance of an algorithm. For instance, if an algorithm has a time complexity of $\Omega(n \log n)$, it means that in the best case, the algorithm's running time will grow at least as

Notation	Formal Definition	Interpretation
$O(g(n))$	There exist constants $C > 0$, n_0 such that $f(n) \leq Cg(n)$ for all $n > n_0$	Upper bound on growth rate (e.g., worst-case time complexity)
$\Omega(g(n))$	There exist constants $C > 0$, n_0 such that $f(n) \geq Cg(n)$ for all $n > n_0$	Lower bound on growth rate
$\Theta(g(n))$	There exist constants $C_1, C_2 > 0$, n_0 such that $C_1 g(n) \leq f(n) \leq C_2 g(n)$ for all $n > n_0$	Tight bound: growth rate is bounded both above and below by $g(n)$ asymptotically
$o(g(n))$	For all constants $C > 0$, there exists n_0 such that $f(n) < Cg(n)$ for all $n > n_0$	Strictly smaller growth: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
$\omega(g(n))$	For all constants $C > 0$, there exists n_0 such that $f(n) > Cg(n)$ for all $n > n_0$	Strictly larger growth: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Table 2: Asymptotic notation and their asymptotic interpretations

fast as $n \log n$ as the input size n increases. Theta notation provides both lower and upper bounds on the growth rate of a function, meaning that the function grows at the same rate as $g(n)$ up to constant factors, both in the upper and lower bounds. If an algorithm has a time complexity of $\Theta(n \log n)$, it means that the running time is asymptotically bounded both above and below by $n \log n$.

Before we start examining the formal intricacies of Turing machines, it is imperative to grasp the integral relationships among the machine's alphabet, its algorithm, its language, and the problems it is designed to solve. This journey is not merely about studying abstract concepts; it is about understanding the fundamental framework that underpins computational theory. A Turing machine utilizes a finite set of symbols known as its alphabet. This collection of symbols is not just a tool; it is the very foundation upon which the machine operates, dictating what it can compute and the nature of the outputs it can produce. The algorithm of a Turing machine is akin to a meticulously crafted blueprint, it outlines precise instructions for the machine to follow. Each step in an algorithm is designed to manipulate and rearrange the symbols from the alphabet, guiding the machine through a series of states towards the resolution of a problem or conclusively determining the absence of a solution. The language of the Turing machine, defined by all the sequences of symbols it can rightfully accept, represents the set of all the problems it can solve. Each problem posed to a Turing machine can be viewed, at its core, as a decision query: Given

an input string, does it belong to the language recognized by the machine? The interplay between the alphabet, algorithm, and language is profound: the alphabet provides the elemental building blocks, the algorithm leverages these blocks to construct meaningful outcomes, and the language defines the scope of challenges the machine is equipped to tackle. This dynamic illustrates the core equivalence between computational problems and formal languages, where each language encapsulates a unique set of problems that a Turing machine, configured accordingly, can resolve. Understanding this equivalence is crucial, it highlights the essential limits and capabilities of computational mechanisms in a deeply theoretical yet practical manner.

Definition 14 (DTIME)**.** Let M denote a DTM computing on the binary input strings $\{0,1\}^*$. Let $T : \mathbb{N} \to \mathbb{N}$ be a function and let A be a language such that $A \subseteq \{0,1\}^*$. A language L is in $\mathbf{DTIME}(T(n))$ if and only if there exists a machine M that runs in time $T(n)$ and decides a language A such that $A = L(M)$.

Definition 15 (DSPACE)**.** Let M denote a deterministic Turing machine computing on the binary input strings $\{0,1\}^*$. Let $S : \mathbb{N} \to \mathbb{N}$ be a function and let A be a language such that $A \subseteq \{0,1\}^*$. A language L is in $\mathbf{DSPACE}(S(n))$ if and only if there exists a machine M that uses space $S(n)$ and decides a language A such that $A = L(M)$.

Definition 16 (Complexity Class P)**.** The complexity class P is defined as the union of all deterministic time complexity classes $DTIME(n^x)$ where $x \geq 1$. P is also referred to as the polynomial-time class.

$$P = \bigcup_{x \geq 1} DTIME(n^x)$$

P and **NP** problems hold significant importance in the theoretical studies of computability, as they serve as a foundational concept for understanding the feasibility or the efficiency of various computational tasks. Until this definition, we have used the term *efficient* on some pages, which may have been somewhat ambiguous and vague. By introducing the polynomial-time complexity class, we can clarify the intended meaning behind the term *efficient*. The inclusion of problems in **P** indicates that they can be tackled efficiently by algorithms, which often translates to practical applications and real-world utility.

Definition 17 (Efficient Algorithms)**.** An algorithm A is considered efficient if it is a polynomial-time algorithm.

Similarly, if two distinct algorithms run at the same time, we will claim that the algorithm terminates faster is more efficient than the other one. In the context of Class **P** Problems, it is conceivable that these challenges can elude the necessity for brute-force methodologies by attaining a solution in polynomial time. Some known **P** Class problems can be listed as follows:

- **Maximum Matching Problem:** In Graph Theory, given an undirected graph, find the largest possible set of maximum matchings.

- **Greatest Common Divisor:** Given two numbers x, y, find the largest positive integer that divides x, y without any remainders.
- **Primality Testing:** Given number x, determining if given number is prime or not [24].
- **Sorting Problem:** Given a list of elements, arranging elements either in ascending or descending order.
- **Relative Primes:** Given two numbers x, y, deciding if they are relatively primes or not.

Definition 18 (Complexity Class NP)**.** Non-deterministic Polynomial Time **NP** is the class of problems or languages that have polynomial time verifiers with 'yes' certificates.

Remark 7. **NP** should not be interpreted as Not-Polynomial; it actually stands for Non-Deterministic Polynomial. Specifically, the problem is resolved by a Non-Deterministic Turing Machine instead of a Deterministic Turing Machine.

In other words, it refers to the collection of decision problems that can be solved in polynomial time using a non-deterministic Turing machine.

If you recall, at the beginning of this chapter, we discussed the traveling salesman problem and emphasized its inherently time-consuming nature[4], even with the aid of practically unattainable computational resources. However, if we reformulate the question as a decision problem—"Does there exist a tour of length at most k?" we gain a simple verification procedure. Given any candidate route, we can in polynomial time confirm that it visits each city exactly once and that its total length does not exceed k. This observation places the decision variant of the Traveling Salesman Problem within the class **NP**. Some similar examples for the well-studied problems in **NP** class may be listed as follows;

- **Subset Sum Problem:** Given a finite set of integers, is it possible to find a subset where the subset sum is equal to the desired value D?
- **Hamiltonian Cycle:** Given a finite graph G, is it possible to visit each node exactly once and return the starting point?
- **Knapsack Problem:** Given a finite set of items each with its own weight and value, and a knapsack with a fixed weight limit, can we choose a selection of items whose combined weight does not exceed the limit and whose combined value is as large as possible?
- **Graph Coloring:** Given a finite graph, is it possible to color the vertices such that adjacent vertices are assigned to k-different colors?

To illustrate the distinction between P and NP problems, we will use an example involving the multiplication of two large prime numbers and the subsequent problem of factorizing the product.

[4]Undoubtedly, there exist more efficient ways compared to brute force method

Example 4. Let us consider two large prime numbers:

$$p_1 = 997$$
$$p_2 = 991$$

Multiplying these primes is straightforward and computationally efficient.

$$N = p_1 \times p_2$$
$$N = 997 \times 991$$
$$N = 988027$$

Problem: Factoring the Product Given the product $N = 988027$, the goal is to find the original prime factors p_1 and p_2. This is an example of a problem in NP (non-deterministic polynomial time).

Why It is Hard to Solve Factoring large numbers, especially when they are products of two large primes, is computationally challenging. The difficulty lies in the fact that there are no known efficient algorithms (i.e., algorithms that run in polynomial time) to factorize such products. To find p_1 and p_2 from N, one would typically have to try a large number of possible factors, which can be very time-consuming.

Why It is Easy to Verify Once the prime factors p_1 and p_2 are known, verifying the solution is straightforward. You simply multiply the two numbers and check if their product equals N. This multiplication can be done in polynomial time. Since the product matches N, the factors p_1 and p_2 are correctly verified.

- **Multiplying two large primes (P problem):** Multiplication is fast, efficient and straightforward.
- **Factoring the product into prime factors (NP problem):** This is a problem that belongs to the NP class where the polynomial time algorithms are not efficient enough to solve. It is easy to verify a solution once it is found but hard to solve initially.

Continuing from the distinction between **P** and **NP** problems illustrated by the multiplication and factorization of large prime numbers, we delve into the broader context of computational complexity and one of the most famous questions in computer science: the **P** vs **NP** problem [25], with a particular focus on its implications for cryptography and the emerging field of quantum computing. The **P** vs **NP** problem, one of the seven Millennium Prize Problems [26], questions whether every problem whose solution can be quickly verified (**NP**) can also be quickly solved (**P**). This distinction is crucial for cryptography, as many cryptographic systems rely on the difficulty of **NP** problems like factoring large numbers to secure data. If P were to equal **NP**, it would mean that these cryptographic protocols could be broken efficiently, leading to a significant vulnerability in digital security. The advent

of quantum computing further complicates this landscape, as quantum algorithms like Shor's algorithm can factor large numbers in polynomial time, threatening the foundations of current cryptographic methods. While quantum computers do not resolve **P** vs **NP** problem, they do offer polynomial-time solutions to specific problems believed to be hard for classical computers, such as integer factorization and discrete logarithms. Despite significant research efforts, the **P** vs **NP** question remains unsolved, continuing to challenge our understanding of computational limits and the security of digital systems. Whether **P** equals **NP** has profound implications, not only offering a million-dollar reward but also the potential to revolutionize technology, cryptography, and our overall approach to computational problems.

Definition 19 (Complexity Class co-NP)**.** Co-Non-Deterministic Polynomial Time **co-NP** is the class of problems or languages that have polynomial time verifiers with 'no' certificates. Equivalently, co-NP contains complements of problems in complexity class **co-NP**.

Remark 8. A problem L might be categorized as **co-NP** if there is a **NDTM** that rejects all words $x \notin L$ in polynomial time. It is still a mystery whether **co-NP** is different than **NP**.

Some examples of the **co-NP** problems are listed below:

- **Co-Subset Sum Problem:** Given a set of integers, the problem is to decide whether there is subset, where the subset sum is equal to the desired value D?
- **Co-Hamiltonian Cycle:** Given a graph, is it possible to decide if there is no Hamiltonian Cycle?
- **Co-Graph Coloring:** Given a graph, is it not possible to color vertices of the graph such that vertices being adjacent by having a common edge are filled with different colors?

It is evident that there exist problems that remain unsolved or lack efficient solutions. Consequently, the inquiry arises as to whether we can effectively solve problems that bear resemblance to the problems that we are failing to solve in an effective manner. The strategic practice of transforming a given problem into another -efficiently solvable if possible- variant, in such a way that a resolution for the transformed problem may be utilized to construct a solution for the original problem, is referred to as **reduction**.

Definition 20 (Karp Reduction)**.** Given two languages $A, B \subseteq \{0,1\}^* = \Sigma^*$, A is polynomial-time Karp reducible to B denoted by $A \leq_p B$, if there exists a problem transformation function denoted by $f : \{0,1\}^* \to \{0,1\}^*$ which is computable in polynomial time, such that for all $x \in \{0,1\}^*$, $x \in A$ if and only if $f(x) \in B$ [18].

Definition 21 (NP-hard)**.** Given that A is a decision problem that returns 'yes' or 'no' answers, A is **NP-hard** if there exists another NP-hard problem L reducible to problem A.

Definition 22 (NP-complete)**.** Given that problem $A \in$ **NP-hard** and A $\in$ **NP**, A problem A is defined as **NP-complete**.

Definition 23 (coNP-hard)**.** Given that A is a decision problem that returns 'yes' or 'no' answers, it is said to be **coNP-hard** if a single other coNP problem L is reducible to problem A.

Definition 24 (coNP-complete)**.** Given that problem $A \in$ coNP-hard and $A \in$ **coNP**, A problem A is defined as **coNP-complete**.

We now shift our focus to another complexity class, which introduces Probabilistic Polynomial Time (PP) to expand our understanding of computational complexity by incorporating the element of probabilistic computation within polynomial time constraints.

Definition 25 (Probabilistic Polynomial Time (**PP**))**.** A language L is in PP if there exists a probabilistic Turing machine M that operates in polynomial time for all random choices and satisfies the following conditions:

- If $x \in L$, then the probability that M accepts x is greater than $\frac{1}{2}$.
- If $x \notin L$, then the probability that M accepts x is at most $\frac{1}{2}$.

This definition indicates that M performs more effectively than a basic algorithm that would accept inputs with a $\frac{1}{2}$ probability.

Definition 26 (Bounded Probabilistic Polynomial Time (**BPP**))**.** A language L belongs to BPP if there exists a probabilistic Turing machine M that operates in polynomial time for all random choices and satisfies the following conditions:

- If $x \in L$, then the probability that M accepts x is at least $\frac{2}{3}$.
- If $x \notin L$, then the probability that M accepts x is at most $\frac{1}{3}$.

Definition 27 (Randomized Polynomial Time (**RP**))**.** A language L is in RP if there exists a probabilistic Turing machine M that operates in polynomial time for all random choices and satisfies the following conditions:

- If $x \in L$, then the probability that M accepts x is at least $\frac{1}{2}$.
- If $x \notin L$, then M rejects x with probability 1.

The classes **PP** (Probabilistic Polynomial Time), **BPP** (Bounded Probabilistic Polynomial Time), and **RP** (Randomized Polynomial Time) represent different levels of probabilistic computational complexity, each with distinct characteristics regarding error tolerance and acceptance probabilities.

Starting with **PP**, this class includes languages for which there exists a probabilistic Turing machine that runs in polynomial time, and for any input, it decides the language with a probability of more than $\frac{1}{2}$ for positive instances and at most $\frac{1}{2}$ for negative instances. The main characteristic of PP is its lenient requirement on the acceptance probability, which only needs to be slightly better than random guessing. This makes PP quite broad and theoretically interesting, but it is less practical for applications requiring high accuracy due to its high allowable error rate.

On the other hand, **BPP** demands a higher standard of confidence in the results. For a language to belong to **BPP**, there must be a probabilistic Turing machine that

runs in polynomial time and decides the language such that the acceptance probability for positive instances is greater than $\frac{2}{3}$, while for negative instances, it is less than $\frac{1}{3}$. This class strikes a balance between efficiency and reliability, ensuring a substantial gap between acceptance probabilities for positive and negative instances, which translates to a lower error rate. As a result, **BPP** is widely considered practical for solving problems efficiently with high confidence.

In contrast, **RP** is characterized by its one-sided error. A language is in **RP** if there is a probabilistic Turing machine running in polynomial time that accepts positive instances with a probability of at least $\frac{1}{2}$ and rejects negative instances with certainty (zero probability of acceptance). This one-sided error model ensures no false negatives, making **RP** particularly useful for applications where it is crucial to avoid false negatives, such as in certain primality testing algorithms. While **RP** ensures reliability in rejecting incorrect instances, its acceptance probability for correct instances can be relatively low compared to **BPP**.

In comparison, the relationship between these classes can be summarized as $\mathbf{RP} \subseteq \mathbf{BPP} \subseteq \mathbf{PP}$ [5]. This hierarchy reflects increasing inclusiveness: every problem in **RP** is also in **BPP**, and every problem in **BPP** is also in **PP**. However, the practical utility decreases as we move from **RP** to **PP**. **RP** is highly reliable for specific applications, **BPP** offers a good balance for a wide range of practical problems, and **PP**, though theoretically significant, is less practical due to its higher tolerance for errors.

In summary, while **PP** provides a broad theoretical framework with minimal accuracy requirements, **BPP** and **RP** offer more practical and stringent conditions, with **BPP** being particularly useful for high-confidence polynomial-time solutions, and **RP** ensuring no false negatives in polynomial time.

Considering the fundamental aspects of the hardness and complexity of a problem discussed thus far, it may appear that acquiring a substantial amount of information and knowledge is essential to discuss the complexity of *The Traveling Salesman* problem, as previously mentioned in 1. At this juncture, we have the basis for asserting that the Traveling Salesman Problem (TSP) falls within the **NP-hard** complexity class [27] since deciding whether a graph has a *Hamiltonian Cycle* is in **NP-hard**. Despite all these discussions and the hardness levels of the problems mentioned, it should be noted that there are problems belonging to much harder classes in the literature, such as *exponential time*. However, these problems are not suitable for cryptographic purposes, since the mere complexity or hardness of a mathematical problem is not sufficient for being suitable to be used in cryptographic purposes. Furthermore, in order to grasp the potential of quantum computers and their impact on cryptography, it becomes imperative to gain insight into the theoretical boundaries of current classical computing through a comprehensive understanding of complexity theory.

[5] While these inclusions are known to hold, whether they are strict remains an open question in complexity theory.

2.7 Algorithm Complexity Analysis

The complexity of an algorithm is typically analyzed in three scenarios:

- **Best-case complexity** refers to the shortest execution time of an algorithm when given the most favorable input.
- **Average-case complexity** represents the expected runtime over all possible inputs, assuming a uniform probability distribution.
- **Worst-case complexity** provides an upper bound on execution time, ensuring performance guarantees even for the most challenging inputs.

Many cryptographic protocols derive their security from problems that are computationally hard in the worst case. However, for a cryptographic scheme to be secure against practical adversaries, the underlying problem must also be difficult on average. If a problem is only worst-case hard but easy for most instances, then cryptographic constructions relying on it may be susceptible to practical attacks.

To mitigate this, cryptographic constructions often employ *worst-to-average-case reductions*, which establish that solving a random instance of the problem is at least as hard as solving the most difficult instances. These reductions ensure that even typical problem instances remain computationally intractable, reinforcing the security of cryptographic systems.

Quantum complexity theory extends classical complexity classes to quantum computational models. The key class of quantum complexity is **Bounded-Error Quantum Polynomial Time (BQP)**, which consists of decision problems solvable by a quantum computer in polynomial time with a probability of error at most $\frac{1}{3}$.

Definition 28 (**BQP**)**.** A language L is in BQP (Bounded-Error Quantum Polynomial Time) if there exists a quantum algorithm A that runs in polynomial time and, for every input x, correctly decides whether x is in L with probability at least $\frac{2}{3}$.

Intuitively, **BQP** captures the class of decision problems that can be efficiently solved by a quantum computer with a bounded probability of error. However, just as **NP** extends **P** by introducing verifiable proofs, the quantum analogue Quantum Merlin-Arthur (**QMA**) extends **BQP** by allowing a polynomial-size quantum proof that a polynomial-time quantum verifier can check.

Definition 29 (**QMA**)**.** A language L is in QMA if there exists a polynomial-time quantum verifier $V(x, \rho)$ such that:

- If $x \in L$, then there exists a quantum witness ρ (a polynomial-size quantum state) such that the verifier accepts with probability at least $\frac{2}{3}$:

$$\exists \rho \quad \Pr[V(x, \rho) = 1] \geq \frac{2}{3}.$$

- If $x \notin L$, then for all quantum states ρ, the verifier accepts with probability

at most $\frac{1}{3}$:

$$\forall \rho \quad \Pr[V(x, \rho) = 1] \leq \frac{1}{3}.$$

Given conditions are also referred as completeness and soundness. This definition mirrors the structure of **NP**, where a classical proof can be verified efficiently, but in **QMA**, the proof is a quantum state rather than a classical string. **QMA-hard** problems are at least as hard as any problem in **QMA** under polynomial-time quantum reductions. A problem is **QMA-complete** if it is both in **QMA** and **QMA-hard**.

While problems in **BQP** are efficiently solvable on quantum computers, **QMA-hard** problems, such as lattice-based problems, are believed to be intractable even for quantum adversaries. Consequently, post-quantum cryptographic schemes rely on assumptions that their underlying problems remain hard in both worst-case and average-case settings, ensuring resilience against quantum attacks.

3 Classical Cryptography

This section explores key concepts in classical cryptography, including cryptanalysis, Shannon's perfect secrecy, and the unbreakable yet impractical one-time pad. It also examines critical elements like key length, hash functions, and prominent systems such as RSA and Diffie-Hellman key exchange, while addressing their quantum vulnerabilities, highlighting the need for advanced cryptographic practices in the quantum era. The discussion references foundational texts, notably *Understanding Cryptography* by Paar and Pelzl [28] and *An Introduction to Mathematical Cryptography* by Hoffstein et al. [4].

The figure below illustrates the fundamental diagram of cryptography, depicting two parties, Alice and Bob, who wish to communicate. Two channels connect them: one secure and one insecure. The insecure channel is susceptible to adversarial access, allowing an entity—commonly referred to as *Eve* the Eavesdropper—to perform either passive or active attacks. A passive attacker merely listens to the communication, whereas an active attacker may attempt to modify, inject, or disrupt messages. To distinguish between these behaviors, we will use the term *Eve* to refer generically to any adversary, with context determining whether the attack is passive or active. This section is based on widely used teaching materials on cryptography, including *Understanding Cryptography: A Textbook for Students and Practitioners* by Paar and Pelzl [28], *An Introduction to Mathematical Cryptography* by Hoffstein, Pipher, and Silverman [4].

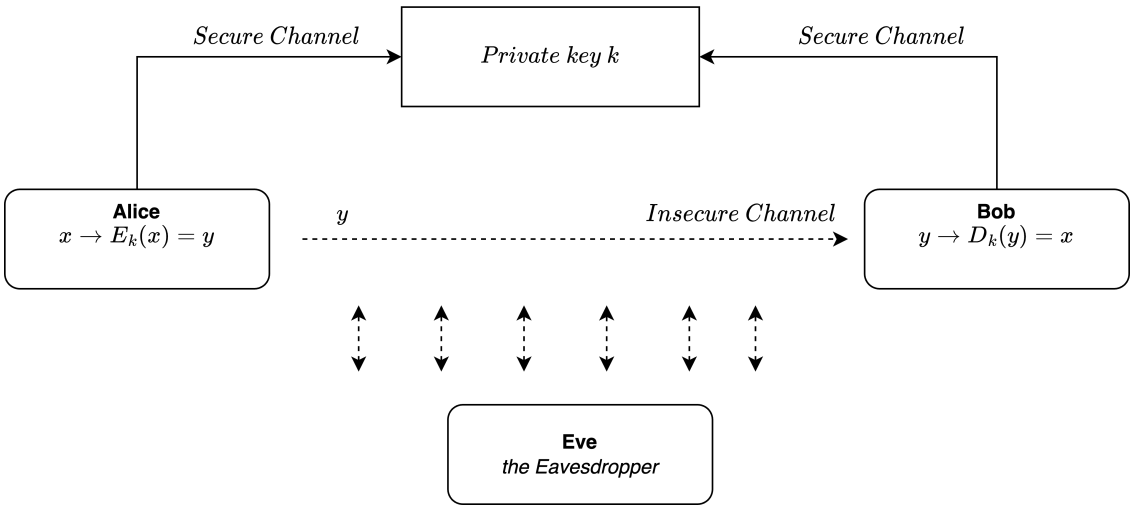

Figure 3: Exploring the Classic Setup of Cryptography

As depicted in the figure, when Alice wants to communicate with Bob over the insecure channel, she sends the value "y", obtained by encrypting her message $E_k(x) = y$. To comprehend the meaning of the received text, Bob attempts to compute the function $D_k(y) = x$. Here E_k and D_k represent the encryption and decryption keys, respectively. The terminology utilized here has been adopted from the research paper in which Ron Rivest, Adi Shamir, and Leonard Adleman introduced the RSA algorithm in 1978 [29], which would subsequently become known by the initials of their

names. Assuming that the system under consideration is secure, the collective operations undertaken to understand the content of the messages sent to this system are referred to as *cryptanalysis*. The term *attack* is used to denote all attempted cryptanalysis methods. Formally, a cryptosystem is a quintuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where:

- $\mathcal{P}$ symbolizes the space for messages or the collection of all potential **plaintexts**;
- $\mathcal{C}$ embodies the space for encrypted messages, the constituents of which are termed as **ciphertexts** or **cryptotexts**;
- $\mathcal{K}$ represents the set containing all feasible keys or the **key space**;
- For each k that is an element of $\mathcal{K}$, an encryption function $E_k : \mathcal{P} \to \mathcal{C} \in \mathcal{E}$ and a corresponding decryption function $D_k : \mathcal{C} \to \mathcal{P} \in \mathcal{D}$ are determined. These functions satisfy the fundamental decryption-encryption property: for every x from $\mathcal{P}$, it holds that $D_k(E_k(x)) = x$.

The field of cryptography is divided into two fundamental categories: **symmetric** and **asymmetric cryptography**, and their difference lies in the key infrastructure utilized for encryption and decryption. Informally, the difference lies within the number of keys. In symmetric cryptography, a single key is enough for encryption and decryption purposes. On the other hand, asymmetric cryptography encompasses two keys, namely **private key** and **public key**, where the private key is kept as secret, and the public key is made publicly available for anyone, even for *Eve*.

Definition 30. (Kerckhoffs's Desideratum)

"An ideal cryptographic system ought to maintain its security integrity even under the scenario where an adversary has comprehensive knowledge of the system's operational parameters, except the secret key. In particular, the cryptosystem's security should remain unbreached, even when the adversary possesses knowledge of both the encryption and decryption methodologies."
Kerckhoff, [30]

3.1 Cryptanalysis

If we disregard Kerckhoff's Principle, we might consider systems secure under the assumption that the adversary lacks pertinent information. However, the security of these systems is jeopardized when the adversary acquires knowledge about the system and is able to formulate it mathematically. This demonstrates the vulnerability of systems relying on *security through obscurity*, reinforcing the importance of designing cryptographic systems such that their security relies on the secrecy of the key rather than the secrecy of the system itself. For the following cryptanalysis methods, we will assume that *Eve* has a reasonable amount of computational resources. To express more explicitly, *Eve* can only employ a probabilistic polynomial time algorithm. For the purpose of facilitating comprehension, we will utilize the Affine cipher in forthcoming examples. This choice is made in consideration of its straightforward nature, which aids in providing a more accessible understanding of the subject. We will discuss two well-known cryptanalysis methods for this cipher.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Figure 4: Revisiting encoding table

For Affine Ciphers, both the plaintext and ciphertext spaces are congruent to the set of integers modulo 26, denoted as $P = C = \mathbb{Z}_{26}$. The key structure is represented as $k = (\alpha, \beta)$, such that α and 26 are coprime, i.e., $\gcd(\alpha, 26) = 1$.

Definition 31. (Affine Cipher) Given that α is a unit in $\mathbb{Z}_{26}$, a multiplicative inverse of α denoted as α^{-1} in $\mathbb{Z}_{26}$. The encryption process for Affine Cipher is then defined by the linear transformation,

$$y = E_k(x) = \alpha x + \beta \,(\text{mod}\, 26).$$

The decryption procedure is therefore conducted via the operation,

$$D_k(y) = \alpha^{-1}(y - \beta) \,(\text{mod}\, 26).$$

Example 5. Consider key $k = (7, 10)$. The Extended Euclidian algorithm gives us $7^{-1} = 15 \pmod{26}$. Based on the given information, provide transformation functions for the Affine Cipher and encrypt plaintext "UNIVERSITY" and decrypt "NUZCU".

Should we attempt to architect our transformation functions utilizing the provided variables, we would subsequently be confronted with the following mathematical representations:

$$y = E_k(x) = 7x + 10 \,(\text{mod}\, 26)$$

$$x = D_k(y) = 15(y - 10) \,(\text{mod}\, 26)$$

Let us bring to mind the numerical representations corresponding to the letters in the alphabet, which are as follows 4:

Applying the encryption function $E(x) = 7x + 10 \,(\text{mod}\, 26)$, we will get:

$$U \implies 20 : E(20) \equiv 20 \to U$$

$$N \implies 13 : E(13) \equiv 23 \to X$$

$$I \implies 8 : E(8) \equiv 14 \to O$$

$$V \implies 21 : E(21) \equiv 1 \to B$$

$$E \implies 4 : E(4) \equiv 12 \to M$$

$$R \implies 17 : E(17) \equiv 25 \to Z$$

$$S \implies 18 : E(18) \equiv 6 \to G$$

$$I \implies 8 : E(8) \equiv 14 \to O$$

$$T \implies 19 : E(19) \equiv 13 \to N$$

$$Y \implies 24 : E(24) \equiv 22 \to W$$

Thus corresponding ciphertext for *UNIVERSITY* is *UXOBMZGONW*. Similarly, if we apply the same logic on *NUZCU* we will see that it resolves to *TURKU*.

The Known-Plaintext Attack

The Known-Plaintext Attack (KPA) is a cryptanalytic method where an attacker, having access to both the plaintext (unencrypted message) and its corresponding ciphertext (encrypted message), seeks to discover the secret key used in the encryption process. This attack leverages the relationship between the plaintext and its ciphertext to identify the cryptographic algorithm's vulnerabilities and potentially predict the transformation of other plaintexts.

Example 6. Suppose we know that the encryption of *UNIVERSITY* is *UXOBMZGONW*. Break the system by employing Known Plaintext Attack.

Using the encryption of letters U and T, for example $UT \to UN$, we obtain equations:

$$\begin{cases} 20\alpha + \beta \equiv 20 \,(\mathrm{mod}\, 26) \\ 19\alpha + \beta \equiv 13 \,(\mathrm{mod}\, 26) \end{cases} \iff \begin{cases} \alpha = 7 \\ \beta = 10 \end{cases}$$

To solve the system of congruences: $20\alpha + \beta \equiv 20 \pmod{26}$ and $19\alpha + \beta \equiv 13 \pmod{26}$, we need to subtract the second equation from the first equation to eliminate β: $\alpha \equiv 7 \pmod{26}$. Then if we substitute $\alpha = 7$ into first equation to find β:$\beta \equiv 10 \pmod{26}$. Therefore, since the solution to the system of congruences is $\alpha \equiv 7 \pmod{26}$ and $\beta \equiv 10 \pmod{26}$, we conclude that the key used is $k = (7, 10)$.

Cryptotext-Only Attack

The primary objective of this approach is to leverage the statistical characteristics of the language utilized in the cryptographic text. It is important to highlight that each letter is consistently encrypted by another letter, irrespective of its content. This phenomenon forms the basis for the application of the so-called frequency analysis technique. To elaborate on the process:

1. The initial step involves calculating the frequencies of individual letters within the cryptographic text.

2. These computed frequencies are then compared with the well-established frequencies of letters in the English language. By making educated assumptions about the correspondences, one can proceed with the next step.

3. Building upon the assumption, the cryptographic key is determined, akin to the methodology employed in Known-Plaintext Attacks.

4. To validate the accuracy of the prediction, it is imperative to decrypt the text using the derived key. In case of discrepancies or uncertainties, one may need to formulate new assumptions and repeat the process accordingly.

Letter	**Percentage**	**Letter**	**Percentage**
E	11.2%	M	3.0%
A	8.5%	H	3.0%
R	7.6%	G	2.5%
I	7.5%	B	2.1%
O	7.2%	F	1.8%
T	6.9%	Y	1.8%
N	6.7%	W	1.3%
S	5.7%	K	1.1%
L	5.5%	V	1.0%
C	4.5%	X	0.3%
U	3.6%	Z	0.3%
D	3.4%	J	0.2%
P	3.2%	Q	0.2%

Table 3: Frequency Analysis of English Letters [31]

Example 7. Using letter frequency analysis and Cryptotext-Only Attack, attempt to decipher the following cryptotext generated by an affine cipher: **'Ote xmf kwq wie zfg kzm kak oxg ngy omx ymy hee gmg yom xym qcm qkj knk nuzc'**

When enumerating individual characters of a given cryptotext in *English*, it becomes apparent that the letters "M" and "K" exhibit the highest frequencies. Our initial conjecture posits that "M" corresponds to the encryption of "E" while "K" aligns with the encryption of "A". Subsequently, we proceed to resolve the resultant system of equations:

$$\begin{cases} 4\alpha + \beta \equiv 12 \,(\mathrm{mod}\, 26) \\ \beta \equiv 10 \,(\mathrm{mod}\, 26) \end{cases} \iff \begin{cases} \alpha = 7 \\ \beta = 10 \end{cases}$$

Deciphering based on letter frequency analysis yields key candidates "ifo, ned, aym," which exhibit striking resemblance to the plaintext "If one day m.." The decryption process may be further pursued by employing the key $k = (7, 10)$, leading to subsequent results.

"If one day, my words are against science, choose science. M. Kemal Ataturk"

Since we managed to find a meaningful text in English language we can consider that crypanalysis for finding secret key is performed successfully.

3.2 Shannon's Perfect Secrecy

The communication theory was pioneered by Claude Shannon in the late 1940s [15], catering not only to the demands of cryptography but also encompassing a wider application domain, error-correcting codes theory. A cryptosystem exhibits *computational security* when the amount of computational resources required to breach the system within a reasonable timeframe surpasses the capabilities of any conceivable adversary. In essence, this implies that even an adversary like Marvin, equipped with state-of-the-art technology and employing the most advanced algorithms available, would be unable to compromise the system effectively. On the other hand, a cryptosystem achieves *unconditional security* if it remains impervious to attacks even from an adversary possessing limitless computing power and an abundance of other resources. In this scenario, the system would withstand any possible attempt at compromise, regardless of the adversary's computational prowess or access to cutting-edge technology.

Definition 32. (Perfect Secrecy) A cryptosystem is considered perfectly secure if $p(x|y) = p(x)$ for all $x \in \mathcal{P}$ and $y \in \mathcal{C}$ where the conditional probability of observing a plaintext element x, given a ciphertext y, is denoted as $p(x|y)$.

Theorem 1. *Assume that we are given a cryptosystem with perfect secrecy. Then, for every pair of plaintext $x \in \mathcal{P}$ and ciphertext $y \in \mathcal{C}$ such that $\Pr[\mathcal{C} = y] > 0$, there must exist a key $k \in \mathcal{K}$ such that $E_k(x) = y$. Consequently, $|\mathcal{K}| \geq |\mathcal{C}|$. Notwithstanding, as the function used for encryption must be injective, we have $|\mathcal{C}| \geq |\mathcal{P}|$. Hence, $|\mathcal{K}| \geq |\mathcal{C}| \geq |\mathcal{P}|$.*

Remark 9. Note that for each fixed key $k \in \mathcal{K}$, the encryption function E_k must be injective; that is, it should not map two distinct plaintexts to the same ciphertext. Otherwise, decryption would become ambiguous.

Shannon secrecy asserts that the distribution of the message after observing the ciphertext is the same as its distribution before observing the ciphertext. In simpler terms, observing the ciphertext provides no more information than not seeing it at all.

3.3 One-time Pad

Interestingly, there is a scheme that is *Perfectly Secret*, referred as One-time Pad or *Vernam's Cipher* [32] and the simple idea behind is having a complete randomness on the message while still keeping it reversible.

Definition 33. Consider an integer $n \geq 1$. The spaces for keys, plaintexts, and ciphertexts are all composed of n-bit strings, denoted as $\mathcal{K} = \mathcal{P} = \mathcal{C} = \{0,1\}^n$. The encryption scheme operates as follows:

- $\mathsf{Generate}$ produces a key k by selecting uniformly at random from $\{0,1\}^n$.
- $\mathsf{Encrypt}_k(p)$ takes a plaintext p and outputs a ciphertext $c = p \oplus k \in \{0,1\}^n$, using the bitwise exclusive-or (XOR) operation.
- $\mathsf{Decrypt}_k(c)$ takes a ciphertext c and returns the plaintext $p = c \oplus k \in \{0,1\}^n$, also using the bitwise exclusive-or (XOR) operation.

It is crucial to acknowledge that the one-time pad encryption scheme imposes significant practical constraints. Foremost among these is the necessity for the key to match the message in length. This requirement renders the one-time pad impractical for real-world applications. When dealing with lengthy messages, the feasibility of Alice and Bob establishing and securely maintaining such an extensive key in advance of communication becomes markedly impractical. Limitations similar to this lead us to think the alternate *acceptable* security levels are good enough even if they are not *perfectly secure*. On the other hand, there are other limitations that favor cryptographers. For instance, there are fundamental limitations to the computational capacity within the physical universe, such as the expected lifespan of our sun, 10^{10} years [33]. At this juncture, cryptography utilizes understanding what is difficult and what is easy to accomplish in areas such as problem complexity, time complexity, and space complexity, thereby presenting a security that is close to perfect, with defined bounds.

Definition 34. (Entropy) Let us consider a random variable, denoted as X, which can take on various values denoted as x_i with corresponding probabilities $p_i = P(x_i)$ for $i = 1, \ldots, n$. It is important to note that the sum of all these probabilities equals 1 ($\sum_{i=1}^{n} p_i = 1$). The entropy of X, denoted as $H(X)$, is given by the formula:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i),$$

where $\log_2(0) = 0$.

3.4 Key Length

Cryptographic key sizes are critical in ensuring the security and confidentiality of encrypted information. The cryptographic key, essentially a series of random bits, varies in length, and this length, referred to as the key size, establishes the total possible combinations. An increased bit count consequently yields a higher potential

for key combinations, thereby escalating the challenge for any unauthorized party attempting to deduce the key and decrypt the information. Classical cryptographic techniques such as RSA and Elliptic Curve Cryptography (ECC) base their security on the intricate nature of certain mathematical tasks. Specifically, the security of RSA is tied to the complexity of factorizing substantial prime numbers, while ECC security is associated with the difficulty of the elliptic curve discrete logarithm problem. As computational prowess continues to expand, so must the length of cryptographic keys to maintain commensurate levels of security. The table referencing NIST's recommendations for key sizes outlines the evolution of various parameters for cryptographic key sizes over the years.

α	β	γ	δ	ϵ	ζ
80	Up to 2010	1024	160	1024	160
112	Up to 2030	2048	224	2048	224
128	Beyond 2030	3072	256	3072	256
192	Beyond 2030	7680	384	7680	384
256	Beyond 2030	15360	512	15360	512

Table 4: NIST key size recommendations [34]

In the context of this discussion, the symbols in the table α, β, γ, δ, ϵ, and ζ represent various cryptographic parameters:

- α: The alpha symbol is utilized to represent the equivalent symmetric key size, effectively characterizing the numerical strength of the symmetric key within the cryptographic system.
- β: The beta symbol signifies the validity period of the cryptographic process, marking the time frame within which the cryptographic parameters maintain their legitimacy and effectiveness.
- γ: Employed to denote the RSA-based modulus size, γ implies the number of bits in the modulus that are used for RSA cryptographic operations.
- δ: Signifying the discrete-log-based key size, δ represents the magnitude of the key within the discrete logarithmic cryptographic framework.
- ϵ: Epsilon represents the group size for discrete-log-based cryptography, providing an indication of the size of the finite group utilized within the cryptographic process.
- ζ: Finally, Zeta is used to represent the elliptic-curve-based key size, encapsulating the size of the key used in elliptic curve cryptographic schemes.

3.5 Hash Functions

In this section, we investigate a unique subset of cryptographic functions, namely the **hash functions**. Hash functions operate without any key requirement. A hash function, denoted as h, possesses the following distinctive properties:

- **Compression:** The output of the hash function $h(x)$ retains a fixed length, such as 160 bits, irrespective of the length of the input x.
- **Efficiency:** The calculation of $h(x)$ is notably effortless and swift, assuming x is known.

Functions adhering to these criteria find utility in non-security-related applications, such as searching within large databases, an instance of which being hash tables. For the purposes of cryptographic applications, a hash function is additionally necessitated to be *one-way*, signifying:

- **Preimage resistance:** When given a value y, it is computationally challenging to identify any x that satisfies $h(x) = y$.

Certain use cases necessitate that the hash function satisfy additional conditions:

- **2nd-preimage resistance:** For a known x, it is computationally arduous to locate any distinct x_0 such that $h(x) = h(x_0)$.
- **Collision resistance:** The discovery of any two different x and x_0 such that $h(x) = h(x_0)$ is computationally infeasible.

While the traditional textbook definition might suggest a hash function needs to have the property of compression, contemporary practices have evolved to recognize the validity of *extendable hashing algorithms*, such as **SHAKE128** and **SHAKE256** [35].

Definition 35. (Birthday Paradox) What is the probability that any two out of x randomly chosen individuals share the same birthday? Alternatively, what is the minimum value of x required to ensure that this probability is at least 50%? The solution to the latter question is strikingly small, with $x = 23$. As a result, this phenomenon is commonly known as the birthday paradox.

What is the minimum size Y should have to prevent collisions from being discovered by brute force? A lower bound for the required |Y| is determined by analyzing the so-called birthday attack. [36]

Example 8. Let us consider a hash function $h : X \to Y$, where $|Y| = n$. Additionally, we assume that the distribution of $h(X)$ is uniform, For each $y \in Y$, $Prob(h(x) = y) = \frac{1}{n}$ for a randomly selected $x \in X$. Our objective is to assess the probability of encountering a collision in h by randomly selecting k elements $x_1, \ldots, x_k \in X$ and comparing their corresponding hash values, $h(x_i)$.

The probability of no collisions among the k values can be expressed as follows:

$$p = \left(1 - \frac{1}{n}\right)\left(1 - \frac{2}{n}\right)\ldots\left(1 - \frac{k-1}{n}\right)$$

$$p = \prod_{i=1}^{k-1} \left(1 - \frac{i}{n}\right)$$

When x is small, we can observe the approximation $e^{-x} \approx 1 - x$, which allows us to obtain the following result.

$$p \approx \prod_{i=1}^{k-1} e^{-\frac{i}{n}}$$

$$p = e^{-\frac{k(k-1)}{2n}}$$

$$k^2 - k = 2n \ln\left(\frac{1}{1-\varepsilon}\right)$$

The probability of encountering at least one collision is expressed as $1 - p$. We can now proceed to solve the equation $1 - p = \varepsilon$ to obtain the solution and the following estimation.

$$k \approx \sqrt{2n \ln\left(\frac{1}{1-\varepsilon}\right)}$$

If we set $\varepsilon = \frac{1}{2}$, we get $\sqrt{k} \approx 1.17\sqrt{n}$. Put differently, when $\sqrt{n}$ random hash values are computed, there exists approximately a 50% likelihood of encountering a collision among them. For instance, if $n = 365$, our estimation suggests that $k \approx 22.3$. This result is closely matching the classical birthday paradox. In conclusion, in order to render the occurrence of collisions practically infeasible, the cardinality of the set Y denoted by $n = |Y|$ must be chosen such that it becomes practically unfeasible to compute $\sqrt{n}$ hash values, $n \geq 2^{128}$. To mitigate the feasibility of such collisions in cryptographic applications, one must ensure that computing $\sqrt{n}$ hash values is beyond current computational capabilities. This leads to the requirement $n = |Y| \geq 2^{128}$. A 128-bit key provides a full 128-bit security level, requiring approximately 2^{128} operations to recover the key. With this understanding, the rationale behind the values presented in Table 4 should now be clearer.

3.6 RSA Cryptosystem

As we mentioned earlier, the idea of public-key cryptography is considered to be founded by Diffie and Hellman. In addition, certain sources attribute the invention of this system[6] to Ralph C. Merkle [37], considering the the interviews and discussions demonstrate evidence of prior discovery of the research in the publication process. Furthermore, the British Intelligence Agency is also claiming that the first representation belongs to themselves [38]. After presenting all the different concepts, let us now focus on the two fundamental problems addressed by public key cryptography, regardless of the variations of these concepts.

Definition 36. (Distributing keys over insecure channel) How to handle the distribution of keys belonging to a specific cryptosystem when communication partners cannot meet in person or use a *secure channel* for key transfer.

[6]Merkle's Puzzles

Definition 37. (Signature Problem) Is it possible to create unforgeable signatures that will help to protect the authenticity of digital messages?

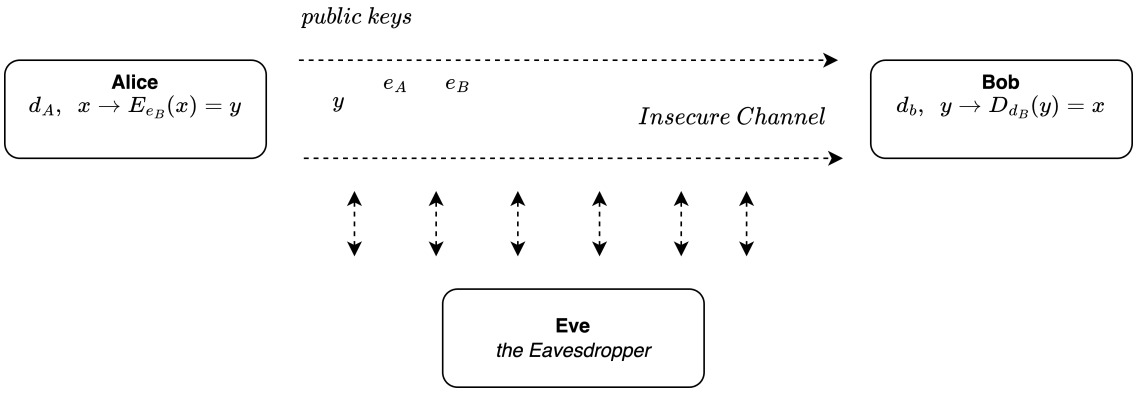

Figure 5: Exploring the Public-Key Cryptography

The illustration indicates a notable distinction between the configurations of public-key cryptosystems and symmetric cryptosystems. In public-key systems, each user, denoted as A, possesses a distinct public encryption key (e_A) and a corresponding private decryption key (d_A). Whenever messages are intended for A, they are encrypted using her public key, e_A. On the other hand, when A sends a message to another user, B, she utilizes B's public encryption key, e_B. Importantly, the private decryption key, d_A, exclusively known to A, safeguards the messages' confidentiality, making it infeasible for external entities to decipher the messages directed to A. Mathematical security of the system relies on **trapdoor functions**, i.e., variants of **one-way functions**. Assuming one-way functions exist, consider a one-way function f, where $e = f(d)$, and by the nature of one-way functions, we may publish e publicly. Two of the most renowned public-key cryptosystems have been constructed using this approach. On one hand, RSA utilizes integer multiplication as a candidate one-way function since efficient factorization of large integers that are product of two primes is still an open problem. Let n be the product of two prime numbers, p and q. It follows that $\varphi(n) = (p-1)(q-1)$, where φ is Euler's totient function. The RSA public key consists of a pair (n, e), where e is coprime to $\varphi(n)$ ($\gcd(e, \varphi(n)) = 1$). The corresponding private key is a pair (n, d), where d is the modular multiplicative inverse of e modulo $\varphi(n)$ ($ed \equiv 1 \mod \varphi(n)$). The encryption and decryption functions are defined as follows:

$$E(x) = E_{n,e}(x) = x^e \mod n$$

$$D(x) = D_{n,d}(y) = y^d \mod n$$

The term e is often referred to as the encryption exponent, while d is known as the decryption exponent.

Euler's theorem states that $x^{\varphi(n)} \equiv 1 \mod n$. Consequently, decrypting the ciphertext $D(E(x))$ simplifies to $x^{ed} \equiv x^{1+k\varphi(n)} = x \cdot x^{k\varphi(n)} \equiv x \mod n$.

Hence, decrypting the ciphertext indeed yields the original plaintext, demonstrating the efficacy of the RSA cryptosystem.

Example 9. Let us consider the actions taken by Bob, a proficient mathematician, to create and use a cryptographic system for secure communication. Bob carefully chooses two prime numbers, $p = 17$ and $q = 11$. These values are then used to compute the modulus $n = 187$, and the Euler's totient function, denoted as $\varphi(n)$, is evaluated to be 160 using the formula $\varphi(n) = (p-1) \times (q-1)$. Next, Bob proceeds to select a public exponent $e = 7$, ensuring that it satisfies the condition of being coprime to n. This requirement is confirmed using Euclid's algorithm, which guarantees that e and n have no common factors other than 1. With e chosen as the public exponent, Bob then calculates its modular multiplicative inverse modulo $\varphi(n)$, resulting in the private exponent $d = 23$. Bob adopts a standard asymmetric encryption scheme, where he publicly reveals the numbers $(n, e) = (187, 7)$ as his public key. The public key is used by anyone, including Alice, who wishes to send Bob a secret message. However, Bob prudently retains the private key $(n, d) = (187, 23)$ exclusively to himself to ensure the confidentiality of encrypted messages. Suppose Alice desires to send a confidential message represented by the plaintext $x = 19$. She leverages Bob's public key to perform encryption, applying the formula:

$$E(x) = 19^7 \equiv 145 \pmod{187}$$

After encrypting the plaintext, Alice transmits the ciphertext $y = 145$ to Bob. Upon receiving the ciphertext, Bob employs his private key to decrypt the encrypted message. The decryption process is carried out using the formula:

$$D(x) = 145^{23} \equiv 19 \pmod{187}$$

In summary, Bob's cryptographic system employs the mathematical properties of prime numbers, modular arithmetic, and asymmetric encryption to enable secure communication with Alice and other users. To achieve a acceptable level of security in cryptographic systems, NIST experts recommend employing a modulus with a binary length of approximately 2048 bits until at the end of 2029. This entails selecting prime numbers p and q in such a way that to have 617 digits. Such a choice of large prime numbers ensures a robust and secure encryption process, making it challenging for adversaries to factorize the modulus and thereby maintain the confidentiality of encrypted data. For illustrative purposes, let us consider a 617-digit integer presented below:

31415926535897932347162095952463104220805947587884797123123123124312432423545431312312312432534645756867978087098765241312312431312324535465890987654323456784424234234234245756879892342423424234234234234234545345345345313159420156321151459687022826496284804703581636775050768511751113837703165649976366298700846652547590244061430956115502883439669230154491854687745297902659305004496535344647645564903545917919984322403206146604875324137410411611611211558474710111046119105107116105111110971141214611111410347119105107105479798971101001111109510411111210195971081089512110195119104111951011101161011149510410114101114101

Figure 6: First 617 Digits of π

Remark 10. In 2020, the factorization of a 250-digit number was accomplished through the algorithmic framework of the Number Field Sieve, a task which consumed **2700 core-years**[7] of computation in total where RSA-250 sieving take 2450 physical core years and RSA-250 Matrix takes 250 physical core years. [39].

3.7 Diffie Hellman Key Exchange

Consider a prime number p and the multiplicative group $\mathbb{Z}_p$. This group has $p-1$ elements and is cyclic. Let α be a generator of this group. Now, for any element $\beta \in \mathbb{Z}_p$, we define the discrete logarithm of β with respect to the base α, denoted as $\log_\alpha \beta$. This discrete logarithm is a unique integer x satisfying the congruence $\alpha^x \equiv \beta \pmod{p}$ and fulfilling the condition $0 \leq x < p-1$.

Definition 38. (Discrete Log Problem) Let G be a finite cyclic group generated by primitive root g, for instance, $G = (\mathbb{Z}/p\mathbb{Z})^*$ Given $b \in G$ and a power a of b, find a positive integer n such that $b^n = a$.

Since $t_b^a = (g^b)^a = (g^a)^b = t_{ba}$, this ensures that Alice and Bob obtain the same value. Furthermore, it is evident that both Alice and Bob can efficiently carry out the required computations. The security of the Diffie-Hellman Key Exchange protocol relies on Alice and Bob being able to exchange messages safely in public. Eve's presence as a potential eavesdropper does not compromise the system. If p is chosen to be sufficiently large, there appears to be no efficient method for Eve to compute k from the given values p, g, t_a, and t_b. This challenge is known as

[7] A core-year is a unit of measurement that represents utilization of a computer core uninterruptedly for 1 year

the Diffie–Hellman Problem (DHP). Additionally, if Eve could compute the discrete logarithm of t_a (or t_b), she would be able to deduce k_1 as well.

Algorithm 1 Diffie-Hellman Key Exchange Algorithm

Require: Consider a prime number p belonging to the set of primes $\mathbb{P}$, along with a primitive root modulo p, denoted as g.
Ensure: Shared secret, i.e., private, key k
Alice selects a private key $a \in \{2, ..., p-2\}$
Alice computes $A \leftarrow g^a \mod p$
Alice sends A to Bob
Bob selects a private key $b \in \{2, ..., p-2\}$
Bob computes $B \leftarrow g^b \mod p$
Bob sends B to Alice
Alice computes shared key $k_A \leftarrow B^a \mod p$
Bob computes shared key $k_B \leftarrow A^b \mod p$
If $k_A = k_B$ the shared key k is established, else go to first step.

If exponentiation is implemented correctly then both parties *already* share the identical value k and no further check is required. In practice, however, explicit or implicit key-confirmation is added to detect implementation errors or man-in-the-middle attacks before any protected data are exchanged. The Diffie-Hellman key exchange is foundational to various cryptosystems, providing a robust mechanism for secure communication even in the presence of potential eavesdroppers.

3.8 Quantum Vulnerabilities of Classical Methods

Since its inception, the RSA cryptosystem has fascinated cryptographers, becoming an enduring symbol of cryptographic security. While mathematical attacks like differential and algebraic cryptanalysis have had limited success in controlled settings, RSA's security fundamentally relies on the difficulty of integer factorization. Algorithms such as the Quadratic Sieve, Elliptic Curve Method, and General Number Field Sieve (GNFS) have improved over time but remain inefficient for breaking real-world RSA keys.

Despite RSA's four decades of resilience, quantum computing threatens its viability. Shor's algorithm, capable of efficient integer factorization, poses a serious risk if large-scale quantum computers become practical. Studies estimate that breaking a 2048-bit RSA key requires around 4096 qubits, with error correction dramatically increasing physical qubit requirements. It is also important to highlight the fact that these refer to the logical qubits which considered ideal. Shor's algorithm for instance requires each qubit and error gate to be perfectly error free, but on the other hand real hardware implementations comes with noise. Every logical qubit must be encoded in thousands of physical qubits so that error-correction can keep the computation reliable for billions of operations. In 2021 a potential attack has been demonstrated using 20 million noisy qubits [40], while Fujitsu estimated that a 2048-bit key would require 10,000 qubits and a 104-day computation on a fault-tolerant

quantum computer. While current limitations keep RSA secure, the prospect of "harvest-and-decrypt" attacks—where adversaries store encrypted data for future quantum decryption—has alarmed the cryptographic community.

To counter this, NIST launched the Post-Quantum Cryptography (PQC) Standardization Process, emphasizing security, cost, and implementation feasibility. In 2022, four algorithms were selected: CRYSTALS-KYBER (key exchange), CRYSTALS-Dilithium (digital signatures), FALCON (for compact signatures), and SPHINCS+ (for diversity beyond lattice-based cryptography). The majority of PQC candidates rely on lattice problems, such as Learning With Errors (LWE) and Learning With Rounding (LWR), which currently lack efficient quantum-breaking algorithms.

Among these, CRYSTALS-KYBER stands out as the preferred choice for standard encryption, offering small key sizes and fast performance.

4 Lattices

As the 1700s neared its conclusion, famous mathematicians such as Joseph Louis Lagrange, Carl Friedrich Gauss, often regarded as the *Princeps Mathematicorum*, and later Hermann Minkowski, made foundational contributions to the study of lattices. Their work established critical mathematical frameworks, including geometric interpretations of lattice structures, which continue to influence modern applications. While early results focused on the structural properties of lattices, the late 20th century saw a dramatic shift towards algorithmic applications, particularly in cryptography.

During the mid-1990s, Ajtai developed an innovative *worst-case to average-case reduction* for lattice problems. He proved that if one can efficiently solve randomly chosen (average-case) instances, then every (worst-case) instance can also be solved efficiently, showing that the problems are just as hard on average as in the worst case. This breakthrough led to the creation of a cryptographic one-way function based on worst-case hardness conjectures [12]. Building on this, Ajtai and Dwork proposed a public-key encryption scheme with similar security assurances [41]. Although these schemes were theoretically significant, they suffered from inefficiency and complexity, making them impractical for widespread use.

Simultaneously, Hoffstein, Pipher, and Silverman introduced the NTRU public key encryption scheme, along with a related digital signature scheme [42]. NTRU encryption demonstrated practical efficiency, offering faster performance compared to contemporary schemes. However, it lacked rigorous theoretical security guarantees, leading to trade-offs between efficiency and provable security. Over time, cryptanalysts uncovered vulnerabilities in certain parameter sets, leading to the compromise of some NTRU-based signature schemes. Despite these challenges, NTRU encryption has demonstrated resilience in an asymptotic sense and remains a practical option for lattice-based cryptography. Lattices, with their rich mathematical structure and versatility, have influenced diverse areas such as cryptanalysis, coding theory, optimization, and cryptography from their historical roots with Lagrange, Gauss, and Minkowski to their modern role in cryptographic breakthroughs like Ajtai's reductions.

4.1 Fundamental Properties of Lattices

To establish a foundation for the study of lattices, we begin by revisiting the essential definitions and structural properties central to lattice theory. This groundwork is particularly important because it underpins lattice-based cryptography, where the algebraic and geometric properties of lattices are crucial for ensuring both security and efficiency. Recall that a set is *discrete* if its points are isolated, and a *subgroup* is a subset that is closed under both addition and taking inverses.

Definition 39 (Discrete Additive Subgroup)**.** Let $G \subset \mathbb{R}^n$ be a subset. We say that G is an *additive subgroup* if it is closed under addition and negation; that is, for all $x, y \in G$, both $x + y$ and $-x$ belong to G. Moreover, G is said to be *discrete* if for

every point $x \in G$ there exists an $\varepsilon > 0$ such that the open ball

$$B(x, \varepsilon) = \{y \in \mathbb{R}^n : \|y - x\| < \varepsilon\}$$

satisfies

$$B(x, \varepsilon) \cap G = \{x\}.$$

In n-dimensional Euclidean space $\mathbb{R}^n$, an essential characteristic of lattices is *discreteness*, which ensures that lattice points are spaced apart, preventing any form of clustering or accumulation.

Definition 40 (Lattice)**.** A discrete additive subgroup of $\mathbb{R}^n$ is a *lattice*.

Let us understand this definition more closely. The term *additive subgroup* means that a lattice $\mathcal{L} \subseteq \mathbb{R}^n$ is closed under addition and contains the additive inverses of its elements. That is, if $\mathbf{v}, \mathbf{w} \in \mathcal{L}$, then $\mathbf{v} + \mathbf{w} \in \mathcal{L}$ and $-\mathbf{v} \in \mathcal{L}$ as well. The requirement that $\mathcal{L}$ is *discrete* ensures that the points of the lattice are isolated, meaning there is a minimal distance between any two distinct points in $\mathcal{L}$.

A lattice can alternatively, and more usefully for our purpose, be represented as the set of all integral linear combinations of m linearly independent vectors, known as a basis.

Definition 41 (Lattice Basis)**.** Let $\mathcal{L}$ be a lattice in $\mathbb{R}^n$. A set of m linearly independent vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m \in \mathbb{R}^n$ is called a *basis* of $\mathcal{L}$ if every point $\mathbf{x} \in \mathcal{L}$ can be expressed as:

$$\mathbf{x} = \sum_{i=1}^{m} x_i \mathbf{b}_i, \quad \text{where } x_i \in \mathbb{Z}.$$

Definition 42 (Equivalent Characterization of Lattice)**.** Let $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m$ be a set of m basis vectors in $\mathbb{R}^n$. The lattice generated by these basis vectors is the set:

$$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_m) = \left\{ \sum_{i=1}^{m} x_i \mathbf{b}_i : x_i \in \mathbb{Z} \right\},$$

where $x_i \in \mathbb{Z}$ are integer coefficients, and $\mathbb{Z}$ denotes the set of integers.

Remark 11. A lattice basis is not unique, as different sets of linearly independent vectors can generate the same lattice.

This definition highlights the algebraic structure of lattices and their geometric interpretation as a grid of points generated by integer linear combinations of the basis vectors. While these formal definitions capture the mathematical essence of a lattice, it may be helpful to develop an intuitive understanding of it. Informally, a lattice $\mathcal{L}$ in m-dimensional space can be viewed as a regular, repeating arrangement of points extending infinitely in all directions.[8]

For any lattice point $\mathbf{x} \in \mathcal{L}$, there exists some $\epsilon > 0$ such that the open ball of radius ϵ centered at $\mathbf{x}$ contains no other lattice points.

[8]The degenerate case $\{0\} \subset \mathbb{R}^m$ is also a lattice, although it consists only of the origin and is finite.

The open ball of radius ϵ centered at $\mathbf{x}$ is defined as:

$$B(\mathbf{x}, \epsilon) = \{\mathbf{y} \in \mathbb{R}^n \mid \|\mathbf{y} - \mathbf{x}\| < \epsilon\},$$

where $\|\cdot\|$ denotes the Euclidean norm.[9]

Thus, the discreteness of the lattice $\mathcal{L}$ can be formally expressed as:

$$\forall \mathbf{x} \in \mathcal{L}, \exists \epsilon > 0 \text{ such that } (B(\mathbf{x}, \epsilon) \cap \mathcal{L}) = \{\mathbf{x}\}.$$

Discreteness of lattices ensures that each lattice point can be uniquely identified within a sufficiently small neighborhood, as no two lattice points lie arbitrarily close to one another. This property is foundational for hard lattice problems that are central to lattice-based cryptography, such as finding the lattice point closest to a given target or identifying a non-zero lattice vector with the shortest length. The isolation of lattice points makes solving such problems computationally hard.

To better understand the flexibility and structure of lattices, we now discuss an important property related to their scaling, which is particularly useful when analyzing lattice behavior under transformations.

Remark 12. For any lattice $\mathcal{L} \subset \mathbb{R}^n$ and any nonzero scalar $\alpha \in \mathbb{R} \setminus \{0\}$, scaling the lattice by α produces another lattice $\alpha\mathcal{L}$, given by:

$$\alpha\mathcal{L} = \{\alpha\mathbf{x} \mid \mathbf{x} \in \mathcal{L}\}.$$

The new lattice $\alpha\mathcal{L}$ is a uniform scaling of $\mathcal{L}$ by a factor of $|\alpha|$.

Example 10. For instance, scaling the set of integers by 2 yields the set of even integers, $2\mathbb{Z}$. Let us remember that $\mathcal{L}$ qualifies as a subgroup if it encompasses the identity element $0 \in \mathbb{R}^n$, and for all $x, y \in \mathcal{L}$, both $-x \in \mathcal{L}$ and $x + y \in \mathcal{L}$.

Lemma 1. *Applying a linear transformation, in fact any linear transformation, to a lattice results in another lattice.*

Proof. Let $\mathcal{L}$ be a lattice in $\mathbb{R}^n$ generated by the basis vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m$, so that:

$$\mathcal{L} = \left\{ \sum_{i=1}^{m} x_i \mathbf{b}_i : x_i \in \mathbb{Z} \right\}.$$

Suppose $T : \mathbb{R}^n \to \mathbb{R}^n$ is a linear transformation. When T is applied to every vector in $\mathcal{L}$, the resulting set $T(\mathcal{L})$ becomes:

$$T(\mathcal{L}) = \left\{ \sum_{i=1}^{m} x_i T(\mathbf{b}_i) : x_i \in \mathbb{Z} \right\}.$$

The set $T(\mathcal{L})$ is also a lattice because the linearity of T ensures that any integer linear combination of the transformed basis vectors $T(\mathbf{b}_1), T(\mathbf{b}_2), \ldots, T(\mathbf{b}_m)$ results in an m-dimensional discrete additive subgroup of $\mathbb{R}^n$. Hence, the resulting set satisfies the definition of a lattice. □

[9]Throughout this thesis, we will use the Euclidean norm unless stated otherwise.

To further illustrate, consider the case when T is a scaling transformation. In this scenario, $T(\mathcal{L})$ represents a scaled version of $\mathcal{L}$, where all lattice points are uniformly expanded or contracted while preserving their relative arrangement. Similarly, if T is a rotation, the lattice points are rotated about the origin, yet the underlying discrete structure remains intact. These examples highlight how the action of linear transformations preserves the fundamental structure of the lattice, providing a foundation for further analytical exploration.

With this understanding of the geometric and structural properties of lattices under linear transformations, we now turn to their algebraic characterization via matrix representations. The matrix representation not only encapsulates the structure of a lattice in a compact form but also enables the use of linear algebraic tools for deeper analysis. This representation is indispensable for a variety of applications, including basis transformations, volume computations, and algorithmic approaches in computational settings.

4.2 Matrix Representation of a Lattice

In lattice theory, the matrix representation offers a concise and algebraically tractable way to describe the structure of a lattice. Recall that a lattice $\mathcal{L}$ is defined as the set of all integer linear combinations of a basis, which is a collection of linearly independent vectors $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m\} \subseteq \mathbb{R}^n$. Arranging these basis vectors as columns of a matrix yields the *basis matrix*, denoted as $B = [\mathbf{b}_1\, \mathbf{b}_2\, \cdots\, \mathbf{b}_m] \in \mathbb{R}^{n\times m}$.

Algebraic and geometric properties of a lattice are often understood through its basis and the matrix representation of that basis. To develop a deeper understanding, we define three fundamental concepts: the basis matrix, the span of the lattice, and the rank of the lattice. These definitions collectively provide insight into the structure, dimensionality, and space-filling properties of a lattice.

Definition 43 (Basis Matrix)**.** Let $B \in \mathbb{R}^{n\times m}$ be a matrix whose columns are the basis vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m$. If B has linearly independent columns, then B is called a *basis matrix*.

The basis matrix B is a compact representation of the lattice's generating set. The lattice generated by B is formally defined as:

$$\mathcal{L}(B) = \mathcal{L}(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m) = \{B\mathbf{x} \mid \mathbf{x} \in \mathbb{Z}^m\}$$

Definition 44 (Span of a Lattice)**.** Let $B \in \mathbb{R}^{n\times m}$ be a matrix whose columns $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m$ are vectors in $\mathbb{R}^n$. The *span of the lattice*, $\operatorname{span}(\mathcal{L}(B))$, is the vector space generated by the columns of B. Formally,

$$\operatorname{span}(\mathcal{L}(B)) = \operatorname{span}(B) = \{B\mathbf{y} \mid \mathbf{y} \in \mathbb{R}^m\}.$$

The span of a lattice defines the ambient vector space in which the lattice resides. Geometrically, it represents the smallest subspace of $\mathbb{R}^n$ that contains all lattice points. This concept is critical for analyzing how the lattice relates to its surrounding space and for determining its embedding within $\mathbb{R}^n$.

Definition 45 (Well-Ordered Basis)**.** Let $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m\}$ be a basis of $\mathbb{R}^n$. This basis is *well-ordered* if, for each $i = 1, 2, \ldots, n-1$, the following inequalities hold:

$$\|\mathbf{b}_i\| \;\leq\; \|\mathbf{b}_i - \mathbf{b}_{i+1}\| \;<\; \|\mathbf{b}_{i+1}\|.$$

Remark 13. Every m-dimensional subspace of $\mathbb{R}^n$ admits a well-ordered basis.

Definition 46 (Rank of a Lattice)**.** The *rank* of a lattice $\mathcal{L}(B) \subseteq \mathbb{R}^n$ is the dimension of the vector space spanned by its basis vectors. Equivalently, it is the rank of the basis matrix B. Formally,

$$\operatorname{rank}(\mathcal{L}(B)) = \dim\left(\operatorname{span}_{\mathbb{R}}(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m)\right).$$

We denote the rank of the lattice by k, where

$$k = \operatorname{rank}(B) \leq \min(m, n).$$

Throughout this work, lattice points are always represented as column vectors. When vectors are more conveniently written as rows, the transpose notation is used. For example, the vectors $\mathbf{b}_1$ and $\mathbf{b}_2$ can be equivalently expressed as

$$\mathbf{b}_1 = \begin{bmatrix} 1 \\ \\ 2 \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} 1 \\ \\ -1 \end{bmatrix},$$

or, using transpose notation, as

$$\mathbf{b}_1 = [1, 2]^\top, \quad \mathbf{b}_2 = [1, -1]^\top,$$

where $A^\top$ denotes the transpose of a matrix A.

Example 11 (Basis Matrix in $\mathbb{R}^{3\times 2}$)**.** Consider the matrix

$$B = \begin{bmatrix} 1 & 2 \\ \\ 0 & 1 \\ \\ 1 & 0 \end{bmatrix} \in \mathbb{R}^{3\times 2}.$$

The columns of B are the vectors

$$\mathbf{b}_1 = \begin{bmatrix} 1 \\ \\ 0 \\ \\ 1 \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} 2 \\ \\ 1 \\ \\ 0 \end{bmatrix}.$$

The span of the lattice generated by B is

$$\operatorname{span}(\mathcal{L}(B)) = \operatorname{span}(B) = \{c_1\mathbf{b}_1 + c_2\mathbf{b}_2 \mid c_1, c_2 \in \mathbb{R}\}.$$

In this example, the span is a 2-dimensional subspace of $\mathbb{R}^3$, since $\mathbf{b}_1$ and $\mathbf{b}_2$ are linearly independent. The rank of B is

$$\operatorname{rank}(B) = k = 2.$$

The lattice generated by B is

$$\mathcal{L}(B) = \left\{ B \begin{bmatrix} x_1 \\ \\ x_2 \end{bmatrix} \mid x_1, x_2 \in \mathbb{Z} \right\} = \left\{ \begin{bmatrix} x_1 + 2x_2 \\ \\ x_2 \\ \\ x_1 \end{bmatrix} \mid x_1, x_2 \in \mathbb{Z} \right\}.$$

Geometrically, the lattice forms a discrete grid of points in $\mathbb{R}^3$, confined to the 2-dimensional subspace spanned by $\mathbf{b}_1$ and $\mathbf{b}_2$. Each lattice point is an integer linear combination of these basis vectors, expressed as $\mathbf{v} = x_1\mathbf{b}_1 + x_2\mathbf{b}_2$ for $x_1, x_2 \in \mathbb{Z}$. This illustrates how the lattice structure and its rank reflect the dimensionality and subspace embedding defined by its basis.

Example 12. Let

$$B = \begin{bmatrix} 2 & 1 \\ \\ 0 & 3 \end{bmatrix}$$

be a 2×2 matrix with columns

$$\mathbf{b}_1 = \begin{bmatrix} 2 \\ \\ 0 \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} 1 \\ \\ 3 \end{bmatrix}.$$

Since the columns of B are linearly independent (as the determinant

$$\det(B) = 2 \cdot 3 - 0 \cdot 1 = 6 \neq 0),$$

B serves as a valid basis matrix.

The lattice $\mathcal{L}(B)$ generated by B consists of all integer linear combinations of the basis vectors and is defined as

$$\mathcal{L}(B) = \left\{ B\mathbf{x} \mid \mathbf{x} \in \mathbb{Z}^2 \right\}.$$

Explicitly, the lattice points take the form

$$\mathcal{L}(B) = \left\{ \begin{bmatrix} 2x_1 + x_2 \\ \\ 3x_2 \end{bmatrix} \mid x_1, x_2 \in \mathbb{Z} \right\}.$$

For instance, if $x_1 = 1$ and $x_2 = 1$, the corresponding lattice point is

$$B \begin{bmatrix} 1 \\ \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ \\ 3 \end{bmatrix}.$$

Since $\mathbf{b}_1$ and $\mathbf{b}_2$ are linearly independent, their span is the entire plane, i.e.,

$$\text{span}(\mathcal{L}(B)) = \text{span}(B) = \mathbb{R}^2.$$

As B has two linearly independent columns, the rank is

$$\text{rank}(\mathcal{L}(B)) = \text{rank}(B) = 2.$$

Definition 47 (Full-Rank Lattice)**.** A lattice $\mathcal{L} \subseteq \mathbb{R}^m$ is called *full rank* if its rank equals m. Equivalently, $\mathcal{L}$ is generated by m linearly independent vectors in $\mathbb{R}^m$, so that $\dim\big(\text{span}(\mathcal{L})\big) = m$.

Remark 14. Not all lattices are full rank, as seen in the examples: a lattice in $\mathbb{R}^3$ with rank 2 spans a 2-dimensional subspace, while a full-rank lattice in $\mathbb{R}^2$ spans the entire plane.

Theorem 2 (Rank Invariance)**.** *Let $\mathcal{L} \subseteq \mathbb{R}^m$ be a lattice. If $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_p\}$ and $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q\}$ are two bases of $\mathcal{L}$, then the number of basis vectors is the same, i.e.,*

$$p = q.$$

Proof. By definition, a lattice basis is a set of linearly independent vectors that generate all lattice points through integer linear combinations. Since both bases span the same lattice $\mathcal{L}$, they must also span the same subspace $V \subseteq \mathbb{R}^m$.

Let $d = \dim(V)$. From linear algebra, the number of vectors in any basis of a d-dimensional subspace must be exactly d. Therefore, both $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_p\}$ and $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q\}$ contain exactly d vectors. It follows that $p = q = d$, which proves the rank invariance. □

This result establishes that while every lattice has a basis, the number of basis vectors, its rank, remains fixed. However, the choice of basis itself is not unique. Unless stated otherwise, we assume throughout this thesis that all lattices are full rank. A full-rank lattice in $\mathbb{R}^n$ forms a discrete subgroup that spans the entire space.

Remark 15. For the purposes of this thesis, we primarily analyze full-rank lattices, where the rank of the lattice equals the dimension of the ambient space.

This assumption simplifies theoretical analysis and aligns with the requirements of many cryptographic applications, such as fully homomorphic encryption and worst-case to average-case reductions. While lattices with rank less than the dimension of the ambient space (e.g., rank-2 lattices in $\mathbb{R}^3$) are significant in other areas, such as subspace embeddings, they are beyond the scope of this thesis. The examples and results discussed earlier, such as lower-rank lattices, remain valid and consistent with this assumption. Before stating the following lemma, we recall:

Definition 48 (Unimodular Matrix)**.** A square integer matrix $U \in \mathbb{Z}^{n\times n}$ is called *unimodular* if $\det(U) = \pm 1$, equivalently if $U^{-1} \in \mathbb{Z}^{n\times n}$.

Lemma 2. *Bases B_1 and B_2 generate the same lattice $\mathcal{L}$ if and only if there exists a unimodular matrix $U \in \mathbb{Z}^{n\times n}$ such that $B_1 = B_2U$.*

Proof. Assume $\mathcal{L}(B_1) = \mathcal{L}(B_2)$, which means $B_1 \cdot \mathbb{Z}^n = B_2 \cdot \mathbb{Z}^n$. Consequently, each column of B_1 can be expressed as an integer combination of the columns of B_2, and vice versa. Therefore, there exist matrices $U, V \in \mathbb{Z}^{n\times n}$ such that $B_1 = B_2U$ and $B_2 = B_1V$. Substituting B_2 in the first equation, we get $B_1 = B_2U$ and $B_2 = B_1V$. Substituting B_2 from the second equation into the first, we have $B_1 = (B_1V)U$, giving $B_1 = B_1(VU)$. Since B_1 is invertible, it follows that $VU = I$, implying $\det(V)\det(U) = 1$. As U and V are integer matrices, we conclude that $\det(U) = \pm 1$ and $\det(V) = \pm 1$.

Conversely, suppose $B_1 = B_2U$ for some unimodular matrix U. Then:

$$\mathcal{L}(B_1) = B_1 \cdot \mathbb{Z}^n = B_2 \cdot (U \cdot \mathbb{Z}^n) = B_2 \cdot \mathbb{Z}^n = \mathcal{L}(B_2),$$

where the equality $U \cdot \mathbb{Z}^n = \mathbb{Z}^n$ follows from the fact that for any $x \in \mathbb{Z}^n$, there exists a $y \in \mathbb{Z}^n$ such that $Ux = y$ and $x = U^{-1}y$, with both U and U^{-1} being integer matrices. □

A fundamental approach to derive one lattice basis from another involves utilizing elementary column transformations. It's noteworthy that such transformations preserve the lattice spanned by the initial basis due to their representation as right multiplication by a unimodular matrix.

1. Interchanging any two columns within matrix B.
2. Inverting the sign of any column, essentially multiplying by -1.
3. Updating a column by adding an integer multiple of another column, specifically $b_i \leftarrow b_i + a \cdot b_j$ where $i \neq j$ and $a \in \mathbb{Z}$.

Remark 16. For any lattice with a higher dimension bigger than 1, there are infinitely many lattice bases.

Proof. Applying different unimodular matrices to a given lattice basis results in different bases. Since there are infinitely many unimodular matrices, there are infinitely many ways to transform a given basis into another, thus creating infinitely many lattice bases. □

Remark 17. There exist subgroups of $\mathbb{R}^n$ that do not form lattices.

To provide an example of subgroups of $\mathbb{R}^n$ that do not form lattices, consider the following: In $\mathbb{R}^n$, a lattice is a discrete subgroup of $\mathbb{R}^n$ spanned by linearly independent vectors. That means you can write any point in the lattice as an integer combination of these basis vectors. Now, let us consider a subgroup of $\mathbb{R}^2$ (for simplicity) that doesn't form a lattice: Consider the subgroup H of $\mathbb{R}^2$ consisting of all points of the form $(x, \pi x)$ where x is a real number. This is indeed a subgroup because it is closed under addition and taking inverses. However, it is not a lattice because there is no set of basis vectors in $\mathbb{R}^2$ that can span this subgroup using only integer combinations. The presence of π, an irrational number, prevents the subgroup from being discrete and spanned by a finite basis of vectors, which is a requirement for a lattice in $\mathbb{R}^n$.

$$H = \{(x, \pi x) \mid x \in \mathbb{R}\}$$

This H is a subgroup of $\mathbb{R}^2$ but does not form a lattice since it fails the discreteness condition.

Any two bases generating the same lattice are related by an unimodular transformation. The Hermite Normal Form provides a canonical[10], unique representation for a lattice basis within its equivalence class, as defined by unimodular transformations (Lemma 2).

Definition 49 (Hermite Normal Form)**.** Let $B \in \mathbb{Z}^{m \times n}$ be a full-rank integer matrix with $m \geq n$. The Hermite Normal Form (HNF) of B is an upper triangular matrix $H \in \mathbb{Z}^{m \times n}$ such that:

1. H is upper triangular, i.e., $H_{ij} = 0$ for all $i > j$,
2. The diagonal entries of H are positive integers ($H_{ii} > 0$),
3. The entries below each pivot in the same column are strictly smaller in magnitude than the pivot: $0 \leq H_{ij} < H_{jj}$ for $i > j$.

Theorem 3 (Hermite Normal Form)**.** *For any full-rank integer matrix $B \in \mathbb{Z}^{m \times n}$, there exists a unique Hermite Normal Form H such that $B = HU$, where $U \in \mathbb{Z}^{n \times n}$ is a unimodular matrix.*

The existence and uniqueness of the Hermite Normal Form can be established through iterative column reduction, ensuring properties such as upper triangularity, positive pivots, and bounded entries below the pivots. This process uses unimodular transformations to preserve the lattice structure. For a detailed proof and related algorithms, see Chapter 2, "Algorithms for Linear Algebra and Lattices", in Cohen's book *A Course in Computational Algebraic Number Theory* [43].

[10]Borrowed from abstract algebra, canonical refers to a standardized form, such as the smallest nonnegative integer in a congruence class.

Remark 18. Normal forms are important because they provide a unique, standardized representation of a lattice basis, enabling us to determine whether different bases generate the same lattice by simply comparing their normal forms.

We have examined lattice representations, rank, and equivalences under unimodular transformations. Our next step is to consider a geometric viewpoint, highlighting how a lattice fills space with discrete points and how one can identify a single shape that encodes the entire repeating pattern.

4.3 Geometric Structures of Lattices

To better understand lattice structures, we can visualize a 2D lattice as a discrete grid of points in $\mathbb{R}^2$. These points are generated by integer linear combinations of two linearly independent basis vectors. The following figure illustrates this concept using the **integer lattice**, where each point corresponds to a pair of integer coordinates, forming a regular, grid-like pattern.

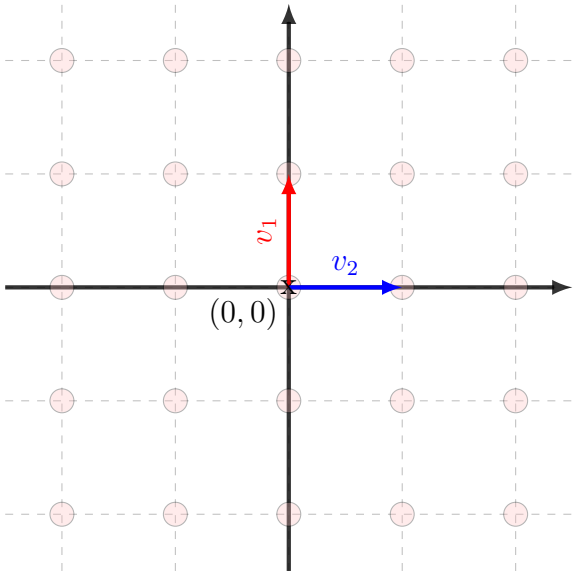

Figure 7: Integer lattice generated by basis vectors $\mathbf{v}_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and $\mathbf{v}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

Figure 7 shows a 2-dimensional lattice in $\mathbb{R}^2$, represented as a regular grid of discrete points. Each point is obtained as an integer linear combination of the two basis vectors originating from the origin $(0, 0)$. The dashed grid lines emphasize the regular spacing of the points, reflecting the discrete nature of the lattice.

The choice of basis vectors determines the lattice's orientation and scale in $\mathbb{R}^2$. While the specific basis can vary, all such bases span the same lattice. This property is illustrated in Figure 8, which demonstrates the same lattice generated using a non-standard basis. Understanding the flexibility in choosing bases is crucial for

applications such as cryptography, where certain *better bases* with specific properties are more advantageous. This concept will be discussed further in subsequent sections.

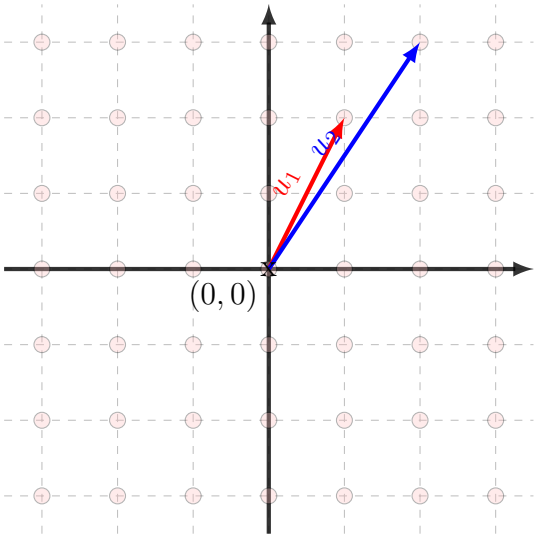

Figure 8: Integer lattice generated by a non-standard basis $\mathbf{u_1} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ and $\mathbf{u_2} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$

Beyond visualizing the lattice itself, it is often useful to study the geometric structure of its fundamental region. The fundamental region provides a way to partition the space $\mathbb{R}^n$ into disjoint regions, each of which maps uniquely to a single lattice point. Consider a standard chessboard with alternating black and white squares. Each square can be thought of as a fundamental unit of the entire pattern, as the board's checkerboard structure is created by repeating this simple arrangement. The concept of a fundamental region formalizes such repetitions, allowing us to describe the entire lattice using just this basic repeating unit.

Definition 50. (Fundamental Region) A subset $F \subseteq \mathbb{R}^n$ is called a fundamental region of a lattice $\mathcal{L}$ if the collection of its translates $x + F := \{x + y : y \in F\}$, for all $x \in \mathcal{L}$, partitions $\mathbb{R}^n$.

In simple terms, a set $\mathcal{F}$ is considered a fundamental region of a lattice $\mathcal{L}$ if you can move this set around by adding different points from the lattice $\mathcal{L}$ to it, and when you do this for all possible points in $\mathcal{L}$, you end up dividing up the entire space $\mathbb{R}^n$ into non-overlapping pieces. The interval $[0, 1)$ is a fundamental region for $\mathbb{Z}$. Why? Any real number x can be written as:

$$x = n + r, \quad \text{where } n \in \mathbb{Z},\ r \in [0, 1).$$

If you have a basis for the lattice (e.g., $\mathbf{v}_1, \mathbf{v}_2, \ldots$ in $\mathbb{R}^n$), you can define a fundamental region using combinations of the basis vectors:

$$\mathbf{r} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots, \quad 0 \leq a_i < 1.$$

This region is called a *parallelepiped*.

Definition 51. (Fundamental Parallelpiped) For any given lattice basis B in $\mathbb{R}^n$, the *Fundamental Parallelpiped* is defined as the set $\mathcal{P}(B)$, which is expressed as:

$$\mathcal{P}(B) = \{Bx \mid x \in \mathbb{R}^n, \forall i : 0 \leq x_i < 1\}.$$

Definition 52. (Volume/Determinant of a Lattice) Let $\mathcal{L}$ be a lattice in $\mathbb{R}^n$ generated by a basis $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ represented as column vectors, the basis vectors are arranged as the columns of a matrix B:

$$B = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{bmatrix}.$$

The volume, or determinant, of $\mathcal{L}$, denoted by $\det(\mathcal{L})$, is defined as:

$$\det(\mathcal{L}) = |\det(B)|.$$

Remark 19. This definition of volume is valid only for full-rank lattices. If the lattice is not full-rank (i.e., some basis vectors are linearly dependent), the determinant $\det(B)$ is zero. In this case, the parallelepiped spanned by the basis vectors has no *volume* in $\mathbb{R}^n$ because it collapses into a lower-dimensional subspace. For example, in $\mathbb{R}^3$, if the lattice vectors lie in the same plane, the determinant is zero because the parallelepiped degenerates to a two-dimensional shape.

Remark 20. The value $|\det(B)|$ remains unchanged regardless of the choice of lattice basis B. According to Lemma 2, any alternate basis B_0 can be expressed as $B_0 = BU$, where U is a unimodular matrix. Since $\det(U) = \pm 1$, it follows that $|\det(B_0)| = |\det(B)|$. The concept of lattice volume $|\det(B)|$ is well-defined because it is invariant under the choice of lattice basis.

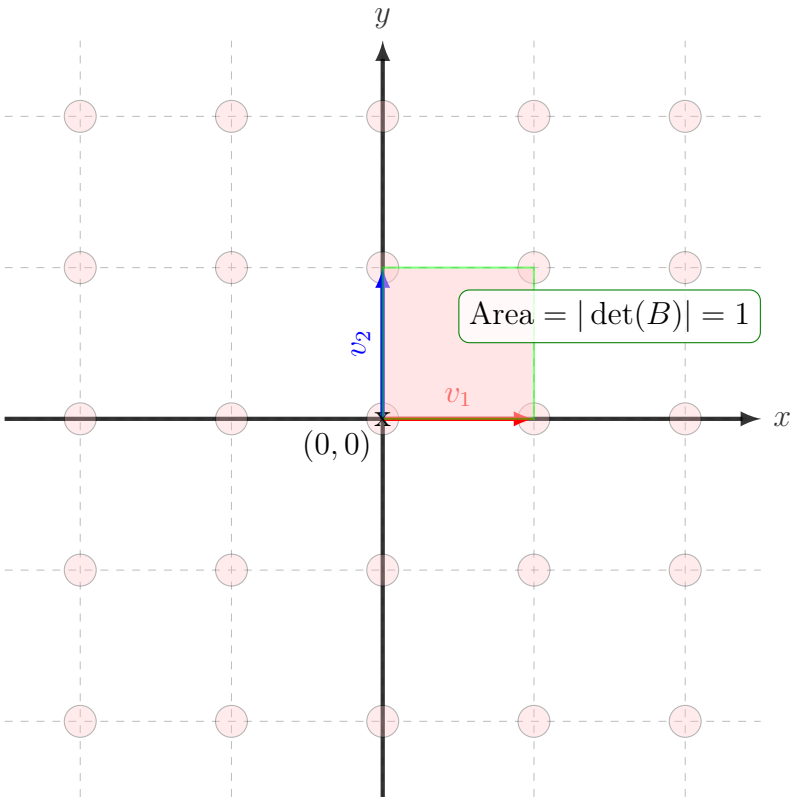

Figure 9: Fundamental Parallelogram of Integer Lattice with Determinant Highlighted

One might initially assume that the parallelogram formed by $\mathbf{u}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ and $\mathbf{u}_2 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$ has a larger area than the unit square formed by $\mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $\mathbf{v}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. After all, $\mathbf{u}_1$ and $\mathbf{u}_2$ have greater magnitudes than $\mathbf{v}_1$ and $\mathbf{v}_2$. However, the area depends on more than just vector lengths: the alignment of the vectors also plays a critical role. Specifically, the determinant captures both these factors — the magnitudes of the vectors and the angle between them.

For $\mathbf{u}_1$ and $\mathbf{u}_2$, the determinant of the matrix with these vectors as columns is:

$$\det \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} = -1.$$

The area of the parallelogram is given by the absolute value of the determinant, $|\det| = 1$. Remarkably, this matches the area of the unit square, where $|\det(B)| =$

1. In this example, the determinant balances the longer magnitudes of $\mathbf{u}_1$ and $\mathbf{u}_2$ against their closer alignment, preserving the overall lattice volume.

Remark 21. The determinant's invariance ensures that the lattice volume remains constant, regardless of the choice of basis B, as long as the basis spans the same lattice.

The relationship between the determinant, the alignment of basis vectors, and their magnitudes can be formalized through the following inequality:

Theorem 4 (Hadamard's Inequality)**.** *Let B be an $n \times n$ matrix whose columns $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ form a basis for a lattice. Then, Hadamard's inequality is expressed as:*

$$|\det(B)| \leq \prod_{i=1}^{n} \|\mathbf{b}_i\|,$$

where equality is achieved if and only if the basis vectors $\mathbf{b}_i$ are mutually orthogonal.

This inequality provides an upper bound for the volume of the fundamental parallelepiped of the lattice in terms of the magnitudes of the basis vectors. For example, in the 2-dimensional case, the area of the parallelogram spanned by the lattice basis $\mathbf{u}_1$ and $\mathbf{u}_2$ is given as follows:

$$|\det(B)| = \left| \det \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \right| = |(1 \cdot 3) - (2 \cdot 2)| = |3 - 4| = 1.$$

The product of the magnitudes of the basis vectors is calculated as:

$$\prod_{i=1}^{n} \|\mathbf{b}_i\| = \|\mathbf{u}_1\| \cdot \|\mathbf{u}_2\| = \sqrt{1^2 + 2^2} \cdot \sqrt{2^2 + 3^2} = \sqrt{5} \cdot \sqrt{13} = \sqrt{65}.$$

It is observed that Hadamard's inequality is satisfied:

$$|\det(B)| = 1 \leq \sqrt{65}.$$

This demonstrates that the alignment of $\mathbf{u}_1$ and $\mathbf{u}_2$ reduces the effective volume of the lattice compared to the product of their magnitudes.

Definition 53 (Hadamard Ratio)**.** For a lattice basis $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$, the *Hadamard ratio* is defined as:

$$\left(\frac{|\det(B)|}{\prod_{i=1}^{n} \|\mathbf{b}_i\|} \right)^{1/n}.$$

The Hadamard ratio is used to quantify the impact of the alignment of basis vectors on the lattice volume. The parameter n, which represents the dimension of the lattice, has been included to normalize the ratio for higher-dimensional lattices. By

raising the ratio to the power of $1/n$, the influence of the dimension on the comparison is mitigated, making the metric applicable to lattices of varying dimensions.

Values of the Hadamard ratio closer to 1 indicate near-orthogonality of the basis vectors, while smaller values reflect greater alignment (and therefore less efficient use of the basis in spanning the lattice volume).

In the given example, the Hadamard ratio is calculated as:

$$\left(\frac{|\det(B)|}{\prod_{i=1}^{n}\|\mathbf{b}_i\|}\right)^{1/n} = \left(\frac{1}{\sqrt{65}}\right)^{1/2} \approx 0.352.$$

To extend the example, the basis vectors

$$b_1 = (10, 0), \quad b_2 = (1, 10)$$

form the matrix

$$B = \begin{pmatrix} 10 & 1 \\ 0 & 10 \end{pmatrix}$$

with $\det(B) = 100$. Since $\|b_1\| = 10$ and $\|b_2\| = \sqrt{101}$, the Hadamard ratio

$$H = \left(\frac{100}{10\sqrt{101}}\right)^{\frac{1}{2}} \approx 0.9975,$$

which is very close to 1. This indicates that the basis vectors are almost orthogonal, ensuring efficient use of the lattice volume relative to the product of their lengths.

Having explored the properties of lattices through the lens of basis vector alignment, we now turn our attention to a deeper structural concept: lattice duality.

4.4 Duality

Duality is considered fundamental in lattice theory and is utilized in applications such as Fourier analysis and solving lattice-based problems. The formal definition and properties of the dual lattice will now be introduced to extend this understanding.

Definition 54. (Dual Lattice) Let $\mathcal{L} \subset \mathbb{R}^n$ be a full-rank lattice. The *dual lattice* of $\mathcal{L}$, denoted by $\mathcal{L}^*$, is defined as

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

where

$$\langle \mathbf{x}, \mathbf{y} \rangle = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

denotes the *standard Euclidean inner product* on $\mathbb{R}^n$.

Example 13. The lattice of integer points satisfies

$$(\mathbb{Z}^n)^* \ = \ \mathbb{Z}^n.$$

Proof. Take any $\mathbf{y} \in \mathbb{Z}^n$, where $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ with $y_i \in \mathbb{Z}$. For any $\mathbf{x} \in \mathbb{Z}^n$, where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ with $x_i \in \mathbb{Z}$, the inner product is

$$\langle \mathbf{x}, \mathbf{y} \rangle \;=\; \sum_{i=1}^{n} x_i \, y_i.$$

Since each $x_i, y_i \in \mathbb{Z}$, their product $x_i y_i$ lies in $\mathbb{Z}$, and hence the sum $\langle \mathbf{x}, \mathbf{y} \rangle \in \mathbb{Z}$. This shows $\mathbf{y} \in (\mathbb{Z}^n)^*$, so $\mathbb{Z}^n \subseteq (\mathbb{Z}^n)^*$.

Conversely, if $\mathbf{y} \in (\mathbb{Z}^n)^*$, then in particular

$$\langle \mathbf{e}_i, \mathbf{y} \rangle \;=\; y_i \;\in\; \mathbb{Z} \quad \text{for each standard basis vector } \mathbf{e}_i.$$

Thus $\mathbf{y} \in \mathbb{Z}^n$, giving $(\mathbb{Z}^n)^* \subseteq \mathbb{Z}^n$. Combining these inclusions yields $(\mathbb{Z}^n)^* = \mathbb{Z}^n$. □

Definition 55. (Dual Basis) Let $B = (b_1, \ldots, b_n) \in \mathbb{R}^{m \times n}$ be a basis matrix whose columns form a basis. The dual basis $D = (d_1, \ldots, d_n) \in \mathbb{R}^{m \times n}$ is the unique basis such that $\text{span}(D) = \text{span}(B)$ and $B^\top D = I$, where I is the identity matrix.

Lemma 3. *For any lattice $\mathcal{L}$, the dual of the dual lattice is the original lattice:*

$$(\mathcal{L}^*)^* = \mathcal{L}.$$

Proof. Let B be a basis of $\mathcal{L}$. Then the matrix $B(B^\top B)^{-1}$ forms a basis for $\mathcal{L}^*$.

To compute a basis for $(\mathcal{L}^*)^*$, observe that:

$$\big(B(B^\top B)^{-1}\big) \cdot \Big(\big(B(B^\top B)^{-1}\big)^\top \cdot B(B^\top B)^{-1}\Big)^{-1} = B.$$

This shows that B is also a basis for $(\mathcal{L}^*)^*$, and thus $(\mathcal{L}^*)^* = \mathcal{L}$. [44] □

Remark 22. Suppose $\mathcal{L} \subset \mathbb{R}^n$ is a full-rank lattice. Then

$$\det\big(\mathcal{L}^*\big) \;=\; \frac{1}{\det(\mathcal{L})}.$$

Proof.

$$\det(\mathcal{L}^*) = \big|\det\big((B^T)^{-1}\big)\big| = \left|\frac{1}{\det(B^T)}\right| = \frac{1}{\det(B)} = \frac{1}{\det(\mathcal{L})}.$$

□

The dual lattice $\mathcal{L}^*$ satisfies fundamental properties, including $(\mathcal{L}^*)^* = \mathcal{L}$ and preservation of dimension, though its geometry differs from $\mathcal{L}$. This structure plays a crucial role in lattice-based cryptography, particularly where duality aids in hardness assumptions and security proofs. However, as lattice dimensions grow, optimizing dual lattice computations remains a critical research focus, balancing precision with computational feasibility.

4.5 Bounding Short Vectors in High-Dimensions

The discrete nature of a lattice $\mathcal{L}$ guarantees the existence of at least one nonzero vector $v \in \mathcal{L}$ with minimal magnitude under a chosen norm. For instance, under the Euclidean norm, the magnitude of any vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ is defined as

$$\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

Definition 56 (Minimum Distance)**.** The *minimum distance* of a lattice $\mathcal{L}$ is the smallest nonzero distance between any two lattice points:

$$\lambda(\mathcal{L}) = \min\{\|x - y\| : x, y \in \mathcal{L}, x \neq y\}.$$

Definition 57 (Successive Minima)**.** The ith *successive minimum* of a lattice $\mathcal{L}$ of rank n, denoted $\lambda_i(\mathcal{L})$, is the smallest real number r such that there exist i linearly independent vectors in $\mathcal{L}$, each having a norm not exceeding r:

$$\lambda_i(\mathcal{L}) = \min\left\{r : \exists\, \{\mathbf{v}_1, \ldots, \mathbf{v}_i\} \subseteq \mathcal{L} \text{ linearly independent, } \|\mathbf{v}_j\| \leq r \text{ for all } j\right\}.$$

Remark 23. Every lattice contains at least two shortest vectors since for any nonzero $v \in \mathcal{L}$, we have $\|v\| = \|-v\|$.

Since we have established the existence of short vectors in every lattice, a natural question arises: What is the maximum possible length of the shortest vector relative to the lattice's volume? Foundational results such as Minkowski's theorem[11] and the notion of Hermite's constant provide sharp upper bounds on these minima by linking them to the geometry and volume of the lattice.

Hermite's constant, γ_d, in dimension d measures the largest possible value of the normalized squared length of the shortest vector in any d-dimensional lattice.

Definition 58 (Hermite Constant)**.** The *Hermite constant* γ_d in dimension d is defined as

$$\gamma_d = \sup_L \frac{\lambda_1(L)^2}{\mathrm{vol}(L)^{2/d}},$$

where the supremum is taken over all d-dimensional lattices L.

For every dimension d, there exists at least one *critical lattice* where the supremum γ_d is achieved. However, the exact values of γ_d are known only in a few dimensions. Table 5 summarizes the known values and approximations, including the latest result for $d = 24$ [45].

For higher dimensions, estimating γ_d becomes increasingly challenging. As an early result, Hermite [46] established the upper bound

$$\gamma_d \leq \left(\frac{4}{3}\right)^{\frac{d-1}{2}},$$

[11] See Section 4.6

d	2	3	4	5	6	7	8	24
γ_d	$\frac{2}{\sqrt{3}}$	$2^{1/3}$	$\sqrt{2}$	$8^{1/5}$	$(64/3)^{1/6}$	$64^{1/7}$	2	4
Approximation	1.1547	1.2599	1.4142	1.5157	1.6654	1.8114	2	4

Table 5: Known values and approximations of Hermite's constant γ_d. [44]

which shows that the normalized squared length of the shortest vector grows at most exponentially with the dimension.

To illustrate the application of Hermite's inequality, consider the lattice with basis vectors

$$b_1 = (1, 1) \quad \text{and} \quad b_2 = (2, 0).$$

The determinant of this lattice is

$$\det(\mathcal{L}) = \begin{vmatrix} 1 & 2 \\ 1 & 0 \end{vmatrix} = 2.$$

Hermite's inequality provides the bound

$$\lambda_1(\mathcal{L}) \leq \gamma_2^{1/2} \det(\mathcal{L})^{1/2},$$

where $\gamma_2 = \frac{4}{3}$. Hence,

$$\lambda_1(\mathcal{L}) \leq \sqrt{\frac{4}{3}} \cdot \sqrt{2} \approx 1.633.$$

Figure 10 illustrates this two-dimensional lattice, highlighting the lattice structure and the region within which the shortest vector (of length $\lambda_1(\mathcal{L})$) lies. In two dimensions, the geometry is sufficiently simple to visually identify these short vectors.

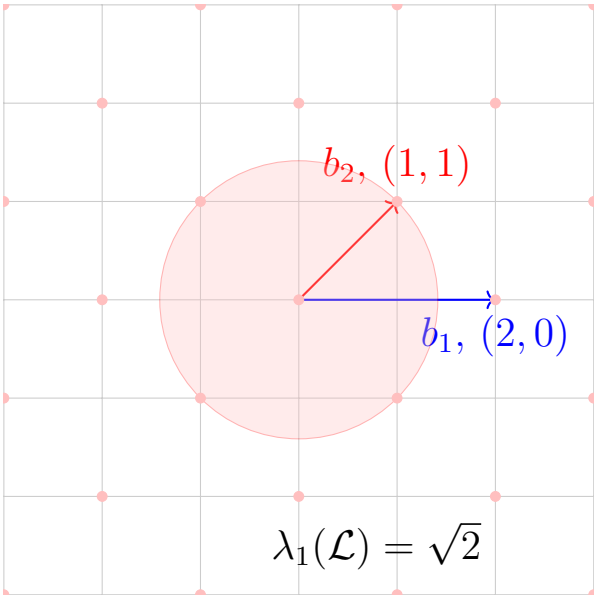

Figure 10: Two-dimensional lattice with the shortest vector norm $\lambda_1(\mathcal{L}) = \sqrt{2}$.

In this lattice, the shortest nonzero vector is found to be

$$\mathbf{v} = (-1, 1),$$

with a norm of

$$\lambda_1(\mathcal{L}) = \sqrt{2} \approx 1.414.$$

This result agrees with the theoretical expectation, as it falls within the Hermite bound of approximately 1.633. While such computations are straightforward in two dimensions, the problem of finding short vectors becomes exponentially more complex in higher dimensions, often requiring sophisticated techniques.

It is noteworthy that no efficient algorithm is currently known for computing the successive minima of a lattice in general. Nevertheless, foundational results such as Minkowski's theorem [47] offer critical insights into these minima, highlighting the deep interplay between lattice geometry and number theory.

4.6 Minkowski's Theorems

The geometry of numbers provides a rich framework for analyzing the connection between lattice structures and convex sets. Foundational theorems in this field, such as those by Blichfeldt and Minkowski, offer powerful tools to study lattice points and their associated properties.

In this subsection, we establish a significant upper bound on the product of the successive minima of lattices, leveraging Minkowski's theorems from the geometry of numbers.

Blichfeldt's theorem asserts that any measurable set in $\mathbb{R}^n$ with volume greater than the determinant of a lattice $\mathcal{L}$ must contain at least one lattice point of $\mathcal{L}$. This result

is particularly useful in problems involving lattice point enumeration and covering properties of convex sets.

Theorem 5 (Blichfeldt's Theorem)**.** *Let $\mathcal{L} \subset \mathbb{R}^n$ be a lattice with determinant* $\det(\mathcal{L})$*, and let $S \subset \operatorname{span}(\mathcal{L})$ be a measurable set with* $\operatorname{vol}(S) > \det(\mathcal{L})$*. Then there exist two distinct points $z_1, z_2 \in S$ such that $z_1 - z_2 \in \mathcal{L}$.*

For a detailed proof of this result, the reader is referred to Micciancio's work on the geometry of numbers and lattice theory in *Complexity of Lattice Problems: A Cryptographic Perspective* [48].

Theorem 6. *(Minkowski's Convex Body Theorem) Consider a lattice $\mathcal{L}$ within $\mathbb{R}^n$. Any convex, centrally symmetric set $S \subset \mathbb{R}^n$ of volume* $\operatorname{vol}(S) > 2^n \cdot \det(\mathcal{L})$ *contains a nonzero lattice point, i.e., $S \cap \mathcal{L} \neq \{0\}$.*

Proof. Let $S' = S/2$, so that $\operatorname{vol}(S') > \det(\mathcal{L})$. Since volumes in $\mathbb{R}^n$ scale by the n-th power of the scaling factor, dividing S in half along each dimension ensures S' still has volume greater than the determinant of the lattice. Using a volumetric pigeonhole principle, we claim that there exist distinct points $x, y \in S'$ such that $x - y \in \mathcal{L}$. To demonstrate this, consider a fundamental region $\mathcal{F}$ of the lattice $\mathcal{L}$. Partition S' into sets of the form: $S'_v = S' \cap (v + \mathcal{F})$, for each lattice point $v \in \mathcal{L}$. Since $\operatorname{vol}(S') > \operatorname{vol}(\mathcal{F})$, the total volume of all the translated sets $S'_v - v \subseteq \mathcal{F}$ exceeds $\operatorname{vol}(\mathcal{F})$. By the pigeonhole principle, at least two of these sets must overlap, implying there exist $z \in (S'_u - u) \cap (S'_v - v)$ for distinct lattice points $u, v \in \mathcal{L}$.

It follows that the points $x = z + u$ and $y = z + v$ lie in S', and their difference $x - y = u - v \in \mathcal{L}$ is a lattice point. Note that $2x, -2y \in S$ due to the scaling symmetry of S'. By convexity of S, their midpoint, also belongs to S, completing the proof. □

Theorem 7 (Minkowski's First Theorem)**.** *Let $\mathcal{L} \subset \mathbb{R}^n$ be a lattice with determinant* $\det(\mathcal{L})$*, and let $\lambda_1(\mathcal{L})$ denote the first minimum of $\mathcal{L}$, i.e., the length of the shortest nonzero lattice vector in $\mathcal{L}$. Then,*

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

Proof. Without loss of generality, assume $\det(\mathcal{L}) = 1$. This assumption is valid because scaling the lattice by a factor of $\det(\mathcal{L})^{-1/n}$ normalizes the determinant to 1, while scaling $\lambda_1(\mathcal{L})$ by the same factor.

Now, consider the set

$$S = \sqrt{n} \cdot \overline{B},$$

where $\overline{B}$ is the closed Euclidean ball of radius 1 centered at the origin. The cube $[-1, 1]^n$, which has side length 2 and volume 2^n, is strictly contained within S for $n > 1$. For $n = 1$, S is simply the interval $[-1, 1]$, and the result follows trivially since $\lambda_1(\mathcal{L}) \leq 1 = \sqrt{1} \cdot \det(\mathcal{L})^{1/n}$.

The volume of S satisfies

$$\operatorname{vol}(S) = \operatorname{vol}(\overline{B}) \cdot (\sqrt{n})^n > 2^n,$$

where $\text{vol}(\overline{B})$ is the volume of the unit n-dimensional Euclidean ball. This inequality holds because the constant factor relating $\text{vol}(\overline{B})$ and the volume of the unit cube is less than 1.

By the Convex Body Theorem, any convex, symmetric set in $\mathbb{R}^n$ with volume exceeding $\det(\mathcal{L}) = 1$ contains at least one nonzero lattice point. Thus, S contains a nonzero lattice point, which implies that there exists a lattice vector of length at most $\sqrt{n}$. Therefore,

$$\lambda_1(\mathcal{L}) \leq \sqrt{n}.$$

Scaling back to the original lattice, the bound generalizes to

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

□

Remark 24. Using a more precise formula for the volume of an n-dimensional Euclidean ball, one can derive a slightly tighter bound for $\lambda_1(\mathcal{L})$:

$$\lambda_1(\mathcal{L}) \leq \sqrt{\frac{n}{2\pi e}} \cdot \det(\mathcal{L})^{1/n}.$$

This improved bound for $\lambda_1(\mathcal{L})$ is detailed in [44, 49], with modern refinements in [50].

Additionally, if the lattice $\mathcal{L}$ and its minimum distance are scaled by a factor c, the determinant of the lattice scales by c^n. Consequently, the bound also scales proportionally, ensuring consistency.

Theorem 8 (Minkowski's Second Theorem)**.** *Let $\mathcal{L} \subset \mathbb{R}^n$ be a lattice with determinant $\det(\mathcal{L})$, and let $\lambda_1, \lambda_2, \ldots, \lambda_n$ denote its successive minima. Then, the geometric mean of the successive minima satisfies the inequality:*

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

Minkowski's First Theorem establishes a bound for the first successive minimum, λ_1, in terms of the determinant of the lattice, $\det(L)$, and the volume of the convex body, $Vol(S)$. Specifically, it guarantees that $\lambda_1 \leq vol(S)/\det(L)$, ensuring the existence of at least one *short* lattice vector within the scaled convex body S. Minkowski's Second Theorem extends this result to all successive minima, λ_i. It provides a bound on the product of these minima, stating that $\prod_{i=1}^{n} \lambda_i \leq vol(S)/\det(L)$. This generalization relates the geometry of S to the lattice structure across all independent lattice directions.

4.7 Gram-Schmidt Orthogonalization

The Gram-Schmidt orthogonalization process is a fundamental technique extensively utilized in lattice-related problems and other areas of linear algebra. It transforms any collection of n linearly independent vectors into a corresponding set of n orthogonal vectors. Furthermore, normalization of these orthogonal vectors produces an orthonormal basis, simplifying many computational and theoretical tasks. For a detailed proof of the Gram-Schmidt Orthogonalization Theorem, the reader is referred to Strang [51].

Theorem 9 (Gram-Schmidt Orthogonalization)**.** *Let $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ be a set of n linearly independent vectors in an inner product space. The Gram-Schmidt orthogonalization process constructs a sequence of orthogonal vectors $\widetilde{\mathbf{b}}_1, \widetilde{\mathbf{b}}_2, \ldots, \widetilde{\mathbf{b}}_n$ recursively as follows:*

1. *Initialize the first vector:*
$$\widetilde{\mathbf{b}}_1 = \mathbf{b}_1.$$

2. *For $i = 2, \ldots, n$, compute the i-th orthogonal vector as:*
$$\widetilde{\mathbf{b}}_i = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \widetilde{\mathbf{b}}_j,$$
where the coefficients $\mu_{i,j}$ are given by:
$$\mu_{i,j} = \frac{\langle \mathbf{b}_i, \widetilde{\mathbf{b}}_j \rangle}{\langle \widetilde{\mathbf{b}}_j, \widetilde{\mathbf{b}}_j \rangle}.$$

After completing the orthogonalization process, the resulting vectors $\widetilde{\mathbf{b}}_1, \widetilde{\mathbf{b}}_2, \ldots, \widetilde{\mathbf{b}}_n$ can be normalized to form an orthonormal basis $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$, where:

$$\mathbf{u}_i = \frac{\widetilde{\mathbf{b}}_i}{\|\widetilde{\mathbf{b}}_i\|}, \quad \text{for } i = 1, \ldots, n.$$

By transforming a given lattice basis into an orthogonal or orthonormal one, the relationships between basis vectors become clearer, simplifying geometric and algebraic analyses. Gram-Schmidt plays a foundational role in lattice algorithms, where orthogonalization is used to iteratively refine a lattice basis into a more *reduced* form with shorter and nearly orthogonal vectors.

Example 14. Consider the vectors

$$\mathbf{b}_1 = \begin{bmatrix} 10 \\ \\ 5 \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} 3 \\ \\ 7 \end{bmatrix}.$$

We will apply the Gram-Schmidt process to compute the orthogonal and orthonormal vectors.

Algorithm 2 Gram-Schmidt Process

Require: Linearly independent vectors $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$
Ensure: Orthonormal vectors $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n\}$

$\widetilde{\mathbf{b}}_1 \leftarrow \mathbf{b}_1$
$\mathbf{u}_1 \leftarrow \frac{\widetilde{\mathbf{b}}_1}{\|\widetilde{\mathbf{b}}_1\|}$
for $i = 2$ to n **do**
    $\widetilde{\mathbf{b}}_i \leftarrow \mathbf{b}_i$
    for $j = 1$ to $i - 1$ **do**
        $\widetilde{\mathbf{b}}_i \leftarrow \widetilde{\mathbf{b}}_i - \frac{\langle \mathbf{b}_i, \mathbf{u}_j \rangle}{\langle \mathbf{u}_j, \mathbf{u}_j \rangle} \widetilde{\mathbf{b}}_j$
    end for
    $\mathbf{u}_i \leftarrow \frac{\widetilde{\mathbf{b}}_i}{\|\widetilde{\mathbf{b}}_i\|}$
end for

Normalize $\mathbf{b}_1$ to find $\mathbf{u}_1$:

We take $\mathbf{b}_1$ as the first orthogonal vector, $\mathbf{e}_1 = \mathbf{b}_1$:

$$\mathbf{e}_1 = \begin{bmatrix} 10 \\ \\ 5 \end{bmatrix}.$$

Next, we compute its norm:

$$\|\mathbf{e}_1\| = \sqrt{10^2 + 5^2} = \sqrt{100 + 25} = \sqrt{125} = 5\sqrt{5}.$$

The normalized vector is:

$$\mathbf{u}_1 = \frac{\mathbf{e}_1}{\|\mathbf{e}_1\|} = \frac{1}{5\sqrt{5}} \begin{bmatrix} 10 \\ \\ 5 \end{bmatrix} = \begin{bmatrix} \frac{2}{\sqrt{5}} \\ \\ \frac{1}{\sqrt{5}} \end{bmatrix}.$$

Orthogonalize $\mathbf{b}_2$ with respect to $\mathbf{u}_1$:

The second orthogonal vector $\mathbf{e}_2$ is obtained by subtracting from $\mathbf{b}_2$ its projection onto $\mathbf{u}_1$:

$$\mathbf{e}_2 = \mathbf{b}_2 - \langle \mathbf{b}_2, \mathbf{u}_1 \rangle \mathbf{u}_1.$$

We first compute the projection coefficient:

$$\langle \mathbf{b}_2, \mathbf{u}_1 \rangle = \begin{bmatrix} 3 \\ \\ 7 \end{bmatrix} \cdot \begin{bmatrix} \frac{2}{\sqrt{5}} \\ \\ \frac{1}{\sqrt{5}} \end{bmatrix} = \frac{6}{\sqrt{5}} + \frac{7}{\sqrt{5}} = \frac{13}{\sqrt{5}}.$$

The projection is:

$$\langle \mathbf{b}_2, \mathbf{u}_1 \rangle \mathbf{u}_1 = \frac{13}{\sqrt{5}} \begin{bmatrix} \frac{2}{\sqrt{5}} \\ \\ \frac{1}{\sqrt{5}} \end{bmatrix} = \frac{13}{5} \begin{bmatrix} 2 \\ \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{26}{5} \\ \\ \frac{13}{5} \end{bmatrix}.$$

Subtracting the projection:

$$\mathbf{e}_2 = \begin{bmatrix} 3 \\ 7 \end{bmatrix} - \begin{bmatrix} \frac{26}{5} \\ \frac{13}{5} \end{bmatrix} = \begin{bmatrix} \frac{15}{5} - \frac{26}{5} \\ \frac{35}{5} - \frac{13}{5} \end{bmatrix} = \begin{bmatrix} -\frac{11}{5} \\ \frac{22}{5} \end{bmatrix}.$$

Normalize $\mathbf{e}_2$ to find $\mathbf{u}_2$:

We compute the norm of $\mathbf{e}_2$:

$$\|\mathbf{e}_2\| = \sqrt{\left(-\frac{11}{5}\right)^2 + \left(\frac{22}{5}\right)^2} = \sqrt{\frac{121}{25} + \frac{484}{25}} = \sqrt{\frac{605}{25}} = \frac{\sqrt{605}}{5}.$$

The normalized vector is:

$$\mathbf{u}_2 = \frac{\mathbf{e}_2}{\|\mathbf{e}_2\|} = \frac{1}{\frac{\sqrt{605}}{5}} \begin{bmatrix} -\frac{11}{5} \\ \frac{22}{5} \end{bmatrix} = \frac{5}{\sqrt{605}} \begin{bmatrix} -\frac{11}{5} \\ \frac{22}{5} \end{bmatrix} = \begin{bmatrix} -\frac{11}{\sqrt{605}} \\ \frac{22}{\sqrt{605}} \end{bmatrix}.$$

Final Orthonormal Basis: The orthonormal basis is:

$$\mathbf{u}_1 = \begin{bmatrix} \frac{2}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{bmatrix}, \quad \mathbf{u}_2 = \begin{bmatrix} -\frac{11}{\sqrt{605}} \\ \frac{22}{\sqrt{605}} \end{bmatrix}.$$

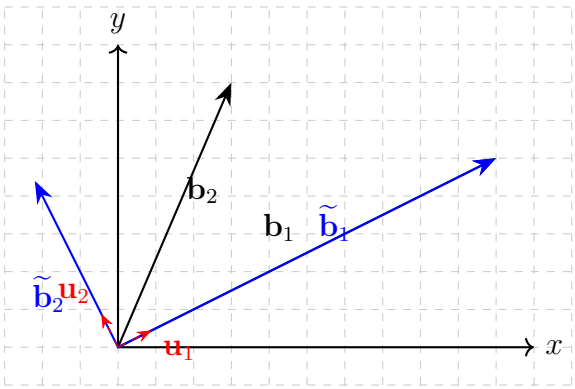

Figure 11: Gram-Schmidt Process for $\mathbf{b}_1$ and $\mathbf{b}_2$

The figure illustrates the Gram-Schmidt process for the vectors $\mathbf{b}_1 = \begin{bmatrix} 10 \\ 5 \end{bmatrix}$ and $\mathbf{b}_2 = \begin{bmatrix} 3 \\ 7 \end{bmatrix}$. The blue vectors represent the orthogonalized vectors $\widetilde{\mathbf{b}}_1$ and $\widetilde{\mathbf{b}}_2$, while the red dashed vectors show the normalized orthonormal basis $\mathbf{u}_1$ and $\mathbf{u}_2$.

This transformation is essential for improving the efficiency and effectiveness of algorithms. This approach not only facilitates easier computations but also improves the precision of numerical operations by minimizing the issues caused by nearly linearly dependent vectors.

5 Essential Hard Problems

Minkowski's First Theorem is a fundamental result in lattice theory, guaranteeing the existence of a short, nonzero vector $\mathbf{v} \in \mathcal{L}$ within any rank n lattice $\mathcal{L}$, bounded by

$$\lambda_1(\mathcal{L}) = \|\mathbf{v}\| \leq \sqrt{n} \cdot |\det(\mathcal{L})|^{1/n}.$$

This theoretical guarantee contrasts with the computational difficulty of efficiently finding such vectors, a discrepancy central to the challenges in lattice-based problems. These problems encompassing search, optimization, and decision tasks are crucial to lattice theory and increasingly relevant in cryptographic applications. This section introduces key lattice problems vital for understanding computational complexity and the security of lattice-based cryptography. Well-studied examples include:

- **Shortest Vector Problem (SVP):** Find a nonzero lattice vector $\mathbf{v} \in \mathcal{L}$ minimizing $\|\mathbf{v}\|$.
- **Closest Vector Problem (CVP):** Given $\mathbf{t} \in \mathbb{R}^n$, find a lattice vector $\mathbf{v} \in \mathcal{L}$ minimizing $\|\mathbf{t} - \mathbf{v}\|$.
- **Short Integer Solution (SIS) Problem:** Find a short, nonzero integer vector $\mathbf{z}$ such that $\mathbf{Az} = \mathbf{0} \pmod{q}$ for a given matrix $\mathbf{A}$.
- **Learning With Errors (LWE):** Recover the secret vector by solving the system of noisy linear equations $\mathbf{s} \in \mathbb{Z}_q^n$ such that $\mathbf{As} + \mathbf{e} \equiv \mathbf{b} \pmod{q}$, given only the public matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and the noise vector $\mathbf{e} \in \mathbb{Z}_q^m$.

These problems and their variations[12] with slight changes are computationally hard. Subsequent sections will detail these problems, exploring their definitions, computational challenges, and algorithmic approaches, highlighting their central role in cryptographic applications.

5.1 Shortest Vector Problem (SVP)

The Shortest Vector Problem is a central, simplest and probably most basic computational challenge in lattice theory and a cornerstone of lattice-based cryptography.

Definition 59 (Shortest Vector Problem)**.** Given a basis $\mathcal{B}$, find a non-zero vector $\mathbf{v} \in \mathcal{L}$ that satisfies

$$\|\mathbf{v}\| = \lambda_1(\mathcal{L}),$$

Given a lattice, SVP requires finding its shortest nonzero vector. Despite its straightforward definition, and straightforward solutions in low dimensions, SVP is conjectured to be computationally hard, even against quantum adversaries [52].

[12]Several other variants of these problems exist, often imposing additional constraints or optimizing for efficiency, yet they remain fundamentally equivalent in complexity

Figure 12 provides an intuitive visualization of the concept. The basis vectors $\mathbf{b}_1$ and $\mathbf{b}_2$, represented as column vectors

$$\mathbf{b}_1 = \begin{bmatrix} 2 \\ \\ 3 \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} -1 \\ \\ 4 \end{bmatrix},$$

span the lattice, where each lattice point corresponds to an integer linear combination of these vectors.

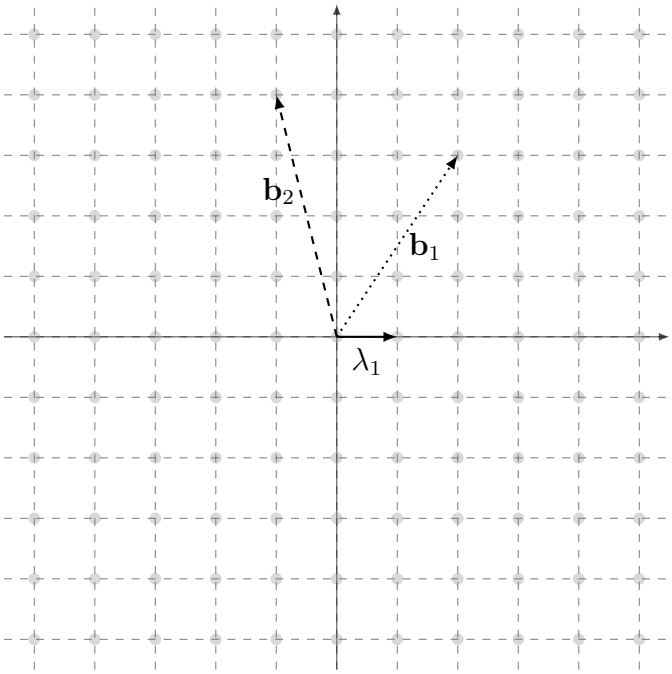

Figure 12: A two-dimensional lattice generated by basis vectors $\mathbf{b}_1$ and $\mathbf{b}_2$, with the shortest nonzero vector highlighted.

The grid of points represents all integer linear combinations of these basis vectors, forming a discrete subgroup of $\mathbb{R}^2$. After observing the geometric representation of the lattice generated by the basis vectors $\mathbf{b}_1$ and $\mathbf{b}_2$, identifying λ_1 becomes almost immediate. However, the example presented here is specifically chosen to illustrate how straightforward it is to determine the answer in $\mathbb{R}^2$. It is quite interesting to note that as the dimension increases, our ability to intuitively grasp and solve the problem diminishes. Notwithstanding, SVP is not merely about determining the length of λ_1; It also involves the following versions, which are not restricted to but include:[13]

- **Decision problem:** Given a lattice basis $\mathcal{B}$ and a real number threshold value $d > 0$, distinguish between the cases $\lambda_1 \leq d$ and $\lambda_1 > d$. For example, if we set $d = 2.5$, we check whether the shortest vector has a norm at most 2.5.

[13]The original versions of these problems are also characterized by the term *Exact*, e.g., Exact SVP.

If all nonzero lattice vectors have norms greater than 2.5, the answer is **NO**; otherwise, it's **YES**.

- **Calculation problem:** Given a lattice basis $\mathcal{B}$, find λ_1.
- **Search problem:** Given a lattice basis $\mathcal{B}$, find a nonzero vector $\mathbf{v} \in \mathcal{L}(\mathcal{B})$ such that $\|\mathbf{v}\| = \lambda_1$. This problem is sometimes considered more difficult than just computing the value of λ_1, as it requires finding integer coefficients.

Remark 25. Solving the Calculation version of SVP immediately implies solving the Decision version.

If we assume we have a solution to the Calculation SVP, meaning we can compute the shortest vector v_{short} in the lattice, then solving the Decision SVP becomes trivial:

1. Compute v_{short} using the assumed solver for Calculation SVP.
2. Compute its norm $\|v_{\text{short}}\|$.
3. Compare $\|v_{\text{short}}\|$ with d:
   - If $\|v_{\text{short}}\| \leq d$, output **YES**.
   - If $\|v_{\text{short}}\| > d$, output **NO**.

Thus, having a solution to the Calculation version immediately gives a way to solve the Decision version. Solving the Decision SVP does not necessarily solve the Calculation SVP. The Decision version only tells us *whether* a short vector exists below a certain threshold, but it does not tell us *what* the shortest vector is. To solve the Calculation SVP using a Decision oracle, one would typically need to perform multiple queries with different thresholds, which is not necessarily efficient.

Remark 26. Bounding the potential values of λ_1 is critical for efficient computation. Specifically, the range of λ_1 is constrained by Minkowski's theorem:

$$1 \leq \lambda_1(\mathcal{L}) \leq \sqrt{n} \cdot \det(\mathcal{B})^{\frac{1}{n}},$$

where $\det(\mathcal{B})$ is the determinant of the lattice basis. This bound, expressed as $2^{\text{poly}(|\mathcal{B}|)}$, ensures that polynomial-time algorithms are feasible since $\det(\mathcal{B})$ can be computed efficiently.

Open Problem 1. Determine the precise computational complexity of the Shortest Vector Problem.

Remark 27. Unlike several other NP-hard problems, it remains unknown whether there exists an instance of the Shortest Vector Problem that possesses an exponentially large number of solutions.

NP-hardness of SVP has only been demonstrated under randomized reductions.

Definition 60 (Approximate Shortest Vector Problem SVP_γ)**.** Let $\gamma(n)$ be a function of n. Given a lattice L of dimension n, find a nonzero vector $v \in L$ such that

$$\|v\| \leq \gamma(n)\|v_{\text{shortest}}\|.$$

Note that setting $\gamma = 1$ recovers the exact formulations of the problems, while increasing γ serves only to simplify them. Specifically, the computational version of SVP_γ transforms into its approximation variant, whereas the decision problem becomes the promise problem GapSVP_γ, with input restrictions.

Definition 61 (GapSVP_γ)**.** Given a lattice basis $\mathbf{B}$, and a positive integer d, and approximation function γ, decide which of the following holds:

$$\lambda(L) \leq d \quad \text{or} \quad \lambda(L) > \gamma\, d.$$

Open Problem 2. Determine whether $\text{SVP}_\gamma \leq \text{GapSVP}_\gamma$ *holds for some or all values of* $\gamma > 1$

Theorem 10. *The SVP is NP-hard* [14] *under randomized reductions [53].*

The computational complexity of the Shortest Vector Problem (SVP) exhibits a strong dependence on the lattice dimension n, with state-of-the-art algorithms for exact SVP achieving a time complexity of $2^{O(n)}$, and no known sub-exponential algorithms exist yet. This exponential scaling underpins the security of lattice-based cryptographic schemes, where high-dimensional lattices ensure computational hardness so far for both classical and quantum computation.

Gauss pioneered lattice reduction in 1801 for two-dimensional lattices, solving SVP in $O(1)$ time for fixed $n = 2$ [54]. The 1982 LLL algorithm by Lenstra, Lenstra, and Lovász marked a breakthrough, offering polynomial-time approximate SVP with an approximation factor $\gamma = 2^{O(n)}$ in $O(n^6 \cdot B)$ time, where B is the input bit length [55]. Schnorr's 1987 improvements and subsequent block reduction techniques refined this further [56], while Micciancio and Goldwasser's 2001 result established NP-hardness for $\gamma = O(1)$ [48]. Modern exact SVP solvers, such as Micciancio and Voulgaris's 2010 deterministic $2^{O(n)}$-time Voronoi cell method [57] and Aggarwal et al.'s 2011 sieving algorithm with $2^{2.377n+o(n)}$ time [58], remain impractical for large n. Ongoing research continues to optimize sieving and enumeration, yet the $2^{O(n)}$ barrier persists, reinforcing SVP's cryptographic relevance.

5.2 Closest Vector Problem (CVP)

Definition 62 (Closest Vector Problem)**.** Given a basis $\mathcal{B}$ of a lattice $\mathcal{L} = \mathcal{L}(\mathcal{B})$ and a target vector $\mathbf{t} \notin \mathcal{L}$, find a lattice vector $\mathbf{v} \in \mathcal{L}$ that minimizes the distance $\|\mathbf{v} - \mathbf{t}\|$.

[14] A rigorous proof of this theorem is presented in [53]

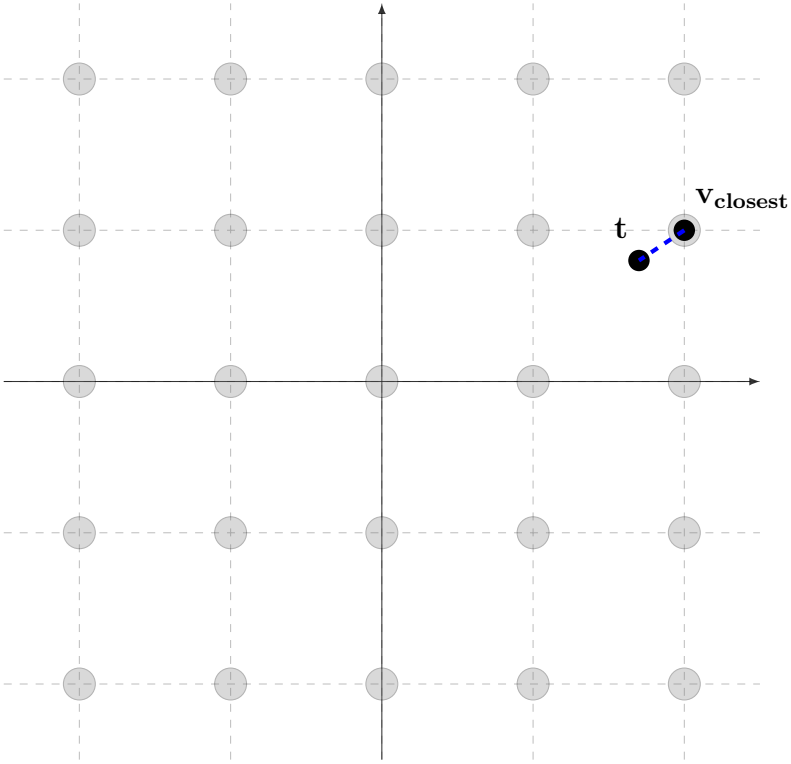

Figure 13: Exact Closest Vector Problem (CVP) in a Lattice. The target vector $\mathbf{t}$ is not part of the lattice. The CVP aims to identify the lattice vector $\mathbf{v}_{\text{closest}}$ that minimizes the Euclidean distance $\|\mathbf{t} - \mathbf{v}\|$.

Remark 28. The target vector $\mathbf{t}$ is generally not a lattice vector itself, otherwise the problem becomes trivial with $\mathbf{v} = \mathbf{t}$ and distance zero.

The Closest Vector Problem can be described in three versions similar to SVP:

- **Decision problem:** Given a lattice basis $\mathcal{B}$, a target vector $\mathbf{t}$, and a real $d > 0$, determine whether:

  $$\exists \mathbf{v} \in \mathcal{L}(\mathcal{B}) \text{ such that } \|\mathbf{v} - \mathbf{t}\| \leq d,$$

  or

  $$\forall \mathbf{v} \in \mathcal{L}(\mathcal{B}),\ \|\mathbf{v} - \mathbf{t}\| > d.$$

- **Optimization problem:** Given a lattice basis $\mathcal{B}$ and a target vector $\mathbf{t}$, find the closest lattice vector $\mathbf{v}^* \in \mathcal{L}(\mathcal{B})$ such that:

  $$\mathbf{v}^* = \arg \min_{\mathbf{v} \in \mathcal{L}(\mathcal{B})} \|\mathbf{v} - \mathbf{t}\|.$$ [15]

- **Calculation problem:** Given a lattice basis $\mathcal{B}$ and a target vector $\mathbf{t}$, find the minimum distance:

  $$\min_{\mathbf{v} \in \mathcal{L}(\mathcal{B})} \|\mathbf{v} - \mathbf{t}\|.$$

[15] Here, arg min denotes the argument (i.e., the value of $\mathbf{v}$) at which the minimum of the function $\|\mathbf{v} - \mathbf{t}\|$ is attained.

Remark 29. It is known that exact solutions are NP-hard, and efficient approximation algorithms exist only for specific cases or for larger approximation factors.

Definition 63 (Approximate Closest Vector Problem)**.** Given a basis $\mathcal{B}$ of a lattice $\mathcal{L} = \mathcal{L}(\mathcal{B})$, a target vector $\mathbf{t} \notin \mathcal{L}$, and an approximation factor $\gamma = \gamma(n) \geq 1$, the goal is to find a lattice vector $\mathbf{v} \in \mathcal{L}$ such that

$$\|\mathbf{v} - \mathbf{t}\| \leq \gamma \cdot \min_{\mathbf{v}' \in \mathcal{L}} \|\mathbf{v}' - \mathbf{t}\|,$$

where the minimum is taken over all lattice vectors $\mathbf{v}' \in \mathcal{L}$.

Definition 64. The Approximate CVP can be described in three variants:

- **Promise Problem (GapCVP$_\gamma$)**: Given a lattice basis $\mathcal{B}$, a target vector $\mathbf{t}$, and a positive real number d, determine whether:

  $$\exists \mathbf{v} \in \mathcal{L}(\mathcal{B}) \text{ such that } \|\mathbf{v} - \mathbf{t}\| \leq d,$$

  or

  $$\forall \mathbf{v} \in \mathcal{L}(\mathcal{B}),\ \|\mathbf{v} - \mathbf{t}\| > \gamma \cdot d.$$

- **Estimation Problem (EstCVP$_\gamma$)**: Given a lattice basis $\mathcal{B}$ and a target vector $\mathbf{t}$, estimate the minimum distance $\|\mathbf{v} - \mathbf{t}\|$ for $\mathbf{v} \in \mathcal{L}$ within a factor of γ, i.e., find d such that:

  $$\min_{\mathbf{v} \in \mathcal{L}} \|\mathbf{v} - \mathbf{t}\| \leq d \leq \gamma \cdot \min_{\mathbf{v} \in \mathcal{L}} \|\mathbf{v} - \mathbf{t}\|.$$

- **Search Problem (SearchCVP$_\gamma$)**: Given a lattice basis $\mathcal{B}$ and a target vector $\mathbf{t}$, find a lattice vector $\mathbf{v} \in \mathcal{L}(\mathcal{B})$ such that:

  $$\|\mathbf{v} - \mathbf{t}\| \leq \gamma \cdot \min_{\mathbf{u} \in \mathcal{L}(\mathcal{B})} \|\mathbf{u} - \mathbf{t}\|.$$

Bounded Distance Decoding (BDD): Problem can be conceptualized as a custom version of the CVP with an additional guarantee. While CVP involves finding the closest lattice vector to a given target vector $\mathbf{t}$ with no assurances about the proximity, BDD modifies this problem by promising that there exists a lattice vector $\mathbf{v}$ within a predefined distance δ from $\mathbf{t}$. Essentially, BDD asserts that the solution vector $\mathbf{v}$ is not just close to $\mathbf{t}$, but within a specific, bounded distance. This guarantee reduces the potential search space for solutions and provides a concrete framework for designing algorithms, potentially easing computational efforts under certain conditions.

The Closest Vector Problem shares similarities with the Shortest Vector Problem, however, CVP's complexity is more definitively understood, with proven NP-hardness, whereas SVP's NP-hardness remains conjectured under certain conditions. In 1986, van Emde Boas established that CVP in the ℓ_∞-norm is NP-hard, providing an early benchmark for its intractability in high dimensions [59]. This was followed by significant progress in 1998, when Goldreich, Micciancio, Safra, and Seifert proved

that approximating CVP within a factor of $n^{1/2-\epsilon}$ (where n is the lattice dimension) is NP-hard, tightening its inapproximability bound [60]. In 2001, Micciancio demonstrated that CVP remains NP-hard even for "almost trivial" approximation factors such as $2^{n/\log n}$, underscoring its intrinsic difficulty unless P = NP [48]. On the algorithmic front, Babai's method from the 1980s, leveraging the LLL algorithm, achieves an exponential approximation factor of $2^{O(n)}$ and remained the best practical approach by 2005, with no polynomial-time exact solution emerging [61]. In 2010, Micciancio and Voulgaris developed a deterministic $2^{O(n)}$-time algorithm for exact CVP, improving space efficiency but retaining exponential complexity [57].

5.3 Short Integer Solution (SIS)

The Short Integer Solution (SIS) problem is a fundamental building block for quantum-resistant cryptographic primitives, including one-way and collision-resistant hash functions, and digital signatures. The problem is parameterized by positive integers n and q, defining the group $\mathbb{Z}_q^n$, along with a positive real β and an integer m representing the number of group elements.

Definition 65 (SIS Problem)**.** Given an integer q and an integer matrix $A \in \mathbb{Z}_q^{n\times m}$, the goal is to find a *non-zero* vector $z \in \mathbb{Z}^m$ such that:

$$f_A(z) := Az = \sum a_i z_i = 0 \pmod q$$

where z is *short*, meaning its Euclidean norm satisfies:

$$\|z\| < \beta,$$

for a predefined bound β.

The primary hardness parameter is n, typically with $n \geq 100$. The modulus q and the bound β are assumed to be at least polynomially dependent on n, with $q > \beta$. Without the restriction on $\|z\|$, a solution could be easily found using Gaussian elimination.

The problem becomes easier with increasing m and harder with increasing n. The norm bound β and the number of vectors m must be sufficiently large to guarantee a solution's existence. The pigeonhole principle guarantees a solution when $\beta \geq \sqrt{m}$ and $m \geq \bar{m}$, where $\bar{m} = \lceil n \log q \rceil$.

We can assume without loss of generality that $m = \bar{m}$. If $m > \bar{m}$, we can always consider a submatrix of A with $\bar{m}$ columns.

Remark 30. Since the number of vectors $x \in \{0,1\}^m$ exceeds q^n, there must exist distinct vectors x, x' such that $Ax = Ax' \pmod q$. Their difference $z = x - x' \in \{0, \pm 1\}^m$ is a valid solution with $\|z\| < \beta$.

Definition 66 (Ajtai's One-Way Function)**.** Let m, n, q be integers. Define the function $f_{\mathbf{A}}$ with the following parameters:

- **Parameters:** $m, n, q \in \mathbb{Z}$.

- **Key:** $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$.
- **Input:** $\mathbf{x} \in \{0,1\}^m$.
- **Output:**

$$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} \mod q.$$

Theorem 11. *For $m > n \log q$, if lattice problems are hard to approximate in the worst case, then the function*

$$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} \mod q$$

is a one-way function. [12]

Remark 31. Inverting Ajtai's function is a lattice problem.

The easy case of this problem is to find an arbitrary integer solution $\mathbf{t}$ to the system of linear equations $\mathbf{A}\mathbf{t} = \mathbf{y} \mod q$. However, all solutions to $\mathbf{A}\mathbf{x} = \mathbf{y}$ take the form $\mathbf{t} + \mathcal{L}^{\perp}(\mathbf{A})$, where $\mathcal{L}^{\perp}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{A}\mathbf{x} = 0 \mod q\}$ is the lattice formed by the kernel of $\mathbf{A}$. The cryptanalytic challenge is then to find a small vector in $\mathbf{t} + \mathcal{L}^{\perp}(\mathbf{A})$, or equivalently, to find a lattice vector $\mathbf{v} \in \mathcal{L}^{\perp}(\mathbf{A})$ that is close to $\mathbf{t}$.

This corresponds to solving an instance of the CVP in a lattice defined by $\mathcal{L}^{\perp}(\mathbf{A})$, which is computationally hard. Since inverting Ajtai's function is an average-case instance of CVP where the lattice is chosen according to $\mathcal{L}^{\perp}(\mathbf{A})$, an efficient inversion algorithm would yield a solution to SIS.

Remark 32. Finding a short vector z ($\|z\| \leq \beta \ll q$ and $z \in \mathcal{L}^{\perp}(A)$) for a uniformly random $A \in \mathbb{Z}_q^{n \times m}$ is related to solving $\mathrm{GapSVP}_{\beta\sqrt{n}}$ and $\mathrm{SIVP}_{\beta\sqrt{n}}$ on any n-dimensional lattice.

Several variants of the SIS problem have been developed to enhance efficiency and security.

The **Inhomogeneous** version of SIS problem modifies the standard SIS definition by introducing a nonzero target vector. Instead of finding a vector $\mathbf{z}$ such that

$$A\mathbf{z} \equiv \mathbf{0} \mod q,$$

the goal is to find $\mathbf{z}$ satisfying

$$A\mathbf{z} \equiv \mathbf{b} \mod q$$

for a given vector $\mathbf{b}$.

The **Ring-SIS (R-SIS)** problem operates over polynomial rings, reducing the problem size and improving computational efficiency. It is defined over the ring

$$R_q = \mathbb{Z}_q[x]/(f(x)),$$

where $f(x)$ is typically a polynomial. Using rings allows for more efficient computations and reduces key sizes in cryptographic applications.

The **Module-SIS (M-SIS)** problem generalizes R-SIS by operating over module lattices in a polynomial ring. It is defined over the module

$$M_q = R_q^d = (\mathbb{Z}_q[x]/(f(x)))^d,$$

where d is the module rank.

The Short Integer Solution (SIS) problem is considered intractable for appropriately chosen parameters (e.g., dimension, modulus, and norm bound) against both classical and quantum algorithms, with no known solutions in polynomial or subexponential time. Recent research into structured variants, such as the Ring-SIS problem, has highlighted its efficiency for cryptographic applications, though it may exhibit reduced hardness compared to SIS. This motivated the development of the Module-SIS problem, which combines Ring-SIS's efficiency with provable hardness guarantees. Specifically, Module-SIS offers worst-case security reductions similar to SIS, where solving it on average is proven to be as difficult as solving worst-case lattice problems over module lattices [62].

- The approximation factor $\gamma = \beta \cdot \tilde{O}(\sqrt{n})$ was obtained by Micciancio and Regev, with values as small as $\tilde{O}(n)$ for suitable β, while the modulus was reduced to $q = \beta \cdot \tilde{O}(n\sqrt{m})$. [63]
- In 2008, the bound on q was improved to $\beta \cdot \tilde{O}(\sqrt{n})$ by Gentry, Peikert, and Vaikuntanathan, while maintaining γ from [64].
- In 2013, q was further reduced to $\beta \cdot n^{\varepsilon}$ ($\varepsilon > 0$) by Micciancio and Peikert, achieving an optimal bound (up to n^{ε}), as SIS instances have trivial solutions of norm q. [65]

5.4 Learning With Errors (LWE)

The Learning With Errors (LWE) problem, first introduced by Oded Regev in his seminal 2005 paper [13], has emerged as a cornerstone of modern cryptography due to its remarkable versatility. With the ability to underpin a broad range of cryptographic constructions, barring only a few exceptions, LWE derives its significance from Regev's groundbreaking reduction, which links worst-case lattice problems to an LWE framework resilient against quantum attacks. This quantum robustness has cemented LWE's status as a foundational pillar of post-quantum cryptography. In Regev's own words:

> "The LWE problem requires recovering a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ from a series of 'approximate' random linear equations over $\mathbf{s}$, each perturbed by a small additive error." [13]

At its core, the LWE problem depends fundamentally on discrete probability distributions, especially those arising from Gaussian functions defined over lattices.

These distributions determine the structure of the error terms, which, in turn, impart computational complexity to the problem. To precisely establish this theoretical foundation, following formal definitions are introduced:

Definition 67 (Discrete Gaussian Distribution)**.** Let $s > 0$, let $\mathcal{L} \subseteq \mathbb{R}^n$ be a lattice, and let $x \in \mathbb{R}^n$. The *discrete Gaussian distribution* over the coset $\mathcal{L} + x$ with parameter s, denoted by $D_{\mathcal{L}+x,s}$, assigns to each $w \in \mathcal{L} + x$ the probability

$$D_{\mathcal{L}+x,s}(w) = \frac{\rho_s(w)}{\sum_{y \in \mathcal{L}+x} \rho_s(y)},$$

where

$$\rho_s(y) = \exp\left(-\pi \frac{\|y\|^2}{s^2}\right).$$

Remark 33. The error terms are typically sampled from a discrete Gaussian distribution over the integers, denoted $\chi = D_{\mathbb{Z},\sigma}$, where σ serves as the width parameter akin to s.

LWE's defining feature is the incorporation of error terms, typically sampled from a discrete Gaussian distribution as described above. These errors give rise to two primary variants of the problem: the search version, which seeks to recover the secret, and the decision version, which involves distinguishing noisy equations from random pairs. Consider a sequence of noisy linear equations of the form:

- $\mathbf{a}_i \in \mathbb{Z}_q^n$, a uniformly random vector representing the coefficients of the i-th equation,
- $\mathbf{s} \in \mathbb{Z}_q^n$, the secret vector to be recovered,
- e_i, a small error term drawn from a Discrete Gaussian distribution,
- $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i \pmod{q}$, the noisy result,
- q, a large prime modulus governing the arithmetic.

Were the error terms absent (i.e., $e_i = 0$), recovering $\mathbf{s}$ would be straightforward: with approximately n linearly independent equations, Gaussian elimination could solve for $\mathbf{s}$ in polynomial time. However, the introduction of even small errors transforms this into a formidable challenge. Applying Gaussian elimination amplifies the error through linear combinations, often rendering the resulting system uninformative about $\mathbf{s}$. This amplification is the crux of LWE's conjectured hardness.

Remark 34. A special case of interest occurs when $q = 2$ and the error distribution is Bernoulli over $\{0, 1\}$. This configuration aligns with the *Learning Parity with Noise* (LPN) problem, a well-known analogue in computational complexity.

To precisely delineate the LWE problem, we define its two main variants:

Definition 68 (Search LWE)**.** Given a dimension n, a prime modulus q, an error distribution $\chi = D_{\mathbb{Z},\sigma}$, and m independent pairs $\{(\mathbf{a}_i, b_i)\}_{i=1}^m$, where:

- $\mathbf{a}_i \in \mathbb{Z}_q^n$ is uniformly random,

- $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i \pmod{q}$, with a fixed secret $\mathbf{s} \in \mathbb{Z}_q^n$ and $e_i \sim \chi$,

the objective is to recover $\mathbf{s}$.

Definition 69 (Decision LWE)**.** Given the same parameters, distinguish between:

- LWE samples: $(\mathbf{a}_i, b_i)$ as defined in Search LWE,

- Uniform samples: $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, chosen uniformly at random.

Theorem 12. *(Regev's Theorem) Let $m = poly(n)$ be any polynomial in n, $q \leq 2^{poly(n)}$ be any modulus, and let χ be a Gaussian error distribution with parameter $\alpha q \geq 2\sqrt{n}$, where $0 < \alpha < 1$. Then, solving the decision-LWE$_{n,q,\chi,m}$ problem is at least as hard as quantumly solving the GapSVP$_\gamma$ and SIVP$_\gamma$ problems on any n-dimensional lattice, for some approximation factor $\gamma = \widetilde{O}\left(\frac{n}{\alpha}\right)$.*

In [13], the above theorem is proved in two main parts: First, search-LWE is shown to be at least as hard as worst-case lattice problems via a quantum reduction. Second, decision-LWE is proved to be equivalent to search-LWE via an elementary classical reduction.

Remark 35. The essential hardness guarantee comes from the approximation factor γ, which degrades with the inverse error rate $\frac{1}{\alpha}$ of the LWE problem.

The quantum aspect of this reduction is particularly significant. To date, no specialized quantum algorithms have been found for solving GapSVP$_\gamma$ or SIVP$_\gamma$ that offer meaningful advantages over classical methods, apart from general quantum speedups such as Grover's algorithm. The hardness guarantee relies crucially on the approximation factor γ, which is closely related to the inverse error rate $\frac{1}{\alpha}$ of the LWE problem. The absence of quantum algorithms capable of significantly outperforming classical approaches further bolsters the post-quantum resilience of LWE. This observation sets the foundation for introducing Regev's Public-Key Encryption Scheme.

Algorithm 3 Regev's Public Key Encryption Scheme

Alice	**Bob**
Key Generation Generate a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ uniformly at random. For $i = 1$ to m, sample $(\mathbf{a}_i, b_i)$ from the LWE distribution with secret $\mathbf{s}$, modulus q, and error parameter α. Publish the public key $\{(\mathbf{a}_i, b_i)\}_{i=1}^m$.	
	Encryption Given a message bit $b \in \{0, 1\}$, choose a random subset $S \subseteq [m]$ uniformly at random. Compute the encryption components: $$\mathbf{a} = \sum_{i \in S} \mathbf{a}_i$$ $$b' = \begin{cases} \sum_{i \in S} b_i & \text{if } b = 0, \\ \left\lfloor \frac{q}{2} \right\rfloor + \sum_{i \in S} b_i & \text{if } b = 1. \end{cases}$$ Send ciphertext $(\mathbf{a}, b')$ to Alice.
Decryption Upon receiving ciphertext $(\mathbf{a}, b')$, compute: $$v = b' - \langle \mathbf{a}, \mathbf{s} \rangle \mod q$$ If v is closer to 0 than to $\left\lfloor \frac{q}{2} \right\rfloor$ modulo q, output decrypted bit 0. Otherwise, output decrypted bit 1.	

To ensure both security and correctness, certain parameter choices are adopted. The modulus q is selected as a prime number within the range of n^2 to $2n^2$. The number of equations m is set according to the relation $m = 1.1 \cdot n \log q$, ensuring a sufficient quantity for reliable encryption.

The noise parameter is defined as $\alpha = \frac{1}{\sqrt{n} \log^2 n}$, balancing the trade-off between security and correctness. In all subsequent operations, additions are performed modulo q.

Without the presence of error, the term $b - \langle \mathbf{a}, \mathbf{s} \rangle$ would deterministically evaluate to either 0 or $\lfloor \frac{q}{2} \rfloor$, contingent on the encrypted bit. As a result, decryption would be guaranteed to be correct. However, decryption errors are introduced when the cumulative error across all subsets S exceeds $\frac{q}{4}$.

Given that the encryption process involves summing at most m independent normal error terms, each characterized by a standard deviation of αq, the resulting standard deviation of the sum is bounded by $\sqrt{m}\alpha q$. Based on the parameter choices, this value remains less than $\frac{q}{\log n}$. Standard probabilistic analysis indicates that the likelihood of the accumulated error surpassing $\frac{q}{4}$ is negligible, thereby ensuring the overall correctness of the cryptosystem under the specified parameter conditions.

The versatility of the Learning With Errors (LWE) problem extends well beyond its foundational role in public-key encryption schemes. As a robust and quantum-resistant hardness assumption, LWE underpins a wide range of cryptographic primitives and protocols, including **Fully Homomorphic Encryption** (FHE), often regarded as the *holy grail* of cryptography [66]. Beyond its theoretical significance, LWE serves as a cornerstone of modern lattice-based cryptography, offering strong security guarantees rooted in worst-case lattice hardness assumptions. However, its direct use in cryptographic schemes presents practical challenges, particularly in terms of key size and computational efficiency.

To address these limitations, structured variants such as Ring-LWE and Module-LWE have been introduced, leveraging algebraic properties to achieve more efficient implementations while preserving security. These refinements have been instrumental in enabling practical post-quantum cryptosystems, including those standardized in the NIST PQC competition. As research continues, further optimizations and new hardness assumptions may enhance the applicability of LWE-based cryptography, ensuring long-term security against both classical and quantum adversaries.

6 Classical Algorithms for Lattice Reduction

At its core, lattice reduction aims to transform a given basis of a lattice into another basis of the same lattice, where the new basis vectors are shorter and more orthogonal to each other. Lattice reduction has a rich history, with roots tracing back to the works of Carl Friedrich Gauss on two-dimensional lattices. The development of the LLL (Lenstra-Lenstra-Lovász) algorithm in the 1980s was a significant milestone, making lattice reduction feasible for higher dimensions and sparking interest in its applications in cryptography and number theory.

A lattice $\mathcal{L}$ with basis $B = \{b_1, \ldots, b_n\}$ undergoes lattice reduction to find a new basis $B' = \{b'_1, \ldots, b'_n\}$ such that the span of B' is equal to the span of B, which is $\mathcal{L}$. Additionally, the vectors in B' are shorter than those in B, meaning $\|b'_i\| \leq \|b_i\|$ for $i = 1, \ldots, n$. Furthermore, the vectors in B' are more orthogonal to each other compared to those in B.

Lattice reduction serves as a fundamental tool for assessing the quality of a lattice basis and, consequently, the security of lattice-based cryptosystems. The effectiveness of lattice reduction algorithms is typically measured using specific quality metrics that evaluate the length and structure of the reduced basis.

- **Orthogonality Defect**: Defined as $\delta(B) = \frac{\prod_{i=1}^{n} \|b_i\|}{\det(\mathcal{L})}$, where $B = \{b_1, \ldots, b_n\}$ is a basis of the lattice $\mathcal{L}$. This quantity measures how far the basis vectors deviate from being mutually orthogonal. For an orthonormal basis, the product of the norms equals the determinant, yielding $\delta(B) = 1$. Larger values indicate increasing linear dependence among the vectors.
- **Hermite Factor**: Given by $\gamma = \frac{\|b_1\|}{(\det(\mathcal{L}))^{1/n}}$, where b_1 is the shortest vector in the reduced basis.
- **Approximation Factor**: The ratio between the length of the shortest vector found by the reduction algorithm and the actual shortest vector in the lattice.

Gaussian heuristic provides an estimate for the expected shortest vector length in a random integer lattice. For a lattice $\mathcal{L}$ with determinant $\det(\mathcal{L})$ and dimension n, the expected length of the shortest vector $\lambda_1(\mathcal{L})$ is approximately given by:

$$\lambda_1(\mathcal{L}) \approx \sqrt{\frac{n}{2\pi e}} \cdot (\det(\mathcal{L}))^{1/n}.$$

This heuristic serves as a useful benchmark for evaluating the performance of lattice reduction algorithms and their approximation to the shortest vector in practice.

6.1 Gaussian Lattice Reduction

Gaussian lattice reduction (GLR), also known as Lagrange-Gauss reduction, is an algorithm for reducing the basis of two-dimensional lattices. It simplifies the basis vectors while maintaining lattice structure, and it serves as a foundation for understanding more advanced lattice reduction techniques. Gaussian reduction operates

by iteratively improving the basis of a 2D lattice. The goal is to find a reduced basis where the vectors are nearly orthogonal and the shorter vector is the shortest non-zero vector in the lattice.

Definition 70. (Gaussian Lattice Reduction) Let $\mathcal{L} \subset \mathbb{R}^2$ be a lattice. Vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ are linearly independent basis vectors. The Gaussian algorithm described below terminates and returns a good basis of $\mathcal{L}$

Algorithm 4 Gaussian Lattice Reduction

loop
  if $\|\mathbf{v}_2\| < \|\mathbf{v}_1\|$ **then**
    Swap $\mathbf{v}_1$ and $\mathbf{v}_2$
  end if
  Compute $m = \left\lfloor \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\|^2} \right\rfloor$
  if $m = 0$ **then**
    return the basis vectors $\mathbf{v}_1$ and $\mathbf{v}_2$
  end if
  Replace $\mathbf{v}_2$ with $\mathbf{v}_2 - m\mathbf{v}_1$
end loop

Remark 36. Gaussian algorithm always terminates [54] in a finite number of steps, as the norm of v_1 strictly decreases in each iteration, ensuring termination.

Remark 37. The shorter vector of the reduced basis is guaranteed to be the shortest non-zero vector of the lattice.

Remark 38. Gaussian-reduced basis achieves the optimal Hermite factor of $\frac{4}{3}$ in two dimensions.

Remark 39. The time complexity of Gaussian reduction is

$$O((\log \max(\|v_1\|, \|v_2\|))^2)$$

where $\|v_1\|$ and $\|v_2\|$ are the lengths of the initial basis vectors, making the algorithm highly efficient for two-dimensional lattices.

GLR works in two dimensions because each basis vector is reduced with respect to another vector via Gram-Schmidt projection. In three dimensions, $\mathbf{v}_3$ would need to be reduced not just against one vector, but against the entire subspace spanned by $\mathbf{v}_1$ and $\mathbf{v}_2$. Pairwise reductions cannot achieve this because they do not account for how $\mathbf{v}_3$ interacts with the plane defined by $\mathbf{v}_1$ and $\mathbf{v}_2$. Furthermore, pairwise swapping in 3 dimensions can lead to cyclic behavior, where the algorithm repeatedly swaps and reduces vectors without converging to a reduced basis. Although primarily of historical interest, Gaussian reduction remains significant since:

- It serves as a conceptual starting point for more complex algorithms.
- It provides a benchmark for understanding lattice reduction principles.

The only required parameters for the algorithm are the norms and dot products of the basis vectors.

Example 15. We provide an example of Gauss's lattice reduction algorithm.

$$\mathbf{v}_1 = (1446840, 1420069),$$
$$\mathbf{v}_2 = (6512996, 6394464).$$

This initial basis will be used to demonstrate the algorithm.

Initialization
The initial lattice basis is:

$$\mathbf{v}_1 = (1446840, 1420069), \quad \mathbf{v}_2 = (6512996, 6394464).$$

The Gram-Schmidt coefficient is computed as:

$$\mu = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\|^2} = \frac{18503843230656}{4109941950361} \approx 5.$$

Using this coefficient, we update $\mathbf{v}_2$ as follows:

$$\mathbf{v}_2 \leftarrow \mathbf{v}_2 - \mu\mathbf{v}_1 = (6512996, 6394464) - 5(1446840, 1420069) = (-721204, -705881).$$

Step 1: Swapping Vectors.
The current basis is:

$$\mathbf{v}_1 = (1446840, 1420069), \quad \mathbf{v}_2 = (-721204, -705881).$$

Since $\|\mathbf{v}_2\| < \|\mathbf{v}_1\|$, we swap the vectors:

$$\mathbf{v}_1 = (-721204, -705881), \quad \mathbf{v}_2 = (1446840, 1420069).$$

The new Gram-Schmidt coefficient is:

$$\mu = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\|^2} = \frac{-185987865559}{92582108707} \approx -2.$$

The second basis vector is updated as:

$$\mathbf{v}_2 \leftarrow \mathbf{v}_2 - \mu\mathbf{v}_1 = (1446840, 1420069) - (-2)(-721204, -705881) = (4432, 8307).$$

Step 2: Further Reduction.
The current basis is:

$$\mathbf{v}_1 = (-721204, -705881), \quad \mathbf{v}_2 = (4432, 8307).$$

Swapping again, the vectors become:

$$\mathbf{v}_1 = (4432, 8307), \quad \mathbf{v}_2 = (-721204, -705881).$$

The Gram-Schmidt coefficient is:

$$\mu = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\|^2} = \frac{-9060129595}{88648873} \approx -102.$$

The updated second basis vector is:

$$\mathbf{v}_2 \leftarrow \mathbf{v}_2 - \mu\mathbf{v}_1 = (-721204, -705881) - (-102)(4432, 8307) = (-269140, 141433).$$

Final Reduced Basis
The algorithm terminates when the reduction condition is met. The final reduced basis is:

$$\mathbf{v}_1 = (4432, 8307), \quad \mathbf{v}_2 = (-269140, 141433).$$

This reduced basis minimizes the vector lengths while maintaining linear independence, thereby fulfilling the goal of Gauss's lattice reduction algorithm. Reduced basis is a solution to SVP for the lattice $\mathcal{L}$.

Initially, the basis vectors were nearly parallel, forming an angle of 0.009°. After reduction, the angle increased to 90.36°, demonstrating GLR's ability to produce a nearly orthogonal basis. The norm of the first vector decreased from 2.03×10^6 to 9415.35, a reduction factor of 215.32, highlighting the compactness achieved.

6.2 Lenstra-Lenstra-Lovász (LLL) Algorithm

The LLL algorithm, introduced named after its inventors Arjen Lenstra, Hendrik Lenstra, and László Lovász in 1982 [55], is a polynomial-time algorithm for lattice basis reduction. It transforms a given basis of a lattice into a reduced basis with relatively short and nearly orthogonal vectors. The algorithm does two things: It shortens the vectors by removing unnecessary overlaps with earlier ones. It swaps and adjusts them to make them less slanted and closer to orthogonal. By repeating these steps, it produces a new set of shorter, well-behaved vectors that still describe the same grid. Given a lattice basis $B = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ in $\mathbb{R}^n$, the algorithm produces a new basis $B' = \{\mathbf{v}'_1, \mathbf{v}'_2, \ldots, \mathbf{v}'_n\}$ satisfying two criteria:

1. **Size Reduction:** The basis vectors are as short as possible.
2. **Lovász Condition:** The basis vectors are as orthogonal as possible.

To develop an improved basis, the process is initiated by constructing an orthogonal basis using the Gram–Schmidt procedure. The process begins with $\mathbf{v}_1^* = \mathbf{v}_1$, and for $i \geq 2$, $\mathbf{v}_i^*$ is defined as

$$\mathbf{v}_i^* = \mathbf{v}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{v}_j^*, \text{ where } \mu_{i,j} = \frac{\mathbf{v}_i \cdot \mathbf{v}_j^*}{\|\mathbf{v}_j^*\|^2} \text{ for } 1 \leq j \leq i-1.$$

Remark 40. Collection of Gram–Schmidt vectors $\mathcal{B}^*$ generated during the procedure is indeed an orthogonal basis for the space spanned however it is not a basis for $\mathcal{L}$ since the Gram-Schmidt procedure involves non-integral coefficients.

Nevertheless, an essential property of these two bases is that they share the same *determinant*.

Proposition 1. *Let $B = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ represent a basis for a lattice L, and let $B^* = \{\mathbf{v}_1^*, \mathbf{v}_2^*, \ldots, \mathbf{v}_n^*\}$ denote the Gram–Schmidt orthogonal basis corresponding to B. Then, the determinant of L is given by*

$$\det(L) = \prod_{i=1}^{n} \|\mathbf{v}_i^*\|.$$

Proof. Let F denote the matrix whose i-th row corresponds to the coordinates of the basis vector $\mathbf{v}_i$ in some fixed coordinate system. The determinant of the lattice L is defined as the absolute value of the determinant of this matrix:

$$\det(L) = |\det F|.$$

Next, consider the analogous matrix F^*, whose i-th row corresponds to the Gram–Schmidt orthogonalized vector $\mathbf{v}_i^*$. The relationship between F and F^* can be expressed as

$$MF^* = F,$$

where M is the change-of-basis matrix arising from the Gram–Schmidt orthogonalization process. The entries of M are defined by

$$M_{i,j} = \begin{cases} 1 & \text{if } i = j, \\ \mu_{i,j} & \text{if } i > j, \\ 0 & \text{if } i < j, \end{cases}$$

where $\mu_{i,j} = \frac{\mathbf{v}_i \cdot \mathbf{v}_j^*}{\|\mathbf{v}_j^*\|^2}$ represents the projection coefficients in the Gram–Schmidt process. Note that M is a lower triangular matrix with ones on its diagonal, so its determinant satisfies

$$\det(M) = 1.$$

where M is the change of basis matrix given by

$$M = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ \mu_{2,1} & 1 & \cdots & 0 & 0 \\ \mu_{3,1} & \mu_{3,2} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \mu_{n,1} & \mu_{n,2} & \cdots & \mu_{n,n-1} & 1 \end{pmatrix}.$$

Substituting this into the determinant expression, we find

$$\det(L) = |\det F| = |\det(MF^*)| = |(\det M)(\det F^*)| = |\det F^*|.$$

Finally, note that the rows of F^*, which are the vectors $\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*$, are pairwise orthogonal. Thus, the determinant of F^* is given by the product of the norms of its rows, yielding

$$\det(L) = \prod_{i=1}^{n} \|\mathbf{v}_i^*\|.$$

□

Definition 71. (LLL Reduction) Let $\mathcal{B} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ be a basis for a lattice L, and let $\mathcal{B}^* = \{\mathbf{v}_1^*, \mathbf{v}_2^*, \ldots, \mathbf{v}_n^*\}$ be the associated Gram-Schmidt orthogonal basis as described in Theorem 7.13. The basis $\mathcal{B}$ is said to be **LLL reduced** if it satisfies the following two conditions:

1. **Size Condition**

$$|\mu_{i,j}| = \left| \frac{\mathbf{v}_i \cdot \mathbf{v}_j^*}{\|\mathbf{v}_j^*\|^2} \right| \leq \frac{1}{2} \quad \text{for all } 1 \leq j < i \leq n.$$

2. **Lovász Condition**

$$\|\mathbf{v}_i^*\|^2 \geq \left(\frac{3}{4} - \mu_{i,i-1}^2 \right) \|\mathbf{v}_{i-1}^*\|^2 \quad \text{for all } 1 < i \leq n.$$

Definition 72. Let V be a vector space, and let $W \subset V$ be a subspace of V. The orthogonal complement of W in V, denoted by $W^\perp$, is defined as

$$W^\perp = \{\mathbf{v} \in V : \mathbf{v} \cdot \mathbf{w} = 0 \text{ for all } \mathbf{w} \in W\}.$$

It can be shown that $W^\perp$ is itself a subspace of V. Furthermore, every vector $\mathbf{v} \in V$ can be uniquely expressed as a sum $\mathbf{v} = \mathbf{w} + \mathbf{w}^\perp$, where $\mathbf{w} \in W$ and $\mathbf{w}^\perp \in W^\perp$.

The Lovász condition can be stated as the inequality

$$\|\mathbf{v}_i^* + \mu_{i,i-1}\mathbf{v}_{i-1}^*\|^2 \geq \frac{3}{4}\|\mathbf{v}_{i-1}^*\|^2,$$

or equivalently as

$$\begin{aligned} &\|\text{Projection of } \mathbf{v}_i \text{ onto } \text{Span}(\mathbf{v}_1, \ldots, \mathbf{v}_{i-2})^\perp\| \\ \geq \frac{3}{4}&\|\text{Projection of } \mathbf{v}_{i-1} \text{ onto } \text{Span}(\mathbf{v}_1, \ldots, \mathbf{v}_{i-2})^\perp\|. \end{aligned}$$

Remark 41. Size reduction ensures that the projection of the vector $\mathbf{v_i}$ onto the direction of $\mathbf{v_j^*}$ is small, bounded by $\mathbf{1/2}$.

Remark 42. The Lovász condition ensures that the Gram–Schmidt orthogonalized vectors $\mathbf{v}_i^*$ are not disproportionately small relative to the previous vectors $\mathbf{v}_{i-1}^*$, preventing an ill-conditioned basis.

Remark 43. The Lovász condition also guarantees that the projection of $\mathbf{v}_i$ onto the orthogonal complement of the span of earlier vectors remains sufficiently large compared to that of $\mathbf{v}_{i-1}$.

This condition stabilizes the lattice basis and maintains a reduced and computationally efficient structure. The significant contribution by Lenstra, Lenstra, and Lovász [55] establishes that an LLL-reduced basis possesses desirable properties and can be efficiently computed in polynomial time.

Theorem 13. *Let $\mathcal{L}$ be a lattice of dimension n. Any LLL reduced basis $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ for $\mathcal{L}$ has the following two properties:*

$$\prod_{i=1}^{n} \|\mathbf{v}_i\| \leq 2^{n(n-1)/4} \det(\mathcal{L}),$$

$$\|\mathbf{v}_j\| \leq 2^{(i-1)/2} \|\mathbf{v}_i^*\| \quad \textit{for all } 1 \leq j \leq i \leq n.$$

Further, the initial vector in an LLL reduced basis satisfies

$$\|\mathbf{v}_1\| \leq 2^{(n-1)/4} \det(\mathcal{L}^{1/n}) \quad \textit{and} \quad \|\mathbf{v}_1\| \leq 2^{(n-1)/2} \min_{0 \neq \mathbf{v} \in \mathcal{L}} \|\mathbf{v}\|.$$

Thus an LLL reduced basis solves approximate version of SVP problem to within a factor of $2^{(n-1)/2}$.

Proof. Under the Lovász condition and the restriction $|\mu_{i,i-1}| \leq \frac{1}{2}$,

$$\|\mathbf{v}_i^*\|^2 \geq \left(\frac{3}{4} - \mu_{i,i-1}^2\right) \|\mathbf{v}_{i-1}^*\|^2 \geq \frac{1}{2} \|\mathbf{v}_{i-1}^*\|^2.$$

Applying this inequality repeatedly yields the estimate

$$\|\mathbf{v}_j^*\|^2 \leq 2^{i-j} \|\mathbf{v}_i^*\|^2.$$

Now, consider

$$\|\mathbf{v}_i\|^2 = \left\| \mathbf{v}_i^* + \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{v}_j^* \right\|^2.$$

Using the orthogonality of $\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*$, this becomes

$$\|\mathbf{v}_i\|^2 = \|\mathbf{v}_i^*\|^2 + \sum_{j=1}^{i-1} \mu_{i,j}^2 \|\mathbf{v}_j^*\|^2.$$

Substituting $|\mu_{i,j}| \leq \frac{1}{2}$ gives

$$\|\mathbf{v}_i\|^2 \leq \|\mathbf{v}_i^*\|^2 + \sum_{j=1}^{i-1} \frac{1}{4} \|\mathbf{v}_j^*\|^2.$$

Using the earlier estimate for $\|\mathbf{v}_j^*\|^2$, we have

$$\|\mathbf{v}_i\|^2 \le \|\mathbf{v}_i^*\|^2 + \sum_{j=1}^{i-1} 2^{i-j-2}\|\mathbf{v}_i^*\|^2.$$

This simplifies to

$$\|\mathbf{v}_i\|^2 = \left(1 + 2^{i-1}\right)\|\mathbf{v}_i^*\|^2 \le 2^{i-1}\|\mathbf{v}_i^*\|^2.$$

Multiplying over $1 \le i \le n$, we obtain

$$\prod_{i=1}^{n} \|\mathbf{v}_i\|^2 \le \prod_{i=1}^{n} 2^{i-1}\|\mathbf{v}_i^*\|^2 = 2^{n(n-1)/2} \prod_{i=1}^{n} \|\mathbf{v}_i^*\|^2.$$

This leads to

$$\prod_{i=1}^{n} \|\mathbf{v}_i\|^2 \le 2^{n(n-1)/2} (\det \mathcal{L})^2.$$

Taking square roots completes the first part of the proof.

For any $j \le i$, using the estimates derived earlier, we find

$$\|\mathbf{v}_j\|^2 \le 2^{j-1}\|\mathbf{v}_j^*\|^2 \le 2^{j-1} \cdot 2^{i-j}\|\mathbf{v}_i^*\|^2 = 2^{i-1}\|\mathbf{v}_i^*\|^2.$$

Taking square roots yields

$$\|\mathbf{v}_j\| \le 2^{(i-1)/2}\|\mathbf{v}_i^*\|.$$

Setting $j = 1$, multiplying over $1 \le i \le n$, and applying the determinant formula gives

$$\|\mathbf{v}_1\|^n \le \prod_{i=1}^{n} 2^{(i-1)/2}\|\mathbf{v}_i^*\| = 2^{n(n-1)/4} \prod_{i=1}^{n} \|\mathbf{v}_i^*\| = 2^{n(n-1)/4} \det(\mathcal{L}).$$

Taking nth roots provides the first estimate for $\|\mathbf{v}_1\|$.

Finally, consider a nonzero vector $\mathbf{v} \in \mathcal{L}$ expressed as

$$\mathbf{v} = \sum_{j=1}^{i} a_j \mathbf{v}_j = \sum_{j=1}^{i} b_j \mathbf{v}_j^*,$$

where $a_i \neq 0$ and $|a_i| \ge 1$. The orthogonality of $\mathbf{v}_1^*, \ldots, \mathbf{v}_i^*$ ensures

$$\mathbf{v} \cdot \mathbf{v}_i^* = a_i \mathbf{v}_i \cdot \mathbf{v}_i^* = b_i \|\mathbf{v}_i^*\|^2,$$

from which $a_i = b_i$. Consequently, $|b_i| \ge 1$, and thus

$$\|\mathbf{v}\|^2 = \sum_{j=1}^{i} b_j^2 \|\mathbf{v}_j^*\|^2 \ge b_i^2 \|\mathbf{v}_i^*\|^2 \ge \|\mathbf{v}_i^*\|^2 \ge 2^{-(i-1)}\|\mathbf{v}_1\|^2 \ge 2^{-(n-1)}\|\mathbf{v}_1\|^2.$$

Taking square roots gives the second estimate for $\|\mathbf{v}_1\|$. □

The LLL algorithm transforms a basis $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ to satisfy the Size Condition by iteratively subtracting integer multiples of earlier vectors $\mathbf{v}_1, \ldots, \mathbf{v}_{k-1}$ from $\mathbf{v}_k$. After size reduction, the Lovász condition is checked; if it fails, the basis is reordered and reduced again.

Theorem 14 (LLL Algorithm)**.** *Let $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ be a basis for a lattice $\mathcal{L} \subseteq \mathbb{Z}^n$, and let $\delta \in (0.25, 1]$ be a parameter. The algorithm terminates after a finite number of iterations and outputs an LLL-reduced basis B' for $\mathcal{L}$.*

Let $\mathcal{B} = \max \|\mathbf{v}_i\|$ denote the maximum norm of the input basis vectors. The main loop of the algorithm executes at most $O(n^2 \log n + n^2 \log B)$ iterations.

In Theorem 14, the complexity $O(n^2 \log n + n^2 \log B)$ describes the number of iterations the LLL algorithm performs, where n is the lattice dimension, and B is the maximum norm of the input basis vectors. Using the logarithmic property $\log(a) + \log(b) = \log(ab)$, this sum can be rewritten as $O(n^2 \log(n \cdot B))$. To simplify further, this is often expressed as $O(n^2 \log X)$, where X represents the combined effect of n and B. This notation abstracts the relationship between n and B while preserving the same asymptotic behavior, focusing on the dominant growth rate.

Remark 44. As a result, the LLL algorithm runs in polynomial time. A detailed proof of LLL complexity is provided in Galbraith's work [67].

Algorithm 5 LLL Basis Reduction Algorithm

Require: Lattice basis $B = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$
Ensure: LLL-reduced basis B'

Set $k \leftarrow 2$
Set $\mathbf{v}_1^* \leftarrow \mathbf{v}_1$
while $k \leq n$ **do**
    for $j = k-1, k-2, \ldots, 1$ **do** ▷ Size Reduction
        $\mathbf{v}_k^* \leftarrow \mathbf{v}_k - \lfloor \mu_{k,j} \rceil \mathbf{v}_j$
    end for
    if $\|\mathbf{v}_k^*\|^2 \geq \left(\frac{3}{4} - \mu_{k,k-1}^2\right) \|\mathbf{v}_{k-1}^*\|^2$ **then** ▷ Lovász Condition
        $k \leftarrow k + 1$
    else
        Swap $\mathbf{v}_{k-1}$ and $\mathbf{v}_k$ ▷ Swap Step
        $k \leftarrow \max(k-1, 2)$
    end if
end while
return B

Example 16. The application of the LLL algorithm will be illustrated using a toy example on a 4-dimensional lattice $\mathcal{L}$. with the $\delta = 0.75$ Due to the length and repetitive nature of the calculations, only one iteration is explicitly shown. The remaining iterations follow analogously.

$$M = \begin{pmatrix} 2 & 7 & 12 & 9 \\ 15 & 15 & 2 & 3 \\ 8 & 1 & 18 & 19 \\ 6 & 10 & 2 & 7 \end{pmatrix}$$

The input basis is:

$$\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4\} = \left\{ \begin{pmatrix} 2 \\ 15 \\ 8 \\ 6 \end{pmatrix}, \begin{pmatrix} 7 \\ 15 \\ 1 \\ 10 \end{pmatrix}, \begin{pmatrix} 12 \\ 2 \\ 18 \\ 2 \end{pmatrix}, \begin{pmatrix} 9 \\ 3 \\ 19 \\ 7 \end{pmatrix} \right\}.$$

- Start by setting $k = 2$ and $\mathbf{v}_1^* = \mathbf{v}_1$.
- Check the $k = 2$, $n = 4$, since $k \leq n$ continue next step.
- Compute Gram–Schmidt orthogonalization $\{v_1^*, v_2^*, v_3^*, v_4^*\}$ and coefficients $\mu_{i,j}$.

Compute v_2^*:

$$v_1^* = v_1 = \begin{pmatrix} 2 \\ 15 \\ 8 \\ 6 \end{pmatrix}.$$

$$\mu_{2,1} = \frac{v_2 \cdot v_1^*}{\|v_1^*\|^2} = \frac{307}{329} \approx 0.933.$$

Round $\mu_{2,1} \approx 1$. and size reduce.

$$v_2^* = v_2 - \lfloor \mu_{2,1} \rceil v_1^* = \begin{pmatrix} 5 \\ 0 \\ -7 \\ 4 \end{pmatrix}.$$

Compute v_3^*:

$$\mu_{3,1} = \frac{v_3 \cdot v_1^*}{\|v_1^*\|^2} = \frac{210}{329} \approx 0.638.$$
$$\mu_{3,2} = \frac{v_3 \cdot v_2^*}{\|v_2^*\|^2} = \frac{-58}{90} \approx -0.644.$$

Round $\mu_{3,1} \approx 1$, $\mu_{3,2} \approx -1$ and size reduce.

$$v_3^* = v_3 - \lfloor \mu_{3,1} \rceil v_1^* - \lfloor \mu_{3,2} \rceil v_2^* = \begin{pmatrix} 15 \\ -13 \\ 3 \\ 0 \end{pmatrix}.$$

Compute v_4^*:

$$\mu_{4,1} = \frac{v_4 \cdot v_1^*}{\|v_1^*\|^2} = \frac{257}{329} \approx 0.781.$$
$$\mu_{4,2} = \frac{v_4 \cdot v_2^*}{\|v_2^*\|^2} = \frac{-60}{90} \approx -0.667.$$
$$\mu_{4,3} = \frac{v_4 \cdot v_3^*}{\|v_3^*\|^2} = \frac{344.674}{435.91} \approx 0.791.$$

Round $\mu_{4,1} \approx 1$, $\mu_{4,2} \approx -1$, $\mu_{4,3} \approx 1$ and size reduce.

$$v_4^* = v_4 - \lfloor \mu_{4,1} \rceil v_1^* - \lfloor \mu_{4,2} \rceil v_2^* - \lfloor \mu_{4,3} \rceil v_3^* = \begin{pmatrix} -3 \\ 1 \\ 1 \\ 5 \end{pmatrix}.$$

Check $\|v_2^*\|^2 \geq \left(\frac{3}{4} - \mu_{2,1}^2\right) \|v_1^*\|^2$:

$$90 \geq \left(0.75 - 0.933^2\right) \cdot 329 \approx 90 \geq -82.25.$$

Condition **holds**. Increment $k = 3$.

For $k = 3$:

Check $\|v_3^*\|^2 \geq \left(\frac{3}{4} - \mu_{3,2}^2\right) \|v_2^*\|^2$:

$$435.91 \geq \left(0.75 - 0.644^2\right) \cdot 90 \approx 435.91 \geq 30.15.$$

Condition **holds**. Increment $k = 4$.

Check $\|v_4^*\|^2 \geq \left(\frac{3}{4} - \mu_{4,3}^2\right) \|v_3^*\|^2$:

$$33.963 \geq \left(0.75 - 0.791^2\right) \cdot 435.91 \approx 33.963 \geq 54.05.$$

Condition **fails**.

Swap v_3 and v_4, set $k = max(k - 1, 2) = 3$.

$$\vdots$$

LLL Reduced Matrix:

$$= \begin{pmatrix} 4 & 3 & -10 & -2 \\ 9 & 5 & 0 & -4 \\ 10 & -3 & -4 & 8 \\ -3 & 5 & 2 & 11 \end{pmatrix}$$

To verify that the LLL algorithm preserves the lattice, we first note that the determinant of the original basis and the LLL-reduced basis both equal 15374. While this match is a necessary condition, it alone does not suffice to prove lattice equivalence. However, the relationship $B' = BU$ holds with U as an integer matrix satisfying $|\det(U)| = 1$, ensuring $\mathcal{L}(B') = \mathcal{L}(B)$, thus confirming that the lattices are identical.

Nearest Plane Algorithm

The Nearest Plane Algorithm provides an efficient solution to the Closest Vector Problem with approximation factor $\gamma = 2\left(\frac{2}{\sqrt{3}}\right)^n$ [61]. Given a target vector $t \in \mathbb{R}^m$ and a lattice basis $B = \{b_1, \ldots, b_n\}$ where $n \leq m$, the algorithm outputs a lattice

point $x \in L(B)$ that is close to t. The output vector x is often called the *Babai point* for t. The algorithm is applicable to arbitrary lattices, including those that are not full-rank, and it runs in polynomial time.

The algorithm consists of two primary phases:

1. **Basis Reduction:** The algorithm begins by applying the δ-LLL algorithm to the basis B with $\delta = \frac{3}{4}$. This step reduces the basis to be nearly orthogonal, ensuring better approximation results in the subsequent steps.
2. **Iterative Nearest Plane Computation:** The algorithm iteratively projects the target vector t onto the lattice planes defined by the reduced basis. The algorithm refines the lattice point approximation step-by-step until it outputs a lattice point close to t.

6.3 Block Korkine-Zolotarev (BKZ) Algorithm

The Block Korkine-Zolotarev (BKZ) algorithm is a lattice reduction technique that generalizes methods like Gaussian Lattice Reduction (GLR) and the Lenstra-Lenstra-Lovász (LLL) algorithm. Introduced by Schnorr and Euchner in 1994, BKZ aims to produce a basis where vectors are nearly orthogonal and the shorter vectors approximate the shortest non-zero vectors in the lattice.

Definition 73 (BKZ-Reduced Basis)**.** Let $\mathcal{L} \subset \mathbb{R}^n$ be a lattice with basis $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$. A basis is BKZ-reduced with block size β if it satisfies the Lovász condition (from LLL) across the entire basis and ensures that, within each local block of β consecutive vectors, the projected sublattice contains a vector close to its shortest non-zero vector, typically found by solving the Shortest Vector Problem (SVP) in that block.

BKZ operates by iteratively refining the basis through local reductions applied to overlapping blocks of size β. It combines size reduction with SVP-solving techniques to improve the global quality of the basis.

Algorithm 6 Block Korkine-Zolotarev (BKZ) Reduction

loop
  for $k = 1$ to $n - 1$ **do**
    Perform size reduction on $\mathbf{b}_{k+1}$ with respect to $\mathbf{b}_1, \ldots, \mathbf{b}_k$
    Define block $\mathbf{B}[k : k + \beta - 1] = \{\mathbf{b}_k, \ldots, \mathbf{b}_{k+\beta-1}\}$
    if Lovász condition fails within the block **then**
      Compute the shortest vector $\mathbf{v}$ in the projected lattice of the block
      Insert $\mathbf{v}$ into the basis and adjust subsequent vectors
    end if
  end for
  if no changes were made in this iteration **then**
    return the basis $\mathbf{B}$
  end if
end loop

Remark 45. The BKZ algorithm is a generalization of the LLL algorithm incorporating specialized subroutines.

Remark 46. The BKZ algorithm terminates in a finite number of steps, as each iteration either reduces the basis quality metric (e.g., the norm of vectors) or leaves it unchanged, eventually stabilizing.

Remark 47. Larger block sizes β improve the quality of the reduced basis, with shorter and more orthogonal vectors, but increase computational complexity exponentially.

Remark 48. When $\beta = 2$, BKZ resembles LLL with pairwise reductions; when $\beta = n$, it seeks an exact SVP solution across the entire lattice, which is computationally infeasible for large n.

Remark 49. The runtime is dominated by SVP-solving within blocks, typically exponential in β, though practical implementations use moderate β (e.g., 20–50) to balance quality and efficiency.

While the LLL algorithm is widely used due to its simplicity, this very simplicity comes at a cost—the quality of LLL is not sufficient for all applications. An improvement in basis quality can be achieved using the Block Korkine-Zolotarev algorithm, though this comes with greater computational overhead [68]. When BKZ terminates, it guarantees a high-quality output basis. However, unless the block size is small, the number of subroutine calls is not only non-polynomial but has also been reported to grow exponentially [69].

Recall the example matrix used in the LLL subsection. Applying the BKZ algorithm to the same matrix yields exactly the same reduced basis.

6.4 Enumeration Algorithms

Lattice enumeration is an exhaustive search that systematically explores integer linear combinations of the basis vectors to find the shortest or closest vector on lattices. Enumeration algorithms date back to the 1980s and were introduced by **Finke and Pohst** [70] and **Kannan** [71]. Despite their exponential runtime, these algorithms are widely used in practice due to their ability to find optimal solutions in moderate dimensions.

The enumeration algorithm follows a depth-first search strategy over integer combinations of the basis vectors while pruning suboptimal branches. The general steps are:

1. **Preprocessing:** Since enumeration is sensitive to the basis quality, an initial **lattice reduction step** (e.g., LLL or BKZ) is performed to obtain a more orthogonal basis.
2. **Bounding:** A search radius R is set, which defines an upper bound on the norm of candidate vectors.
3. **Recursive Search:**

- Construct integer combinations of the basis vectors systematically.
- Maintain a partial sum of the vector being explored.
- Prune search branches where the partial sum norm exceeds R.

4. **Backtracking:** If a branch does not yield a valid candidate, backtrack to explore alternative integer coefficients.
5. **Output:** The shortest (or closest) vector found within the given bound.

Algorithm 7 Kannan's Enumeration Algorithm for SVP

Require: A reduced basis $B = \{b_1, b_2, \ldots, b_n\}$ of a lattice L
Ensure: A shortest nonzero vector v in L

```
Initialize v_best ← ∞ (store shortest vector found)
Perform LLL or BKZ reduction on B to improve enumeration efficiency
Start depth-first search from the last coordinate i = n
while searching through coefficients c_i of lattice vectors do
    Compute the partial sum v = Σ_{j=i}^{n} c_j b_j
    if ‖v‖ ≥ ‖v_best‖ then
        Backtrack to previous level
    else if i = 1 and v ≠ 0 then
        Update v_best ← v if ‖v‖ is smaller
    else
        Move to the next coordinate i ← i − 1
    end if
end while
Return v_best
```

Enumeration can be interpreted as a tree traversal, where each level corresponds to a coefficient of a basis vector. The leaves of the search tree correspond to full lattice points, and only those with a sufficiently small norm are considered as potential shortest vectors.

The worst-case time complexity of lattice enumeration is **exponential** (or worse), making it infeasible for high-dimensional lattices. However, in practice, it remains highly effective for dimensions up to 80–100, especially when combined with reduction techniques.

To improve efficiency, several optimizations have been introduced:

- **Kannan's Algorithm (1983)**: Introduced a more structured way to enumerate vectors, leading to better pruning strategies.
- **Pruned Enumeration (1990)**: Instead of performing a full exhaustive search, certain subtrees of the search space are pruned based on the probability of containing a short vector [72]. This significantly reduces the number of branches explored.

- **Extreme Pruning:** A more aggressive pruning approach that eliminates most branches early, relying on probabilistic estimates.
- **Parallelization:** Running multiple enumeration trees concurrently to explore different sections of the search space.

In practice, enumeration is **never applied directly** to a given lattice basis. Instead, it is always preceded by a **basis reduction** step, such as LLL or BKZ, to improve efficiency and reduce the number of candidates to explore. While enumeration is not feasible for very high dimensions, it remains one of the most effective algorithms for solving lattice problems in dimensions relevant to cryptanalysis and computational number theory. Despite its exponential runtime, enumeration remains a key technique in lattice-based problem solving, particularly when combined with advanced pruning and reduction techniques.

6.5 Sieving

Lattice sieve algorithms are a class of randomized exponential time algorithms designed to solve lattice problems exactly or approximately. These algorithms significantly improve the asymptotic complexity of traditional enumeration methods, reducing the running time from $n^{O(n)}$ to a single exponential complexity of $2^{O(n)}$. However, this advantage comes at the cost of using randomization and requiring exponential space. The first sieve algorithm for lattice problems was introduced by Ajtai, Kumar, and Sivakumar [73], which provided a solution to the Shortest Vector Problem (SVP) in the Euclidean norm within a single exponential time bound of $2^{O(n)}$.

Algorithm 8 Sieving Algorithm for Lattice Problems

Require: A lattice basis B and an integer N (size of the sieve list)
Ensure: A short vector in the lattice
1: Generate N random lattice vectors $S = \{v_1, v_2, \ldots, v_N\}$
2: **for** each pair (v_i, v_j) in S with $i \neq j$ **do**
3: **if** $\|v_i - v_j\|$ is small **then**
4: Replace v_i with $v_i - v_j$
5: Replace v_j with $v_j - v_i$
6: **end if**
7: **end for**
8: Return the shortest vector found in S

Over time, improvements have been made to reduce the constant in the exponent:

- Micciancio and Voulgaris [57] improved the efficiency of sieve algorithms using improved search strategies.
- Hanrot, Pujol, and Stehlé [74] further optimized the algorithm for both SVP and Closest Vector Problem (CVP).

- Aggarwal et al. [75] introduced a sieve algorithm using discrete Gaussian sampling, achieving an exponent $c = 1$, which remains the asymptotically fastest provable sieve algorithm.

While these provable algorithms achieve theoretical improvements, their high exponential space complexity and inefficiencies due to random perturbations make them less practical compared to heuristic variants.

Heuristic Sieve Algorithms

Heuristic variants of lattice sieve algorithms remove the need for random perturbations, significantly improving their practical performance at the cost of losing provable guarantees. The earliest heuristic sieve algorithm was proposed by Nguyen and Vidick [76], demonstrating practical feasibility for moderate lattice dimensions. The Gauss Sieve algorithm provided further optimizations, becoming the basis of many experimental studies. Several improvements have been introduced to enhance sieving efficiency. Wang et al. [77] refined the Nguyen-Vidick sieve, leading to better performance. Zhang, Pan, and Hu [78] proposed a three-level sieve strategy to further optimize the process. Becker, Gama, and Joux [79] introduced overlattices, which contributed to improved sieving efficiency. Additionally, Laarhoven [80, 81] employed locality-sensitive hashing, achieving a heuristic complexity of $2^{0.297n}$, making it the fastest known heuristic sieve algorithm.

7 Discussion

Quantum computers threaten currently employed cryptosystems like RSA and ECDH. Shor's algorithm efficiently breaks their security foundations, much to cryptographers' concern. While large-scale quantum computers are not yet here, their arrival appears increasingly imminent. Once a decades-off worry [82], the estimated timeline has shrunk dramatically to around five years, perhaps even sooner with Majorana fermions [83] [84].

A fault-tolerant quantum computer presents an existential threat to traditional public-key cryptography, a potential "*cryptocalypse*". Current estimates suggest breaking 2048-bit RSA requires 1730 qubits [85], and 256-bit ECC 2330 [86]; physical qubit counts would be considerably higher.

The "store-now-decrypt-later" threat is a major concern for RSA/ECC. Importantly, quantum computing does not offer general polynomial-time solutions for NP-hard problems, and quantum algorithms provide insufficient speedups to break lattice-based cryptography.

Lattice cryptography is promising for post-quantum security due to its efficiency, versatility, and security. A leading candidate for adoption, it forms the basis for three of four NIST-standardized post-quantum schemes: Kyber, Dilithium, and Falcon.

The future of quantum computing remains uncertain—whether it will scale to break modern cryptographic schemes or remain an unfulfilled promise. Likewise, the presumed hardness of lattice-based cryptography holds until history repeats itself with another breakthrough, much like Shor's algorithm did for integer factorization.

It is worth recalling that *"secure until proven insecure"* is cryptography's analogue to *"innocent until proven guilty"*. Consequently, relying on lattice-based problems, current cryptographic constructions remain secure.

Fortunately for most, but unfortunately for researchers pursuing the next breakthrough, there are currently no known polynomial-time classical or quantum algorithms capable of solving hard instances of lattice problems.

Appendix A: Mathematical Preliminaries

We begin with an exploration of divisibility and number theory, delving into the properties and relationships of integers. Next, we introduce one-way functions, crucial for cryptography and complexity theory. Following this, we cover the fundamentals of linear algebra and vector spaces, which form the backbone of numerous mathematical applications. Finally, we investigate abstract algebra, examining the structure and properties of groups, rings, and fields, which are pivotal for understanding advanced algebraic systems.

Definition 74. (Divisibility) Given two numbers $a, b, k \in \mathbb{Z}$ and $a \neq 0$. We say that **a divides b** if b is a product of multiplication of a and k. If $b = k \cdot a$ holds for some integer k, we denote it $a|b$.

Example 17. Let $a = 7$ and $b = 21$, $a|b$ since $21 = 3 \cdot 7$.

Definition 75. An integer $p \geq 2$ is a prime number also known as a prime if its only positive divisors are 1 and number p itself. $\mathbb{P}$ denotes the set of all primes.

Theorem 15. *Every integer $n > 1$ can be represented as a product of prime powers, that is, in the form $p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$ where each $n_i \geq 0$ and all p_i are unique primes. Furthermore the factorization is also unique where its order of variables disregarded.*

Deciding whether given a number is a prime or not holds a position of significant importance due to its direct implications in various fields including, but not limited to, cryptography, computer science, and information security. It is often the seemingly simplest concepts that hold the greatest complexity and challenge, underscoring the importance of rigorous exploration and examination of even the most basic notions. Fermat's Last Theorem which was proved after nearly four centuries serves as an illustrative example of this principle [87].

Theorem 16. *If integer n is not a prime number, i.e. composite number $n \notin \mathbb{P}$, then n has a prime factor $p \leq \sqrt{n}$*

Proof. Let n be a composite number. Then there exists, $a, b \in \mathbb{Z}$ such that $n = a \cdot b$.

$$1 \leq a, b \leq n$$

$$a \leq \sqrt{n} \text{ or } b \leq \sqrt{n} \text{ since } a \cdot b > n$$

would be contradiction. We can assume, without loss of the generality of the argument, that

$$a \leq \sqrt{n} \implies p|n, \quad p \leq \sqrt{n}$$

□

Definition 76. (Greatest Common Divisor (GCD)) The gcd of two non-zero integers a and b, denoted as $\gcd(a, b)$, is the largest positive integer that is capable of dividing both a and b without leaving a remainder. More formally, $\gcd(a, b)$ is non-negative integer d satisfying the following two properties: d divides both a and b, i.e., there exist integers x and y such that $a = dx$ and $b = dy$. If d' is any other positive integer that divides both a and b, such that $d' \neq d$ then $d' \leq d$.

Theorem 17. *If there exist a greatest common divisor $d = gcd(a, b)$, there exist $u, v \in \mathbb{Z}$ where $d = ua + vb$ non-unique equation holds and we call this expression as* ***Bezout's identity*** *and u, v denoted as Bezout coefficients.*

Definition 77. Two non-zero integers a and b as being coprime or relative prime or mutually prime when their Greatest Common Divisor (GCD) is 1, denoted $\gcd(a, b) = 1$

Remark 50. Two distinct primes p, q are always relative primes.

Definition 78. The **Euclidean Algorithm** is a method and procedure for determining the greatest common divisor (gcd) of two non-zero integers a and b, based on the principle of repeated division. Specifically, to compute $gcd(a, b)$, the algorithm proceeds through the following steps:

The algorithm commences with the initial values of a and b and operates iteratively as follows:

$$\begin{aligned} a &= q_1 b + r_1, \\ b &= q_2 r_1 + r_2, \\ r_1 &= q_3 r_2 + r_3, \\ &\vdots \end{aligned}$$

$$r_{n-1} = q_{n+1} r_n,$$

The algorithm continues until a remainder of zero is produced and then halts. The final nonzero remainder r_n is then, $\gcd(a, b) = r_n$. In the forthcoming algorithmic representantion, the symbol '%' denotes the remainder from division when one number is divided by another. Euclid explains his way oy thinking on this manner with the following sentences in his book *Elements, Book VII.*

> *Two unequal numbers being set out, and the less begin continually subtracted in turn from the greater, if the number which is left never measures the one before it until an unit is left, the original numbers will be prime to one another.*

[88]

Algorithm 9 Euclidean Algorithm

```
1: procedure GCD(a, b)                          ▷ Input: a, b ∈ ℤ
2:     r ← a % b                                ▷ Remainder r = a − q_x b
3:     while r ≠ 0 do
4:         a ← b                                ▷ Assign the value of b to a
5:         b ← r                                ▷ Assign the value of the remainder r to b
6:         r ← a % b                            ▷ Recalculate the remainder
7:     end while
8:     return b
9: end procedure
```

Example 18. Calculate the GCD of 370 and 192 using the division method:

$$\begin{aligned}
370 &= 1 \cdot 192 + 178 \quad r_1 = 178, b = 192 \\
192 &= 1 \cdot 178 + 14 \quad r_2 = 14 \\
178 &= 12 \cdot 14 + 10 \quad r_3 = 10 \\
14 &= 1 \cdot 10 + 4 \quad r_4 = 4 \\
10 &= 2 \cdot 4 + 2 \quad r_5 = 2 \\
4 &= 2 \cdot 2 + 0 \quad r_6 = 0
\end{aligned}$$

Since r_5 is the last non-zero remainder, $gcd(370, 192) = 2$. Following is the calculations for finding the coefficients of Bezouts identity such that $au + bv = gcd(a, b)$

$$\begin{aligned}
2 &= 10 - 2 \cdot 4 \\
&= 10 - 2 \cdot (14 - 1 \cdot 10) = -2 \cdot 14 + 3 \cdot 10 \\
&= -2 \cdot 14 + 3 \cdot (178 - 12 \cdot 14) = 3 \cdot 178 - 38 \cdot 14 \\
&= 3 \cdot 178 - 38 \cdot (192 - 1 \cdot 178) = -38 \cdot 192 + 41 \cdot 178 \\
&= -38 \cdot 192 + 41 \cdot (370 - 1 \cdot 192) = 41 \cdot 370 - 79 \cdot 192 \\
&\Rightarrow 370 \cdot 41 + 192 \cdot (-79) = 2
\end{aligned}$$

Therefore, the Bézout coefficients for $370 \cdot u + 192 \cdot v = \gcd(370, 192)$ are $u = 41, \quad v = -79$.

Remark 51. Bezout's identity is a natural by-product of the Euclidean algorithm.

Definition 79. (Modulus) Let n be a positive integer, and let a and b be integers. We say that a and b are **congruent modulo** m if their difference is divisible by m without remainders. To indicate this congruence, we write

$$a \equiv b \pmod{m}$$

Definition 80. (Multiplicative Inverse) Let a be an integer, the existence of its multiplicative inverse in $\mathbb{Z}_n$ (the integers modulo n) is precisely determined by the property that a and n are relative primes, i.e., their greatest common divisor (gcd) is

equal to 1. The multiplicative inverse of a is an integer x that satisfies the congruence relation:

$$a \cdot x \equiv 1 \pmod{n}$$

In other words, it is an integer x for which there exists an integer k such that:

$$a \cdot x = 1 + k \cdot n$$

By rearranging the equation, we can express it as:

$$a \cdot x - k \cdot n = 1$$

We will denote the value of x as $a^{-1} \pmod{p}$ for the multiplicative inverse of a.

Remark 52. Euclidean algorithm and the Bezouts identity together provides an effective way to calculate multiplicative inverse.

Remark 53. By applying the principles of the Euclidean algorithm in a sequence with Bezout's identity, the resulting overall process is commonly known as the Extended Euclidean algorithm.

Definition 81. The Euler Totient function, denoted by $\phi(n)$, is defined as

$$\phi(n) = |\{x : 1 \leq x < n \text{ and } \gcd(x, n) = 1\}|,$$

which represents the cardinality of the set of integers less than n that are coprime to n.

If p is a prime number and $\alpha \geq 1$, then

$$\phi(p^{\alpha}) = p^{\alpha-1}(p-1).$$

Furthermore, if m and n are coprime, that is, $\gcd(m, n) = 1$, then

$$\phi(mn) = \phi(m)\phi(n).$$

Given these principles, the Euler Totient function can be computed for any integer n. In a general sense, we express:

$$\begin{aligned}\phi(n) &= \phi\left(\prod_{i=1}^{k} p_i^{\alpha_i}\right) \\ &= \prod_{i=1}^{k} \phi\left(p_i^{\alpha_i}\right) \\ &= \prod_{i=1}^{k} p_i^{\alpha_i - 1}(p_i - 1)\end{aligned}$$

This means that $\phi(n)$ of any integer n can be evaluated as the product of ϕ of its prime power factors, where each prime power factor $p_i^{\alpha_i}$ contributes $p_i^{\alpha_i - 1}(p_i - 1)$ to the product.

Remark 54. The number of elements in $\mathbb{Z}_m$ equals to $\Phi(m)$.

Definition 82. (Quadratic residues) Quadratic residues modulo p are the numbers that can be expressed as squares of integers modulo p.

Consider the congruence $a^2 \equiv b^2 \pmod{p}$. This implies that p divides the difference $a^2 - b^2$. Since $a^2 - b^2$ can be factored as $(a-b)(a+b)$, we have $p \mid (a-b)(a+b)$. Given that p is a prime number, it must divide one of the factors. Therefore, $p \mid (a-b)$ or $p \mid (a+b)$. As a result, $a \equiv \pm b \pmod{p}$.

Theorem 18. *(Fermat's Little Theorem) Let α be an integer and p be a prime, then: $\alpha^p \equiv \alpha \pmod{p}$.*

Definition 83. Given that $p \nmid \alpha$, Fermat's Little Theorem may also be stated in following form:

$$\alpha^{p-1} \equiv 1 \pmod{p}$$

Proof. Let p be a prime number, and consider the set of integers $S = \{1, 2, 3, \ldots, p-1\}$. We want to show that the product of all the elements in S is congruent to 1 modulo p, i.e.,

$$\prod_{a \in S} a \equiv 1 \mod p$$

). Now, since p is a prime, all elements in S are coprime to p. For any element a in S, we can find a unique integer a' such that $a \cdot a' \equiv 1 \mod p$. This is because each element in S has an inverse modulo p due to the fact that p is prime. Now, we can pair each element a in S with its inverse a', and the product of each pair is congruent to 1 modulo p, i.e., $a \cdot a' \equiv 1 \mod p$ for each pair. Since the number of elements in S is even (there are $p-1$ elements, and p is odd), we can form $\frac{p-1}{2}$ pairs of elements in S, and the product of all these pairs can be represented as:

$$\prod_{a \in S} a = (a_1 \cdot a_1') \cdot (a_2 \cdot a_2') \cdots (a_{\frac{p-1}{2}} \cdot a_{\frac{p-1}{2}}') \equiv 1 \cdot 1 \cdots 1 \equiv 1 \mod p$$

Thus, we have shown that $\prod_{a \in S} a \equiv 1 \mod p$. □

Theorem 19. *(Euler's Theorem) For integers a and m with $\gcd(a, m) = 1$, we have:*

$$a^{\phi(m)} \equiv 1 \pmod{m}.$$

One-way Functions

After defining the complexity classes **P** and **NP**, now we may talk about the *intractability*. Most of the cryptographic systems that are already implemented are relying on the existence of the one-way functions. In simplest terms, one way functions are representing the family of functions that considered *easy* to compute and *hard* to find the inverse of function.

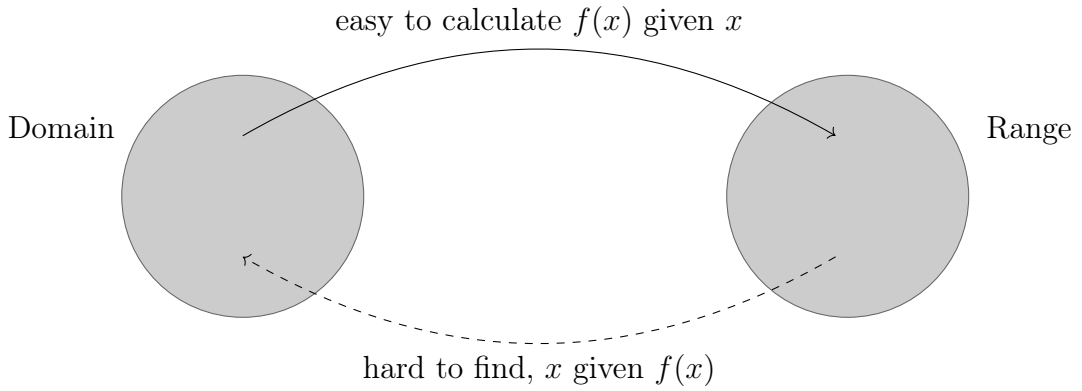

Figure 14: Illustration of One-Way Functions

Preceding our exploration of the concept of one-way functions, it is of critical importance to underline the fact that the existence of such functions has yet to be proven. Furthermore, if one-way functions exists. As **P** versus **NP** problem is still preserving its significance as remaining unsolved more than 50 years [89], since Stephen Cook introduced the idea, a solution to this major problem will cause a paradigm shift in cryptography and various other fields. In order to introduce the concept of a one-way function in a formal proper way, it is necessary to first define what a **negligible function** entails. Hence, the following formal definitions have been arranged:

Definition 84. (Negligible Function) A function $f : \mathbb{N} \to \mathbb{R}$ is a negligible function if and only if for every natural number constant, i.e., $c \geq 0$, there exists an integer k_c such that $|f(k)| < k^{-c}$ whenever $k > k_c$.

Example 19. Show that the function $f(k) = \frac{1}{(-4)^k}$ is a negligible function.

Consider a natural number $c \in \mathbb{N}$ and let's analyze the function $f(k) = \frac{1}{(-4)^k}$. The correct expression for this function in terms of its absolute value is:

$$|f(k)| = \left| \frac{1}{(-4)^k} \right| = \frac{1}{4^k}$$

To show that $f(k)$ is negligible, we need to demonstrate that for any $c \geq 0$, there exists k_c such that:

$$|f(k)| = \frac{1}{4^k} < k^{-c} \quad \text{for all} \quad k > k_c$$

Given that $\frac{1}{4^k}$ decreases exponentially and k^{-c} decreases polynomialy, the exponential decay of $\frac{1}{4^k}$ ensures that it will eventually be smaller than k^{-c} for sufficiently large k. Therefore, we can find a k_c for each c where this condition is met, affirming that $f(k) = \frac{1}{(-4)^k}$ is indeed a negligible function.

Negligible functions and the idea of **negligibility** is really important for the field of cryptography and cryptographers since proofs of the security of the system is also

required. In general this notion helps when cryptographers are in need of quantifying a security of a crypto-system. In common parlance, considering a new candidate algorithm or a system, if the probability of successful attack is extremely low or gathering the resources or the computational expenses are impractical then this probability might be neglected.

Definition 85. (Probabilistic Algorithm) Given that M is a Turing machine with extra **randomness tape**, A n algorithm A_p is considered probabilistic if it is capable of making random decisions based on a randomness generator and a probability distribution that changes transition functions randomly at each step during the computation of given problem.

Definition 86. (One-way Functions) Let A_p be a probabilistic algorithm that runs in polynomial time and f be a function defined as follows: $f : \{0,1\}^* \rightarrow \{0,1\}^*$. Function f is **said to be** a one-way function if the following two conditions are satisfied:
1. f should be evaluated easily, in polynomial time.

$$\Pr[A_p(x) = f(x)] = 1.$$

2. For every A_p, there exists a negligible function $neg(n)$ such that for all sufficiently large values of n when the input is uniformly distributed:

$$\Pr[A_p(f(x)) = x] \leq neg(n)$$

Existence of the one-way functions are simply conditional to the pre-assumption of $\mathbf{P} \neq \mathbf{NP}$. If one-way functions exists, then one-way permutations exists and it directly implies the *existence*[16] of secret-key agreement, proven by *Impagliazzo et al.* [91] in 1989.

Definition 87. (Trapdoor Functions) A trapdoor function can be defined as a function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ if it is also a one-way function an it satisfies the following conditions. Given that a polynomial p and a probabilistic polynomial time (PPT) algorithm A, such that for every K, there is a corresponding private secret key or trapdoor key $K_{trap} \in \{0,1\}^*$ with a length $|K_{trap}|$ that is less than or equal to $p(K)$. For all $x \in \{0,1\}^*$, the algorithm A applied to $f(x)$ as $A(f(x), K_{trap}$ with the trapdoor key K_{trap} yields a y such that $f(y) = f(x)$.

We define and utilize **trapdoor functions** by virtue of one-way functions. The key distinction between one-way function and a trapdoor function lies in preserving the 'hardness' of inverse computation for one-way functions while enabling the computation of the function's inverse with the aid of a secret key, known as *trapdoor*, in trapdoor functions. For instance, in the RSA cryptosystem, the capability to compute the y-th power of an unknown x modulo n is recognized as one of the most notable candidates for trapdoor functions.

Theorem 20. *Assume that f is a stable one-way function. Then the idea and implementation of computationally secure cryptosystems is valid [92].*

[16] I kindly recommend this thought-provoking research paper that Impaliazzo's *Five Worlds* for better understanding on relationship between complexity classes and one-way functions [90]

Proof. Assume f is a stable one-way function, meaning it's easy to compute $y = f(x)$ for any x, but computationally infeasible to find x given $y = f(x)$, barring some additional information. Let us construct a cryptosystem using this function f. Here, a message P is encrypted with a key k to yield ciphertext $C = f(P, k)$. The key k serves as the 'trapdoor' allowing feasibly computing $f^{-1}(C)$. For anyone lacking k, finding P from C is computationally hard due to f's stability. Conversely, a legitimate recipient with k can feasibly decrypt C to P. Thus, assuming a stable one-way function, we have validated the feasibility of creating a secure cryptosystem, resistant against adversaries. This confirms the viability of employing stable one-way functions for cryptosystem construction. □

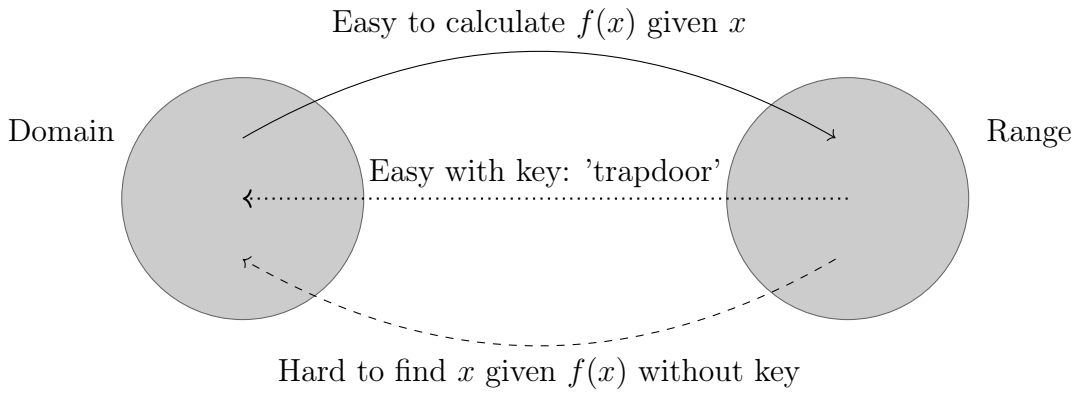

Figure 15: Illustration of Trapdoor Functions

After these definitions, we underscore the significance of one-way functions and trapdoor functions in forming secure and reliable cryptographic systems, due to fact that if these two elements were proven not to exist, it would imply the downfall of a majority of the systems currently deemed "secure". One-way functions, distinguished by their computational simplicity in one direction and the computational impracticality of reversing the operation, play indispensable roles in crafting cryptographic hash functions, securing password systems, and formulating digital signature protocols. Considering the anarchic, decentralized systems of the present day, notably cryptocurrencies, we observe that a collapse of the architecture would ensue if one-way functions were proven non-existent. This structure, proposed by Nakamoto and now widely acknowledged and utilized across various sectors, would be fundamentally undermined since all the transactions and timestamps are based on *hash* functions [93]. In a parallel manner, trapdoor functions, a unique class of one-way functions, enhance the security structure of public-key cryptography. The mutual relationship between these two types of functions is essential in upholding the cardinal principles of digital security: confidentiality, integrity, and availability. As we further advance into the information age, acquiring a comprehensive understanding of these functions and their broad applications, which we will discuss later, becomes of paramount importance to both cybersecurity specialists and theoretical mathematicians alike.

Abstract Algebra

Definition 88. (Group) A *group* is a collection of elements denoted by the set G, which possesses an operation $*$ that ensures closure

1. **Closure** For all elements $a, b \in G$, the result of the operation $a * b$ belongs to G, i.e., $a * b \in G$.
2. **Identity** There exists an element e, neutral element or identity, in G such that for every element x in G, the operation $e * x$ and $x * e$ result in x.
3. **Inverse** For each element x in G, there exists an element y in G such that the operation $x * y$ and $y * x$ both yield the identity element e.
4. **Associativity** The group operation satisfies the associative property for all elements x, y, and z in G, i.e., $x * (y * z) = (x * y) * z$.

We may also say that G is a group under the operation $*$. Furthermore, if G also satisfies the property of **commutativity**, for all elements $a, b \in G$, the operation $a * b$ is equal to $b * a$, then group G is **abelian** or **commutative**.

Example 20. The set of integers $\mathbb{Z}$ with addition $(+, \mathbb{Z})$ is a group.

When a group G is finite, it can be represented using a group table. The elements of G are listed down the first column and across the top row of the table. The cell at the intersection of row a and column b contains the product $a * b$. For example, the group table for Z_5 is shown in the specified table.

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Table 6: Group table for $\mathbb{Z}_5$

The set $\mathbb{Z}/n\mathbb{Z}$, consisting of integers modulo n under addition, satisfies the group axioms as follows. Closure is satisfied since the sum of any two integers modulo n remains an integer modulo n by definition. If $a, b \in \mathbb{Z}/n\mathbb{Z}$, then $(a + b) \bmod n$ is also in $\mathbb{Z}/n\mathbb{Z}$. Here, the binary operation $*$ corresponds to addition modulo n. The identity element is 0 modulo n, as for any $a \in \mathbb{Z}/n\mathbb{Z}$, the operation $a * 0$ corresponds to $a + 0 \equiv a \pmod{n}$. Thus, 0 acts as the identity element in this structure. Each

element $a \in \mathbb{Z}/n\mathbb{Z}$ possesses an inverse element b such that $a * b = 0$, which implies $a+b \equiv 0 \pmod{n}$. The additive inverse of a modulo n is $n-a$, satisfying the inverse property. Associativity holds because addition of integers is inherently associative. That is, for any $a, b, c \in \mathbb{Z}/n\mathbb{Z}$, the equality

$$a + (b + c) = (a + b) + c$$

implies

$$a + (b + c) \equiv (a + b) + c \pmod{n}.$$

Thus, the operation $*$, defined as addition modulo n, is associative in $\mathbb{Z}/n\mathbb{Z}$.

Theorem 21. *Let G be a group. The identity element in G is unique, and for any $a \in G$, the inverse a^{-1} is also unique.*

Proof. Suppose e and f are both identity elements in G. Since f is an identity element, we have $ef = e$. Similarly, since e is an identity element, $ef = f$. Hence, $e = f$. □

Definition 89. Let $\mathbb{Z}_n$ be a group under the operation of multiplication. The **order** of this group, denoted by $\phi(n)$, is defined as Euler's phi-function.

Definition 90. (Cyclic Group) A group G is called cyclic if there exists an element $a \in G$ such that every element of G can be expressed as a power of a.

Consider the set of integers $\mathbb{Z}$ under addition, $(\mathbb{Z}, +)$. This set forms a group. It is cyclic because all elements can be expressed as $n \times 1$ or $n \times (-1)$ for some integer n. Here, the integer 1 serves as a generator of the group, as every other integer can be obtained by repeatedly adding 1 or its inverse (-1). Thus, $(\mathbb{Z}, +)$ is an example of an infinite cyclic group.

Definition 91. (Subgroup) Let G represent a group. A subset H within G is designated as a subgroup of G if it fulfills the following conditions: First, H exhibits closure with respect to the group operation, implying that for any elements a and b belonging to H, their product $a \cdot b$ remains an element of H. Second, the identity element of the group, denoted as 1, is an element of H. Finally, for every element a in H, its corresponding inverse element a^{-1} also belongs to H.

Every group is considered a subgroup of itself, and the set e, consisting solely of the identity element, is a subgroup of any group. When we mention a proper subgroup of G, we are referring to any subgroup that is not equal to G itself. The set of even integers is a subgroup of $(\mathbb{Z}, +)$.

Definition 92. (Homomorphism) Let $(G, *)$ and $(G', \cdot)$ be groups. A mapping $f : G \to G'$ is considered a (group) homomorphism if it adheres to the following condition: For all elements $a, b \in G$, the homomorphism f preserves the group operation, meaning that $f(a * b) = f(a) \cdot f(b)$.

Definition 93. (Isomorphism) Let G and H be groups. A group isomorphism (or simply, an isomorphism) from G to H is a bijective homomorphism from G to H.

Definition 94 (Ring)**.** A *ring* is a set R equipped with two binary operations, addition $+$ and multiplication $\times$, such that: $(R, +)$ is an abelian group, multiplication is associative, multiplication is distributive over addition on both sides.

Example 21. The set of integers $\mathbb{Z}$ with the usual addition and multiplication is a ring.

Definition 95. (Subring) Let $(R, +, \cdot)$ be a ring. A subset $S \subseteq R$ is a subring of R if $(S, +, \cdot)$ is a ring and $1_S = 1_R$.

Example 22. Consider the ring of integers $\mathbb{Z}$. The set of even integers forms a subring of $\mathbb{Z}$.

Definition 96. (Ideal) Consider a ring R. A subset $I \subseteq R$ is an ideal in R if the following conditions are met:

1. The set $(I, +)$ forms a subgroup of the additive group $(R, +)$.
2. For any element $r \in R$ and $a \in I$, ra belongs to I ($rI \subseteq I$).
3. For any element $r \in R$ and $a \in I$, ar belongs to I ($Ir \subseteq I$).

Definition 97 (Module)**.** Let R be a ring. A *left R-module* is a set M equipped with an addition $+$ making $(M, +)$ an abelian group, along with an operation $R \times M \to M$, denoted by $(r, m) \mapsto rm$, satisfying for all $r, s \in R$ and $m, n \in M$:

$$r(m+n) = rm + rn, \quad (r+s)m = rm + sm, \quad (rs)m = r(sm), \quad 1_R m = m,$$

where 1_R is the multiplicative identity in R (if it exists). A *right R-module* is defined similarly with scalar multiplication on the right.

Definition 98 (Field)**.** A *field* is a set F equipped with two binary operations, addition $+$ and multiplication $\cdot$, such that $(F, +)$ is an abelian group, $(F \setminus \{0\}, \cdot)$ is an abelian group, and multiplication is distributive over addition; that is, for all $a, b, c \in F$,

$$a(b+c) = ab + ac \quad \text{and} \quad (a+b)c = ac + bc.$$

"A field is a commutative ring with multiplicative identity element $1 \neq 0$ in which every nonzero element is a unit" [94].

Theorem 22. *The ring denoted as $\mathbb{Z}_n = \mathbb{Z}/\langle n \rangle$ becomes a field if and only if n holds the property of being a prime number.*

Proof. Let us consider the case when n is prime. Suppose $a \neq 0$ in $\mathbb{Z}_n$, where $\mathbb{Z}_n = \mathbb{Z}/\langle n \rangle$. Since n is prime, a and n are coprime because a is an element in the set $\{1, 2, \ldots, n-1\}$, and prime numbers have no positive divisors other than 1 and themselves. Therefore, $\gcd(a, n) = 1$, and a cannot be divisible by n. As a and n are coprime, there exists an integer $x \in \mathbb{Z}$ such that $ax \equiv 1 \pmod{n}$. In other words, $ax = kn + 1$ for some integer k. This implies that $ax - kn = 1$, showing that a has an inverse in $\mathbb{Z}_n$. Therefore, $\mathbb{Z}_n$ is a field when n is prime. Conversely, let us assume that n is not a prime number. Thus, it can be factored as $n = ab$, where a and b are positive integers less than n. Consequently, $a \neq 1$ and $b \neq 1$, and

since a and b are both nonzero elements in $\mathbb{Z}_n$, we have $ab = 0$ in $\mathbb{Z}_n$. Since a and b are not equal to 1, they do not have multiplicative inverses in $\mathbb{Z}_n$, meaning there are no integers x and y such that $ax \equiv 1 \pmod{n}$ and $by \equiv 1 \pmod{n}$. Therefore, $\mathbb{Z}_n$ is not a field when n is not a prime number. In conclusion, the quotient ring $\mathbb{Z}_n = \mathbb{Z}/\langle n \rangle$ is a field if and only if n is a prime number. $\square$

Appendix B: Quantum Computing

As previously mentioned, the inception of the classical computer derived from a predominantly theoretical concept known as the Turing Machine. Every passing day, we manage to render technologies and technological advancements obsolete within a few years. However, upon closer examination, this phenomenon assumes a profound and concerning aspect. Considering the significant rise in computational power, it is evident that we have yet to fully comprehend the extent of this transformative development. Apollo Guidance Computer were able to handle roughly 14500FLOPS, where its computational power is approximately 30 million times less compared to that of the computer in 2021 [95]. In 1965, Moore conducted an empirical observation and subsequently formulated a prediction, positing that the density of transistors on a chip would approximately double every one and half years. As the dimensions of transistors approach the atomic scale, quantum effects, which pertain to the behavior of matter at the smallest level, will progressively govern their operation. Considering this situation as evidence of the potential integration of quantum computers into our lives, understanding quantum computers becomes an imperative rather than a mere research endeavor.

A bit of history

In quantum world, particles exhibit wavelike attributes, and their behavior is described by a specific wave equation known as the Schrödinger equation. And not surprisingly, it is behaving a little bit different than compared to the classical physics wave equations. Before discussing it further, let us follow a chronological timeline. In 1900, Max Planck put forward a revolutionary hypothesis, stating that light emissions with a specific frequency ν are not continuous but rather quantized. This meant the energy was distributed in discrete packets, with each packet being an integral multiple of a fundamental value. The energy of these packets can be represented as:

$$E = h\nu = \hbar\omega$$

where: E is the energy of the light, h is Planck's constant, ν is the frequency of the light, $\hbar$ (h-bar) is the reduced Planck's constant, and ω is the angular frequency. [96] This constant is utilized in quantum mechanics for the quantum of action. $h = 6.62607015 \times 10^{-34}\,\mathrm{J}\cdot\mathrm{s}$. Planck's quantum hypothesis introduces a somewhat unsettling notion. Throughout the eras of Galileo and Newton, the scientific community has engaged in a continuous debate regarding whether light should be perceived as a particle flow or as waves while later on Albert Einstein explained the **photoelectric effect** and introduced the idea and notion of **photon** [97]. In 1913, Niels Bohr proposed the idea that electrons within atoms exhibit wavelike characteristics. This concept provided a suitable explanation for certain phenomena observed in hydrogen atoms, particularly the existence of quantized energy levels that had been previously documented [98]. In 1924, Louis de Broglie, a notable French physicist, postulated the equivalence of relationships between particles and photons [99]. In 1925, Werner Heisenberg developed the matrix mechanics formulation, where it is equivalent of wave formulation of Schrödinger. Nearly half a century

later, Richard Feynman started to discuss how to simulate physics with computers by explaining why quantum computers cannot be simulated by ordinary computers and [100] and later on he introduced **Quantum Mechanical Computers** by highlighting the need for computers that obeys reversibility of quantum mechanics [101]. Subsequently, Deutsch was the first to recognize the insufficiency of the computational models presented by Benioff [102], demonstrating it with his own model, and introducing the concept of the universal quantum computer [103]. In the following years, first algorithms were pioneered by Deutsch and Jozsa [104] and further advanced by Simon [105]. Quantum complexity theory were introduced by Bernstein and Vazirani. Bernstein and Vazirani have achieved remarkable progress through the introduction of quantum complexity theory, thereby establishing a robust theoretical foundation for comprehending the computational capabilities and limitations of quantum computers [10]. Shor furthered Simon's research [106] by introducing his own groundbreaking algorithm for solving integer factorization and discrete logarithm problems [11].

Quantum Information

Understanding classical information is of vital importance for comprehending quantum information and quantum computation. Therefore, we will briefly discuss some prerequisite concepts herein. Take into account a physical system, denoted X, capable of information storing.

Suppose that, at any given instant, X can exist in one among a finite collection of classical states. We denote this set of states as Σ. For instance, if 'X' represents a binary bit, the corresponding set Σ would be $\{0, 1\}$. On the other hand, if 'X' signifies a four-sided die, the corresponding set Σ would be $\{1, 2, 3, 4\}$. If the probability distribution among these states is not equal, column vectors are utilized for the representation of probabilistic states of X.

Assume $\Pr(x = 0) = \frac{1}{10}$ and $\Pr(x = 1) = \frac{9}{10}$. In classical information theory, we represent this as a column vector:

$$\begin{pmatrix} \frac{1}{10} \\ \\ \frac{9}{10} \end{pmatrix}$$

where the probability of $x = 0$ is placed at the top of the vector and the probability of $x = 1$ is placed at the bottom. On probabilistic systems, we are generally using a similar method for representing probabilistic states with the probability vectors. To be precise, we can represent any probabilistic state through a column vector $\mathbf{p}$ satisfying two properties:

1. All entries of the vector are non-negative real numbers:

$$p_i \geq 0 \quad \forall i$$

2. The sum of the entries is equal to 1:

$$\sum_i p_i = 1$$

This same vectors corresponding to classical bits could be represented as: $|0\rangle = \binom{1}{0}$ and $|1\rangle = \binom{0}{1}$ and in general we may use following notation to represent same probability.

$$\frac{1}{10}|0\rangle + \frac{9}{10}|1\rangle$$

Let Σ be a finite set of symbols. For each $a \in \Sigma$, we define $|a\rangle$ as the column vector with a 1 in the position corresponding to a and 0 in all other positions. This representation is commonly referred to as *ket* a. For the row vectors on the other hand, we are using $\langle a|$ and read as *bra*.

Remark 55. In vector operations, it is straightforward to see that

$$\langle a| \, |b\rangle = \langle a|b\rangle$$

corresponds to the inner product of vectors, which is equivalent to multiplying a row vector (the conjugate transpose of $|a\rangle$) with a column vector $|b\rangle$.

Example 1:

If $\Sigma = \{0, 1\}$, then

$$|0\rangle = \begin{pmatrix} 1 \\ \\ 0 \end{pmatrix} \quad \text{and} \quad |1\rangle = \begin{pmatrix} 0 \\ \\ 1 \end{pmatrix}.$$

Remark 56. For each $a \in \Sigma$, we represent the column vector $|a\rangle$ as having a 1 in the position corresponding to a, with all other entries being 0. These vectors are known as **standard basis vectors**.

Remark 57. This is merely an alternate way for representing vectors.

This approach that we have employed in quantum physics for representing quantum states and their transformations is *Dirac's bra-ket* notation.

Elements of Quantum Computing

Utilizing quantum phenomena could potentially enable the execution of computational tasks which surpass the limitations inherent to the classical computational theory. Consequently, it is advisable to commence explorations in this domain, rather than postponing this task until the malfunction of transistors.

Definition 99. (Standart Basis) Consider the 2-dimensional complex vector space $\mathbb{C}^2$. The standard basis, also referred to as the computational basis, is given by

$B = \{|0\rangle, |1\rangle\}$. This set forms an orthonormal basis for $\mathbb{C}^2$, with the basis vectors defined as follows:

$$|0\rangle = \begin{pmatrix} 1 \\ \\ 0 \end{pmatrix} \quad \text{and} \quad |1\rangle = \begin{pmatrix} 0 \\ \\ 1 \end{pmatrix}.$$

Definition 100. (Qubit) In quantum computing, a qubit, or quantum bit, is a fundamental unit of quantum information, represented by a two-state quantum-mechanical system.

State of a single quantum bit can be repsented as a vector

$$\alpha\,|0\rangle + \beta\,|1\rangle$$

where the L_2 norm of the vector is equal to 1,

Here are two examples of quantum states of a qubit:

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle\,.$$

$$\begin{pmatrix} \frac{1+2i}{3} \\ \\ -\frac{2}{3} \end{pmatrix} = \frac{1+2i}{3}\,|0\rangle - \frac{2}{3}\,|1\rangle\,.$$

Remark 58. When considering vectors, it is natural to wonder if we can express any vector as $\alpha|0\rangle + \beta|1\rangle$. This expression accurately represents the mathematical formulation of quantum bits.

Consider a quantum state denoted as $|\psi\rangle$, represented as a linear combination of two basis states: $|0\rangle$ with coefficient α and $|1\rangle$ with coefficient β. When this qubit is subjected to a measurement in the standard basis, the probability of obtaining outcome 0 is given by $|\alpha|^2$, and the state collapses to $|\psi_0\rangle = |0\rangle$. Similarly, the probability of obtaining outcome 1 is $|\beta|^2$, leading to the state collapsing to $|\psi_0\rangle = |1\rangle$.

Definition 101. (Quantum Gate) An operation on a qubit is a a unitary mapping $U : H_2 \to H_2$ [97] is called a quantum gate.

Remark 59. States of a quantum system are typically represented by vectors, while the operations performed on these states are depicted using unitary matrices.

One can think of quantum gates as a tranformation on a single qubit with a linear operation. Given the basis states $|0\rangle$ and $|1\rangle$, the gate transforms these states as follows:

$|0\rangle \mapsto a|0\rangle + b|1\rangle$ and $|1\rangle \mapsto c|0\rangle + d|1\rangle$.

Here, a, b, c, and d are complex numbers that define how the gate transforms the basis states. The transformation can be represented by a matrix. For the above transformations, the matrix is:

$$\begin{pmatrix} a & c \\ b & d \end{pmatrix}$$

If $a = x + iy$ (where x and y are real numbers), then the complex conjugate a^* is given by $a^* = x - iy$. Similarly, $(a, b)^T$ used for indicating the transposition where

$$(a, b)^T = \begin{pmatrix} a \\ b \end{pmatrix}$$

As a shorthand for conjugate transposition sometimes we may employ $M^\dagger$ when it is more convenient.

For the matrix to represent a valid quantum operation, it must be unitary. A matrix U is unitary if it satisfies:

$$UU^\dagger = I$$

where $U^\dagger$ is the conjugate transpose of U, and I is the identity matrix.

Example 23. The operation is represented by this matrix:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Checking that H is unitary is a straightforward calculation:

$$H^\dagger H = \left(\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right)^\dagger \left(\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$= \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \cdot 1 + 1 \cdot 1 & 1 \cdot 1 + 1 \cdot (-1) \\ 1 \cdot 1 + (-1) \cdot 1 & 1 \cdot 1 + (-1) \cdot (-1) \end{pmatrix}$$

$$= \frac{1}{2} \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Thus, H is unitary.

Example 24. Let us apply the Hadamard operation H, followed by a operation B, and then another Hadamard operation. The combined operation is:

$$HBH = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}}_{B} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1+i}{2} & \frac{1-i}{2} \\ \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix}$$

Applying this resulting unitary operation twice yields:

$$(HBH)^2 = \begin{pmatrix} \frac{1+i}{2} & \frac{1-i}{2} \\ \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix}^2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Remark 60. The concept of forming linear combinations of states is prevalent and sometimes referred as *superposition*.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle.$$

Where both $\alpha \in \mathbb{C}$ $\beta \in \mathbb{C}$. From a different perspective, the state of a qubit can be described as a vector in a two-dimensional complex vector space. This vector space has two special states, denoted as $|0\rangle$ and $|1\rangle$, which are referred to as computational basis states. These basis states are both orthogonal and normalized, forming a complete basis for representing any qubit state in this space. A qubit is described by a normalized superposition state

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

where the coefficients satisfy the normalization condition

$$|\alpha|^2 + |\beta|^2 = 1.$$

Remark 61. When the quantum state $|0\rangle$ is measured, the result is always 0. Similarly, measuring the quantum state $|1\rangle$ always yields the result 1.

Quantum Registers

A quantum register consisting of n qubits can be described as a quantum system with a state vector in a 2^n-dimensional Hilbert space. Mathematically, the state of an n-qubit quantum register can be represented as a vector in the complex vector space $\mathbb{C}^{2^n}$. Tensor products are essential for the systems that are made of at least two qubits, because they allow us to describe systems with multiple particles. Imagine you have two qubits, each representing a simple piece of information. When you put them together, you need a way to describe their combined state.

Definition 102. (Tensor Product) Given two vectors $|\psi_1\rangle \in \mathbb{C}^{d_1}$ and $|\psi_2\rangle \in \mathbb{C}^{d_2}$, their tensor product is defined as:

$$|\psi_1\rangle \otimes |\psi_2\rangle = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_{d_1} \end{pmatrix} \otimes |\psi_2\rangle = \begin{pmatrix} \alpha_1|\psi_2\rangle \\ \vdots \\ \alpha_{d_1}|\psi_2\rangle \end{pmatrix},$$

and the resulting vector $|\psi_1\rangle \otimes |\psi_2\rangle$ resides in the combined state space $\mathbb{C}^{d_1} \otimes \mathbb{C}^{d_2}$.

For example, if $|\psi_1\rangle = \begin{pmatrix} 1 \\ \\ 2 \end{pmatrix}$ and $|\psi_2\rangle = \begin{pmatrix} a \\ \\ b \end{pmatrix}$, then their tensor product is:

$$|\psi_1\rangle \otimes |\psi_2\rangle = \begin{pmatrix} 1 \\ \\ 2 \end{pmatrix} \otimes \begin{pmatrix} a \\ \\ b \end{pmatrix} = \begin{pmatrix} 1 \cdot \begin{pmatrix} a \\ \\ b \end{pmatrix} \\ \\ 2 \cdot \begin{pmatrix} a \\ \\ b \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a \\ \\ b \\ \\ 2a \\ \\ 2b \end{pmatrix}.$$

Definition 103. (Tensor Product of Matrices) Given two matrices $A \in \mathbb{C}^{m \times n}$ and $B \in \mathbb{C}^{p \times q}$, their tensor product $A \otimes B$ is an $mp \times nq$ matrix formed by multiplying each element of A by the entire matrix B. Mathematically, if $A = \begin{pmatrix} a_{11} & a_{12} \\ \\ a_{21} & a_{22} \end{pmatrix}$ and $B = \begin{pmatrix} b_{11} & b_{12} \\ \\ b_{21} & b_{22} \end{pmatrix}$, then:

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B \\ \\ a_{21}B & a_{22}B \end{pmatrix} = \begin{pmatrix} a_{11} \begin{pmatrix} b_{11} & b_{12} \\ \\ b_{21} & b_{22} \end{pmatrix} & a_{12} \begin{pmatrix} b_{11} & b_{12} \\ \\ b_{21} & b_{22} \end{pmatrix} \\ \\ a_{21} \begin{pmatrix} b_{11} & b_{12} \\ \\ b_{21} & b_{22} \end{pmatrix} & a_{22} \begin{pmatrix} b_{11} & b_{12} \\ \\ b_{21} & b_{22} \end{pmatrix} \end{pmatrix}.$$

For example, if $\mathbf{u} = \begin{pmatrix} 3 & 1 \\ 4 & 1 \end{pmatrix}$ and $\mathbf{v} = \begin{pmatrix} 5 & 9 \\ 2 & 6 \end{pmatrix}$, then their tensor product is:

$$\mathbf{u} \otimes \mathbf{v} = \begin{pmatrix} 3\begin{pmatrix} 5 & 9 \\ 2 & 6 \end{pmatrix} & 1\begin{pmatrix} 5 & 9 \\ 2 & 6 \end{pmatrix} \\ 4\begin{pmatrix} 5 & 9 \\ 2 & 6 \end{pmatrix} & 1\begin{pmatrix} 5 & 9 \\ 2 & 6 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} 15 & 27 \\ 6 & 18 \end{pmatrix} & \begin{pmatrix} 5 & 9 \\ 2 & 6 \end{pmatrix} \\ \begin{pmatrix} 20 & 36 \\ 8 & 24 \end{pmatrix} & \begin{pmatrix} 5 & 9 \\ 2 & 6 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 15 & 27 & 5 & 9 \\ 6 & 18 & 2 & 6 \\ 20 & 36 & 5 & 9 \\ 8 & 24 & 2 & 6 \end{pmatrix}.$$

Example 25. A system composed of two qubits resides in a four-dimensional Hilbert space, denoted as $\mathcal{H}_4 = \mathcal{H}_2 \otimes \mathcal{H}_2$.

Basis set of $\mathcal{H}_4$ can be listed as $\{|0\rangle|0\rangle, |0\rangle|1\rangle, |1\rangle|0\rangle, |1\rangle|1\rangle\}$ or $\{|00\rangle|01\rangle, |10\rangle|11\rangle\}$ for shorter representation. The state of $\mathcal{H}_4$ can be represented as a normalized vector given by

$$c_0|00\rangle + c_1|01\rangle + c_2|10\rangle + c_3|11\rangle,$$

where the normalization condition requires that

$$|c_0|^2 + |c_1|^2 + |c_2|^2 + |c_3|^2 = 1.$$

It is also important to note that the observation of the number of qubits does not alter the normalized length. In other words, individual qubit observations are still required to adhere to unit length.

Definition 104. (Decomposable States) A state $z \in H_4$ is called decomposable if it can be written as a tensor product of two states from H_2. Mathematically, this is expressed as:

$$z = x \otimes y$$

where $x, y \in H_2$.

In simpler terms, a decomposable state is like two separate puzzle pieces that fit together without changing their individual shapes. Each part can be independently described without referring to the other.

Definition 105. (Entangled States) A state that is not decomposable is referred to as an entangled state. This means that the state $z \in H_4$ cannot be written as a simple product of two states in H_2. Instead, the state z has properties that cannot be separated into individual contributions from each qubit.

Think of entangled states as puzzle pieces that have morphed together so that the whole picture cannot be understood by looking at each piece alone; they are interconnected in a way that each piece affects the other. Famous example for an entangled state is Bell State.

Definition 106 (Bell State)**.** One of the four maximally entangled two-qubit states, known as Bell States, is given by:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right).$$

This state is also commonly referred to as the Einstein-Podolsky-Rosen pairs. In relation to the correlation observed in the Bell State, the Einstein-Podolsky-Rosen (EPR) paradox represents a conceptual experiment in quantum mechanics. It involves entangled particles that demonstrate instantaneous correlations, irrespective of the distance between them. This paradox challenges classical concepts of locality and completeness of quantum theory [107]. Another famous variation originates from the same question is *Schrödinger's Cat* as a hypothetical thought experiment. It is also interesting to notice that the experiments have shown that this correlation can remain even if the qubits are spatially separated more than 10 kilometers [108] .

No-Cloning Theorem

The no-cloning theorem in quantum mechanics asserts that it is fundamentally impossible to create an exact copy of an arbitrary unknown quantum state. This theorem has significant implications in various domains, including quantum computing. The no-cloning theorem is an extension of the no-go theorem proposed by James Park in 1970 [109], which established that a perfect and non-intrusive measurement scheme cannot exist. This result was independently rediscovered in 1982 by both William Wootters and Wojciech H. Zurek [110], as well as by Dennis Dieks [111].

Consider a quantum system having n basis states $|a_1\rangle, \ldots, |a_n\rangle$. Let us denote the state space by $\mathcal{H}_n$ and specify the state $|a_1\rangle$ to be the *blank sheet state*. A unitary mapping in $\mathcal{H}_n \otimes \mathcal{H}_n$ is called a quantum copymachine, if for any state $|x\rangle \in \mathcal{H}_n$,

$$U(|x\rangle|a_1\rangle) = |x\rangle|x\rangle.$$

Theorem 23. *(No-Cloning Theorem) For $n > 1$, there is no quantum copymachine. [112]*

The following proof is borrowed from the work of [97].

Proof. Assume, for the sake of contradiction, that a quantum copymachine exists for $n > 1$. This implies the existence of a unitary operator U such that for any state $|x\rangle \in H_n$, we have:

$$U(|x\rangle|a_1\rangle) = |x\rangle|x\rangle.$$

Given that $n > 1$, there exist two orthogonal states $|a_1\rangle$ and $|a_2\rangle$. According to the definition of the quantum copymachine, we should have:

$$U(|a_1\rangle|a_1\rangle) = |a_1\rangle|a_1\rangle \quad \text{and} \quad U(|a_2\rangle|a_1\rangle) = |a_2\rangle|a_2\rangle.$$

Consider the superposition state $\frac{1}{\sqrt{2}}(|a_1\rangle + |a_2\rangle)$. The unitary operator U should satisfy:

$$U\left(\frac{1}{\sqrt{2}}(|a_1\rangle + |a_2\rangle)|a_1\rangle\right) = \left(\frac{1}{\sqrt{2}}(|a_1\rangle + |a_2\rangle)\right)\left(\frac{1}{\sqrt{2}}(|a_1\rangle + |a_2\rangle)\right).$$

Expanding the right-hand side, we get:

$$\left(\frac{1}{\sqrt{2}}(|a_1\rangle + |a_2\rangle)\right)\left(\frac{1}{\sqrt{2}}(|a_1\rangle + |a_2\rangle)\right) = \frac{1}{2}(|a_1\rangle|a_1\rangle + |a_1\rangle|a_2\rangle + |a_2\rangle|a_1\rangle + |a_2\rangle|a_2\rangle).$$

Since U is linear, we can also write:

$$U\left(\frac{1}{\sqrt{2}}(|a_1\rangle + |a_2\rangle)|a_1\rangle\right) = \frac{1}{\sqrt{2}}U(|a_1\rangle|a_1\rangle) + \frac{1}{\sqrt{2}}U(|a_2\rangle|a_1\rangle).$$

Substituting the values from the assumption of the quantum copymachine, we get:

$$\frac{1}{\sqrt{2}}|a_1\rangle|a_1\rangle + \frac{1}{\sqrt{2}}|a_2\rangle|a_2\rangle.$$

However, the two expressions for $U\left(\frac{1}{\sqrt{2}}(|a_1\rangle + |a_2\rangle)|a_1\rangle\right)$ do not match, leading to a contradiction. Hence, no such quantum copymachine exists. □

Quantum Circuits

In the context of quantum computing, particularly within the quantum circuit model of computation, quantum logic gates (or quantum gates) refer to fundamental quantum circuits that act on a limited number of qubits. These gates serve as the fundamental building blocks for constructing quantum circuits, similar to how classical logic gates function in traditional digital circuits. Quantum state transformations for an n-qubit system can be accomplished by deploying a series of single and two-qubit quantum state transformations. Such quantum state transformations, when they operate exclusively on a limited quantity of qubits, are referred to as quantum gates. The chronological implementation of these quantum gates is known as a quantum gate array or a quantum circuit. In the quantum circuit framework, the wires symbolize qubits, while the gates denote unitary operations as well as measurements.

Remark 62. Circuits are also serves as the models of computations, similar to the Turing machines.

Pauli Gate

The Pauli operators, commonly referred to as I, X, Y, and Z, are predominant transformations applied to individual qubits:

The Identity operator (I) can be defined as [17]:

$$I : |0\rangle\langle 0| + |1\rangle\langle 1|$$

which corresponds to the matrix representation:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The Pauli-X operator (X), analogous to the NOT operation in classical computing on the standard basis $|0\rangle$ and $|1\rangle$, is depicted as:

$$X : |1\rangle\langle 0| + |0\rangle\langle 1|$$

with its corresponding matrix being:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

The Pauli-Y operator (Y), a composition of the Pauli-X and Pauli-Z operations, is defined as:

$$Y : -i|1\rangle\langle 0| + i|0\rangle\langle 1|$$

which in matrix form becomes:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

Lastly, the Pauli-Z operator (Z), which alters the relative phase of a superposition in the standard basis, is given by:

$$Z : |0\rangle\langle 0| - |1\rangle\langle 1|$$

and is represented in matrix form as:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

The Pauli operators are typically portrayed graphically as square boxes.

[17] For further details on ket–bra notation, the reader is referred to [97]

Pauli Operator	Gate Diagram
I	$\boxed{I}$
X	$\boxed{X}$
Y	$\boxed{Y}$
Z	$\boxed{Z}$

Table 7: Pauli operators along with corresponding diagrams

The Hadamard Gate

A significant transformation applied to a single qubit is the Hadamard transformation, denoted by $H = \frac{1}{\sqrt{2}}(|0\rangle\langle 0| + |1\rangle\langle 0| + |0\rangle\langle 1| - |1\rangle\langle 1|)$. This transformation can be alternatively expressed as

$$H : \begin{cases} |0\rangle \to |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ |1\rangle \to |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{cases}$$

Through this transformation, an even superposition of $|0\rangle$ and $|1\rangle$ is created from either element of the standard basis. It is crucial to note that the Hadamard transformation is self-inverse, as demonstrated by the relationship $HH = I$. In the context of the standard basis, the Hadamard transformation is represented by the Hadamard matrix H.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

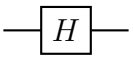

Figure 16: Circuit representation of the Hadamard transformation

Controlled-NOT Gate

The Controlled-NOT gate (commonly denoted as C_{NOT}), which operates within the standard basis of a two-qubit system, utilizes the binary representations $|0\rangle$ and $|1\rangle$ as classical bit states. Its function is as follows: it inverts the state of the second qubit given that the first qubit is in the $|1\rangle$ state, while leaving the second qubit unaltered if the first qubit is in the $|0\rangle$ state. The representational form of the C_{NOT} transformation is given by:

$$C_{\text{NOT}} = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X$$

$$= |0\rangle\langle 0| \otimes (|0\rangle\langle 0| + |1\rangle\langle 1|) + |1\rangle\langle 1| \otimes (|1\rangle\langle 0| + |0\rangle\langle 1|)$$

$$= |00\rangle\langle 00| + |01\rangle\langle 01| + |11\rangle\langle 10| + |10\rangle\langle 11|,$$

By examining this, it becomes straightforward to interpret the effect of the C_{NOT} gate on standard basis elements:

$$\begin{aligned} C_{\text{NOT}} : |00\rangle &\to |00\rangle, \\ |01\rangle &\to |01\rangle, \\ |10\rangle &\to |11\rangle, \\ |11\rangle &\to |10\rangle. \end{aligned}$$

Given the standard basis, the matrix form of the C_{NOT} gate is represented as:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The C_{NOT} gate possesses unitary properties and is characterized by its ability to serve as its own inverse. It's noteworthy that the C_{NOT} gate cannot be expressed as a tensor product of two distinct single-qubit transformations. Graphical representation of C_{NOT} gate differs from the previous ones.

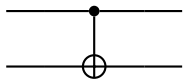

Figure 17: Circuit representation of the C_{NOT} transformation

Phase Shift Gate

The phase shift gate is a category of single-qubit transformations that yield the basis states $|0\rangle \mapsto |0\rangle$ and $|1\rangle \mapsto e^{i\varphi}|1\rangle$. While the probabilities of measuring $|0\rangle$ or $|1\rangle$ are conserved post-application of this gate, it induces a modification in the phase

of the quantum state. This transformation is analogous to effectuating a constant-latitude, or horizontal circle traversal, corresponding to a rotation about the z-axis of the Bloch sphere by an angle of φ radians.

The phase shift gate can be portrayed through the matrix:

$$P(\varphi) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix}$$

Here, φ denotes the phase shift value, characterized by a periodicity of 2π.

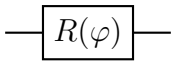

Figure 18: Circuit representation of the Phase Shift transformation

SWAP Gate

The SWAP gate, which performs an interchange operation on two qubits, can be represented in the context of the basis states $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$. The matrix that corresponds to the SWAP gate is:

$$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The SWAP gate can also be expressed in the form of a sum:

$$\text{SWAP} = \frac{I \otimes I + X \otimes X + Y \otimes Y + Z \otimes Z}{2}.$$

Here, I, X, Y, and Z denote the identity and the Pauli matrices, respectively.

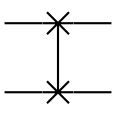

Figure 19: Circuit representation of the SWAP gate

Toffoli Gate

The Toffoli gate, often referred to as the CCNOT gate or the Deutsch gate denoted as $D(\frac{\pi}{2})$, bears the name of its discoverer, Tommaso Toffoli. This gate operates on a 3-bit system, serving as a universal gate for classical computation, however, it is not universal for quantum computation. The quantum variant of the Toffoli gate functions equivalently, but within a 3-qubit system. When considering only input qubits that are $|0\rangle$ and $|1\rangle$, the Toffoli gate induces a Pauli-X (or NOT) operation on the third qubit if the first two qubits are in the $|1\rangle$ state. Otherwise, no operation is performed. This gate epitomizes a controlled-controlled unitary (CC-U) gate. As it is the quantum counterpart of a classical gate, its operations can be thoroughly defined by a corresponding truth table. The universality of the Toffoli gate can be achieved when used in conjunction with the single qubit Hadamard gate.

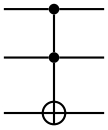

Figure 20: Circuit representation of the Toffoli gate

Reversible Computation

A computation is considered reversible if the input can be uniquely determined from the output. For instance, the binary NOT operation is reversible because if the output bit is 0, the corresponding input bit must have been 1, and conversely, if the output bit is 1, the input must have been 0. In contrast, the AND operation is not reversible because the output does not uniquely determine the input. For example, an output of 0 can result from multiple input combinations—$(0, 0)$, $(0, 1)$, or $(1, 0)$—making it impossible to reconstruct the original inputs from the output alone.

In quantum computing, reversible computation is fundamental and is implemented through quantum gates that manipulate qubits in ways that can be reversed. Quantum gates are inherently reversible because they are represented by unitary matrices, which are mathematical constructs that preserve information and ensure that the original quantum state can be recovered by applying the inverse transformation. This characteristic aligns with the principles of quantum mechanics, where the evolution of closed systems must be unitary.

Examples of reversible quantum gates include the Pauli-X gate, the Hadamard gate (which creates superpositions), and the CNOT gate (which entangles qubits). These gates can be combined to construct quantum circuits that perform complex operations while maintaining reversibility.

Reversibility ensures that no information is lost during computation. This preservation of information is critical for maintaining quantum coherence, allowing key quantum phenomena like superposition and entanglement to persist throughout the

computational process. These phenomena underpin the efficiency of powerful quantum algorithms, such as Shor's algorithm for integer factorization and Grover's algorithm for unstructured search.

Alice, Bob and Entanglenment

Let us call back our old friends, steadfast protagonists of secure communication Alice and Bob. Yes, they are back, still exchanging secrets, and this time, they are getting a quantum upgrade! If you thought their classical cryptographic adventures were thrilling, wait until you see what quantum mechanics has in store for them. They now face a new frontier where quantum mechanics redefines the very nature of security.

In classical cryptography, Alice and Bob have demonstrated the intricacies of secure key exchange, encryption, and decryption. Now, they are poised to take on new challenges where the principles of quantum mechanics redefine security. Imagine their excitement as they transition from traditional bits to the enigmatic qubits, and from conventional communication channels to the strange yet fascinating phenomena of entanglement and superposition.

The quantum realm offers Alice and Bob unprecedented tools for safeguarding their communications. No longer reliant solely on the computational difficulty of problems, they now exploit the fundamental laws of nature to ensure the integrity and confidentiality of their messages. This shift not only strengthens security but also paves the way for innovative protocols, such as quantum key distribution (e.g., BB84) and quantum teleportation, which we will explore in detail.

Consider a scenario where Alice possesses a single qubit in the state

$$a|0\rangle + b|1\rangle$$

However, this state is unknown to Alice. Alice's objective is now to transmit the qubit state to Bob. One potential, albeit theoretical, approach is for Alice to transmit the entire two-state quantum system to Bob. This scenario implies the existence of a quantum channel from Alice to Bob. Conversely, if Alice can send classical bits to Bob, it is said that there is a classical channel between them.

In the absence of a quantum channel but the presence of a classical one, Alice's task of transmitting the state to Bob appears daunting. It is important to note that since the state itself is also unknown to Alice, she cannot provide Bob with instructions on how to reconstruct it.

Consider the following scenario: Alice might attempt to observe her qubit state and subsequently communicate the result, either 0 or 1, to Bob. However, this strategy fails instantly if both parameters a and b are non-zero. Under these conditions, Bob is unable to reconstruct the original state. Moreover, Alice faces significant limitations in performing multiple observations. Each measurement she makes inherently disturbs the state of her qubit.

Additionally, she cannot generate multiple copies of her qubit to facilitate numerous observations, due to impossibility of duplicating an unknown quantum state (recall **No-cloning Theorem**). If Alice were permitted an unlimited number of observations, she could achieve highly precise approximations of the probabilities of obtaining 0 and 1 as outcomes. This means she could approximate the magnitudes $|a|^2$ and $|b|^2$ with great accuracy. However, even possessing the exact values of $|a|^2$ and $|b|^2$ would not suffice for Bob to reconstruct the quantum state. This limitation becomes apparent when considering that the values of $|a|^2$ and $|b|^2$ are identical for the states

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

and

$$\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle .$$

Thus, knowledge of $|a|^2$ and $|b|^2$ alone is insufficient for Bob to distinguish between these two distinct quantum states.

In fact, it is impossible for Alice to transmit her quantum bit to Bob using only a classical channel. To understand this, consider that classical information can be perfectly cloned. If it were possible to reconstruct the state from classical information, then we would be able to produce an unlimited number of reconstructions of the state. This would imply the existence of a quantum copy machine, which is a contradiction.

This brings us to entanglement, a uniquely quantum phenomenon where two particles share a special correlation, such that the state of one instantly influences the other, regardless of distance. By sharing an entangled pair, Alice and Bob can exploit this property to achieve what a classical channel cannot. A classic example is the Bell state:

$$|\phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle),$$

a maximally entangled two-qubit state often called an *e-bit*—a unit of entanglement. When Alice holds qubit A and Bob holds qubit B in this state, they possess a powerful tool for quantum communication. Known as an Einstein-Podolsky-Rosen (EPR) pair, this shared resource enables protocols like quantum teleportation, which we now examine.

It is customary to regard entanglement as a valuable resource that can be harnessed to perform various tasks. When we adopt this perspective, the state $|\phi^+\rangle$ is viewed as representing a single unit of entanglement, commonly referred to as an e-bit. To say that Alice and Bob share an e-bit is to envision a scenario where Alice possesses a qubit A and Bob holds a qubit B. Together, this pair (A, B) exists in the entangled state $|\phi^+\rangle$. On the other hand, if Alice and Bob initially share an EPR pair, there is a method to accomplish the necessary task which is known as quantum teleportation. Alice and Bob share one e-bit: Alice possesses qubit A, Bob possesses qubit B, and the combined system (A, B) is in the state $|\phi^+\rangle$. Additionally, Alice has a qubit Q that she wishes to transmit to Bob.

Algorithm 10 Quantum Teleportation Protocol

1: Alice performs a CNOT operation on Q (control) and A (target).
2: Alice applies a Hadamard gate to Q.
3: Alice measures Q and A, obtaining bits q and a (each 0 or 1).
4: Alice sends q and a to Bob via the classical channel.
5: Bob applies operations to B based on q and a:
- If $qa = 00$: Identity (no operation).
- If $qa = 01$: Apply Z.
- If $qa = 10$: Apply X.
- If $qa = 11$: Apply XZ.

After these steps, Bob's qubit B assumes the state $|\psi\rangle$, including any pre-existing entanglement Q had with other systems. For instance, if Q was entangled with a third qubit C, B inherits that entanglement post-teleportation. Crucially, this is not cloning: the original qubit Q collapses to $|q\rangle$ due to Alice's measurement, and the e-bit $|\phi^+\rangle_{AB}$ is consumed, as A and B are no longer entangled. This respects the no-cloning theorem while achieving perfect state transfer. Additionally, the entanglement resource, or e-bit, is consumed in the process: qubit A transitions to state $|a\rangle$ and loses its entanglement with B (or any other system). This depletion of the e-bit represents the inherent cost of teleportation.

Remark 63. This is a teleportation of the state, not a physical qubits are transmitted.

Quantum Fourier Transforms

The classical Fourier transform, widely employed in mathematics and signal processing, converts a function $f(x)$ from the time domain to the frequency domain.

Definition 107 (Classical Fourier Transform)**.** For a continuous function $f : \mathbb{R} \to \mathbb{C}$, the Fourier transform $\hat{f}(k)$ is defined as:

$$\hat{f}(k) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i k x} \, dx,$$

where the exponential term encodes frequency components.

In discrete settings, the Fast Fourier Transform (FFT) computes this transformation for a sequence of N points in $O(N \log N)$ operations, a result of considerable practical utility. This provides a basis for comparison with its quantum counterpart.

In quantum computation, data is represented as superpositions of discrete states across n qubits, spanning a 2-dimensional Hilbert space. A classical FFT applied to such a system would necessitate an exponential number of operations to process the full state space, rendering it impractical. The Quantum Fourier Transform, by contrast, operates directly on the quantum state, transforming its amplitudes in a manner that preserves coherence and capitalises on quantum parallelism. This adaptation motivates its formal definition.

Definition 108 (Quantum Fourier Transform)**.** Let n be a positive integer, and consider an n-qubit state in the computational basis:

$$|\psi\rangle = \sum_{x=0}^{2^n-1} \alpha_x |x\rangle,$$

where $\alpha_x \in \mathbb{C}$ satisfy the normalisation condition $\sum_{x=0}^{2^n-1} |\alpha_x|^2 = 1$, and $|x\rangle$ denotes the basis state corresponding to integer $x \in \{0, 1, \ldots, 2^n - 1\}$. The Quantum Fourier Transform, denoted QFT, is the unitary operator mapping $|\psi\rangle$ to:

$$\mathrm{QFT}|\psi\rangle = \sum_{k=0}^{2^n-1} \beta_k |k\rangle,$$

where the coefficients β_k are given by:

$$\beta_k = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} \alpha_x e^{2\pi i \frac{kx}{2^n}}.$$

Remark 64. For a single basis state $|x\rangle$, the action of the QFT is:

$$\mathrm{QFT}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi i \frac{kx}{2^n}} |k\rangle,$$

yielding a uniform superposition with phase factors determined by x.

Definition 109 (Matrix Representation)**.** The QFT is represented by a unitary matrix $F_n \in \mathbb{C}^{2^n \times 2^n}$, with elements:

$$(F_n)_{k,x} = \frac{1}{\sqrt{2^n}} e^{2\pi i \frac{kx}{2^n}}, \quad k, x \in \{0, 1, \ldots, 2^n - 1\},$$

where the factor $\frac{1}{\sqrt{2^n}}$ ensures $\langle F_n^\dagger F_n \rangle = I$.

The QFT's implementation on a quantum computer requires $O(n^2)$ elementary gates, a polynomial cost in the number of qubits, contrasting with the FFT's $O(N \log N) = O(2^n n)$ operations for $N = 2^n$ points. This efficiency arises from the QFT's capacity to transform all 2^n amplitudes concurrently, a consequence of quantum superposition. Hirvensalo's book [97] provides a detailed and sophisticated analysis of these topics. The Quantum Fourier Transform finds application in many quantum algorithms that involve phase or frequency analysis. For instance, Shor's algorithm uses the QFT to identify the period of a modular exponential function, which is essential for factoring large integers. In quantum phase estimation, the QFT is used to estimate the eigenvalue (or phase) of a unitary operator, serving as a foundational subroutine. The Harrow-Hassidim-Lloyd algorithm applies the QFT to determine the eigenvalues of a Hermitian matrix, enabling the solution of linear systems. Certain versions of the Quantum Approximate Optimization Algorithm use the QFT to manipulate phases. Simpler algorithms such as the Deutsch-Jozsa and Bernstein-Vazirani algorithms use related transformations like the Hadamard operation.

References

[1] David Klotz. The enigmatic frieze of ramesses ii at luxor temple. *Enigmatic Writing in the Egyptian New Kingdom: Revealing, Transforming, and Display in Egyptian Hieroglyphs*, pages 49–99, 2020.

[2] Richard A Mollin. *An introduction to cryptography*. CRC Press, 2000.

[3] Ibrahim A Al-Kadit. Origins of cryptology: The arab contributions. *Cryptologia*, 16(2):97–126, 1992.

[4] Jeffrey Hoffstein, Jill Pipher, Joseph H Silverman, Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. *An Introduction to Cryptography*. Springer, 2014.

[5] H-R Schuchmann. Enigma variations. In *Cryptography: Proceedings of the Workshop on Cryptography Burg Feuerstein, Germany, March 29–April 2, 1982 1*, pages 65–68. Springer, 1983.

[6] Cipher A Deavours and Louis Kruh. The turing bombe: was it enough? *Cryptologia*, 14(4):331–349, 1990.

[7] B Jack Copeland. *Colossus: The secrets of Bletchley Park's code-breaking computers*. Oxford University Press, 2010.

[8] Whitfield Diffie and Susan Landau. The export of cryptography in the 20th and the 21st centuries. In *The History of Information Security*, pages 725–736. Elsevier, 2007.

[9] Whitfield Diffie and Martin E Hellman. New directions in cryptography. In *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*, pages 365–390. Association for Computing Machinery, New York, NY, United States, 2022.

[10] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 11–20, 1993.

[11] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[12] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108, 1996.

[13] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 84–93, 2005.

[14] Chris Peikert et al. A decade of lattice cryptography. *Foundations and trends® in theoretical computer science*, 10(4):283–424, 2016.

[15] Claude E Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949.

[16] Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.

[17] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001.

[18] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[19] John Talbot and Dominic JA Welsh. *Complexity and cryptography: an introduction*, volume 13. Cambridge University Press, 2006.

[20] David Alan Grier. *When computers were human*. Princeton University Press, 2013.

[21] Gregg Jaeger. Classical and quantum computing. *Quantum Information: An Overview*, pages 203–217, 2007.

[22] Alan Mathison Turing et al. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[23] Hartley Rogers Jr. *Theory of recursive functions and effective computability*. MIT press, 1987.

[24] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, pages 781–793, 2004.

[25] Boaz Barak. Introduction to theoretical computer science. *Under preperation*, 2022.

[26] Lance Fortnow and Steve Homer. A short history of computational complexity. *Bulletin of the EATCS*, 80(01):2003, 2003.

[27] Michael R Garey and David S Johnson. Computers and intractability. *A Guide to the*, 1979.

[28] Christof Paar and Jan Pelzl. *Understanding cryptography*, volume 1. Springer, 2010.

[29] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[30] Auguste Kerckhoffs. La cryptographie militaire. *J. Sci. Militaires*, 9(4):5–38, 1883.

[31] Letter frequencies. `https://www3.nd.edu/~busiforc/handouts/cryptography/letterfrequencies.html`, 2023.

[32] F.L. Bauer. Vernam cipher. In H.C.A. van Tilborg and S. Jajodia, editors, *Encyclopedia of Cryptography and Security*. Springer, Boston, MA, 2011.

[33] Chris Peikert. Theory of cryptography: Lecture 2 - computational hardness, 2019. Accessed: 2024-06-17.

[34] Alexander W Dent. Choosing key sizes for cryptography. *information security technical report*, 15(1):21–27, 2010.

[35] Morris J Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. 2015.

[36] Mihir Bellare and Tadayoshi Kohno. Hash function balance and its impact on birthday attacks. In *Advances in Cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings 23*, pages 401–418. Springer, 2004.

[37] Ralph C Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, 1978.

[38] JH Ellis. The possibility of non-secret digital encryption. Technical report, CESG Research Report, 1970.

[39] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II 40*, pages 62–91. Springer, 2020.

[40] Craig Gidney and Martin Ekerå. How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021.

[41] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 284–293, 1997.

[42] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. Ntru: A ring-based public key cryptosystem. In *International algorithmic number theory symposium*, pages 267–288. Springer, 1998.

[43] Henri Cohen. *A Course in Computational Algebraic Number Theory*, volume 138 of *Graduate Texts in Mathematics*. Springer Science & Business Media, Berlin, Heidelberg, 1993.

[44] John H. Conway and Neil J. A. Sloane. *Sphere Packings, Lattices and Groups*. Springer-Verlag, 3rd edition, 1999.

[45] Henry Cohn and Abhinav Kumar. The densest lattice in twenty-four dimensions. *Electronic Research Announcements of the American Mathematical Society*, 10(7):58–67, 2004.

[46] Charles Hermite. Extraits de lettres de m. ch. hermite à m. jacobi sur différents objects de la théorie des nombres.(continuation). *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1850(40):279–315, 1850.

[47] Hermann Minkowski. *Geometrie der Zahlen.* Teubner, Leipzig, 1896.

[48] Daniele Micciancio and Shafi Goldwasser. *Complexity of lattice problems: a cryptographic perspective*, volume 671. Springer Science & Business Media, 2002.

[49] Peter M. Gruber and Cornelis G. Lekkerkerker. *Geometry of Numbers.* North-Holland, 1987.

[50] Wojciech Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 296:625–635, 1993.

[51] Gilbert Strang. *Introduction to Linear Algebra.* Wellesley-Cambridge Press, 5th edition, 2016.

[52] Gorjan Alagic, Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, et al. Status report on the first round of the nist post-quantum cryptography standardization process. 2019.

[53] Miklós Ajtai. The shortest vector problem in l2 is np-hard for randomized reductions (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 10–19, New York, NY, USA, 1998. Association for Computing Machinery.

[54] Carl Friedrich Gauss and William C Waterhouse. *Disquisitiones arithmeticae.* Springer, 2018.

[55] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261:515–534, 1982.

[56] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical computer science*, 53(2-3):201–224, 1987.

[57] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1468–1480. SIAM, 2010.

[58] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using dynamic programming. *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, 2011. Later optimized with sieving techniques.

[59] Peter van Emde Boas. Another np-complete problem and the complexity of computing short vectors in a lattice. *Tecnical Report, Department of Mathmatics, University of Amsterdam*, 1981.

[60] Oded Goldreich, Daniele Micciancio, Shmuel Safra, and J-P Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Information Processing Letters*, 71(2):55–61, 1999.

[61] László Babai. On lovász'lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986.

[62] Adeline Langlois and Damien Stehle. Worst-case to average-case reductions for module lattices. Cryptology ePrint Archive, Paper 2012/090, 2012.

[63] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM journal on computing*, 37(1):267–302, 2007.

[64] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206, 2008.

[65] Daniele Micciancio and Chris Peikert. Hardness of sis and lwe with small parameters. In *Annual cryptology conference*, pages 21–39. Springer, 2013.

[66] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on computing*, 43(2):831–871, 2014.

[67] Steven D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, version 2.0 edition, 2012.

[68] Jianwei Li and Phong Q Nguyen. A complete analysis of the bkz lattice reduction algorithm. *Journal of Cryptology*, 38(1):1–58, 2025.

[69] Nicolas Gama and Phong Q Nguyen. Predicting lattice reduction. In *Advances in Cryptology–EUROCRYPT 2008: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings 27*, pages 31–51. Springer, 2008.

[70] Ulrich Fincke and Michael Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of computation*, 44(170):463–471, 1985.

[71] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 193–206, 1983.

[72] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66:181–199, 1994.

[73] Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 601–610, 2001.

[74] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the shortest and closest lattice vector problems. In *International Conference on Coding and Cryptology*, pages 159–190. Springer, 2011.

[75] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2n time using discrete

gaussian sampling. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 733–742, 2015.

[76] Phong Q Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008.

[77] Xiaoyun Wang, Guangwu Xu, and Xuexin Zheng. Improved nguyen-vidick heuristic sieve algorithm for shortest vector problem. *Advances in Cryptology – ASIACRYPT 2011*, 7073:1–19, 2011.

[78] Jiang Zhang, Yingjie Pan, and Chenghuai Hu. A three-level sieve algorithm for the shortest vector problem. *Information Security and Cryptology – Inscrypt 2013*, 8567:47–58, 2013.

[79] Anja Becker, Nicolas Gama, and Antoine Joux. A new technique for solving hard lattice problems overlattices: Reduced complexity for svp and cvp. *Advances in Cryptology – EUROCRYPT 2014*, 8441:159–178, 2014.

[80] Thijs Laarhoven. Angular lsh for approximate nearest neighbors. *arXiv preprint*, 2014.

[81] Thijs Laarhoven. Search problems in cryptography. *PhD Thesis, Eindhoven University of Technology*, 2015.

[82] Ward Beullens, Jan-Pieter D'Anvers, Andreas T Hülsing, Tanja Lange, Lorenz Panny, Cyprien de Saint Guilhem, and Nigel P Smart. Post-quantum cryptography: Current state and quantum mitigation. 2021.

[83] Ettore Majorana. Teoria simmetrica dell'elettrone e del positrone. *Il Nuovo Cimento (1924-1942)*, 14(4):171–184, 1937.

[84] Catherine Bolgar. Microsoft's majorana 1 chip carves new path for quantum computing. *Microsoft Source*, 2025.

[85] Clémence Chevignard, Pierre-Alain Fouque, and André Schrottenloher. Reducing the number of qubits in quantum factoring. Cryptology ePrint Archive, Paper 2024/222, 2024.

[86] Martin Roetteler, Michael Naehrig, Krysta M Svore, and Kristin Lauter. Quantum resource estimates for computing elliptic curve discrete logarithms. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II 23*, pages 241–270. Springer, 2017.

[87] Andrew Wiles. Modular elliptic curves and fermat's last theorem. *Annals of mathematics*, 141(3):443–551, 1995.

[88] Thomas Little Heath et al. *The thirteen books of Euclid's Elements*. Courier Corporation, 1956.

[89] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[90] Russell Impagliazzo. A personal view of average-case complexity. In *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*, pages 134–147. IEEE, 1995.

[91] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 44–61, 1989.

[92] Andrew C Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)*, pages 80–91. IEEE, 1982.

[93] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, 2008.

[94] Jiří Adámek, Horst Herrlich, and George Strecker. *Abstract and concrete categories*. Wiley-Interscience, 1990.

[95] Charles Averill. A brief analysis of the apollo guidance computer. *arXiv preprint arXiv:2201.08230*, 2022.

[96] Max Planck. On the law of distribution of energy in the normal spectrum. *Annalen der physik*, 4(553):1, 1901.

[97] Mika Hirvensalo. *Quantum computing*. Springer Science & Business Media, 2003.

[98] Edward M Purcell and David J Morin. *Electricity and magnetism*. Cambridge university press, 2013.

[99] Louis De Broglie. *Recherches sur la théorie des quanta*. PhD thesis, Migration-université en cours d'affectation, 1924.

[100] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21:467–488, June 1982.

[101] Richard P Feynman. Quantum mechanical computers. *Optics news*, 11(2):11–20, 1985.

[102] Paul Benioff. Quantum mechanical hamiltonian models of turing machines. *Journal of Statistical Physics*, 29:515–546, 1982.

[103] David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[104] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, 1992.

[105] Daniel R Simon. On the power of quantum computation. *SIAM journal on computing*, 26(5):1474–1483, 1997.

[106] Ronald De Wolf. Quantum computing: Lecture notes. *arXiv preprint arXiv:1907.09415*, 2019.

[107] John S Bell. On the einstein podolsky rosen paradox. *Physics Physique Fizika*, 1(3):195, 1964.

[108] Juan Yin, Ji-Gang Ren, He Lu, Yuan Cao, Hai-Lin Yong, Yu-Ping Wu, Chang Liu, Sheng-Kai Liao, Fei Zhou, Yan Jiang, Xin-Dong Cai, Ping Xu, Ge-Sheng Pan, Jian-Jun Jia, Yong-Mei Huang, Hao Yin, Jian-Yu Wang, Yu-Ao Chen, Cheng-Zhi Peng, and Jian-Wei Pan. Satellite-based entanglement distribution over 1200 kilometers. *Science*, 356(6343):1140–1144, June 2017.

[109] James L. Park. The concept of transition in quantum mechanics. *Foundations of Physics*, 1(1):23–33, February 1970.

[110] William K. Wootters and Wojciech H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, October 1982.

[111] Dennis Dieks. Communication by epr devices. *Physics Letters A*, 92(6):271–272, October 1982.

[112] William K Wootters and Wojciech H Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982.