Simulation of Quantum Homomorphic Encryption

Sohrab Ganjian

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the degree of
Master of Science Mathematics and Statistics[1]

Department of Mathematics and Statistics
Faculty of Science
University of Ottawa

[1]The M.Sc. program is a joint program with Carleton University, administered by the Ottawa-Carleton Institute of Mathematics and Statistics

Abstract

When interacting with a cloud server, two ideals we could aim for, among others, are performing computation and ensuring the privacy of the data. Quantum homomorphic encryption is a cryptographic primitive that seeks to address these two objectives in the quantum setting, allowing for quantum computation on encrypted quantum data. In this work, we provide a software simulation of a quantum homomorphic encryption scheme, known as the "EPR scheme", for universal quantum circuits, developed by Broadbent and Jeffery.

We demonstrate the near-term viability of this scheme and provide evidence that the computational cost of classical subroutines in the simulation is negligible compared to the cost of simulating the quantum operations. The simulation package is an open-source Python implementation, called "py-qhe-epr", with a focus on extensive documentation and ease of use for educational, verification and adaptability purposes, supporting future research and development. Furthermore, this implementation serves as a step towards further hardware applications of quantum homomorphic encryption between networked quantum devices.

Lay Summary

Sometimes we need to perform computations that are beyond the capabilities of our own personal computers. For example, instead of analyzing the medical data of just tens of individuals, we might be tasked with analyzing data from millions of people. In such cases, personal computers might not be able to handle the task, so we turn to more powerful, remote computers for these calculations. However, if our data contains sensitive information, like private medical records or personal information, we may not want to share this raw data with anyone, including the remote powerful computer.

Homomorphic encryption is a cryptographic tool that allows us to send data securely. The powerful computer can perform the required computations on the encrypted data without ever seeing the actual content. Once the computation is done, the results are sent back to us in the encrypted form, and through a transformation on our end, we can retrieve the final result without compromising the privacy of our sensitive information.

So far, we have considered classical data, which is stored as digital information in the form of 0s and 1s. If we allow our data to exist as both 0 and 1 simultaneously, while satisfying additional constraints of quantum mechanics, we obtain quantum data. Quantum homomorphic encryption extends the objectives of classical homomorphic encryption to quantum data. This work explores quantum homomorphic encryption and implements a practical simulation software to help bring theory closer to reality.

Acknowledgements

I would like to thank my supervisor Dr. Anne Broadbent and my research collaborator Dr. Connor Paddock, who shared their insights and wisdom throughout the development of this work, while being supportive and present at every step of the way.

I would also like to thank our research group members, past and present: Eric, Sébastian, Martti, Silvia, Daniel, Bennett, Sherry, Nagisa, Josh, Jyoti, and Denis. Special thanks to Pierre for being a great gym partner, to Christine for sharing her experiences, and to Peter, who was always there for a conversation about anything.

I would like to extend my gratitude to Natural Resources Canada for offering me an internship opportunity and allowing me to put my studies into practice. It has been an extremely rewarding and enjoyable experience. I would like to thank my manager, Zarrin Langari.

I would also like to acknowledge the many individuals in the service industry who have directly or indirectly supported me throughout my master's degree. This includes those working behind the scenes, such as conference organizers, construction workers, pilots and airplane crew, train and bus personnel, and hotel and cleaning staff. I am deeply grateful to all who made it possible for me to attend various conferences across Canada and internationally, which led to a richer experience, both intellectually and culturally.

I thank my friends and family for their love and support. A special thanks to my close friend, Kourosh Bagheri Sadr, who is more like a brother, for always being there to talk with me. I would also like to thank my father, Enayat, my sister, Sanam, and my mother, Neda, for their unconditional love and support, even though what I study probably seems more like a topic from a science fiction movie to them!

Finally, I would like to take a moment for some self-gratitude. I express my utmost gratitude to David Goggins for being an inspiring example who guided me through one of the most difficult times in my life. I entered this program as young adult and now I am leaving it as a man. With transitioning out of a 4-year relationship, failing a course, feeling lost in the quantum realm, and loneliness among other difficult life experiences, I was in the perfect condition to fail the program. But I chose to fight back, stay disciplined, make sacrifices and work as hard as I could. I believe there is so much potential within every single one of us, but we must extract it by putting ourselves in uncomfortable situations. Pain builds character, and we must embrace it, because on the other side of it is the person we want to become. I completed my first ever marathon in May of this year, while running the last 24 km on one leg due to a previous injury. Through that experience, I learnt firsthand that our minds are far more powerful than we think. Today, I am wrapping up my master's program with one publication and a second under review, while working full-time at Natural Resources Canada. I am proud of myself, but I also acknowledge that there is still much more within me. While I do not know what the future holds for me, I strive to be the best I can be. With that, I would like to thank myself for not giving up on me and for believing in me.

Ottawa, September 2024

List of Publications

The following is a list of publications by the author, with an asterisk (∗) indicating work has been incorporated into thesis.

- S. Ganjian, C. Paddock, and A. Broadbent. Demonstrating quantum homomorphic encryption through simulation. 2024. arXiv preprint. To appear in QCE24. arXiv:2406.16247 [quant-ph].*
- V. Nguyen and S. Ganjian. Crime attractors and offender mobility. *SIAM Undergraduate Research Online*, vol. 14, 2021. Project Advisors: Razvan Fetecau, Patricia L. Brantingham. Completed at Simon Fraser University. doi:10.1137/21S1420551.

Contents

List of Figures

List of Symbols and Abbreviations

Symbol/ Abbreviation	Description	Page
Ω	Sample space	6
$\mathscr{P}(A)$	Power set of set A	6
$\mathcal{F}$	σ-algebra	6
$a \in A$	a is a member of the set A	6
$\emptyset$	Empty set	6
$A \cup B$	Union of sets A and B	7
$A \cap B$	Intersection of sets A and B	7
$\mathbb{Z}$	Set of integers	9
$H \leqslant G$	H is a subgroup of G	9
$N_G(H)$	Normalizer of the subgroup H in the group G	9
$\langle S \rangle$	The subgroup generated by the subset S	10
$\forall$	For all	10
$\exists$	There exists	10
$\oplus$	Addition modulo 2	11
$\mathbb{R}$	Set of real numbers	13
$\mathbb{C}$	Set of complex number	13
$\|\cdot\|$	A norm of a vector space	15
$\mathcal{H}$	Hilbert space	16
$\otimes$	Tensor product	17
$A^\dagger$	The conjugate transpose of matrix A	18
U	Unitary matrix	18
QIS	Quantum information science	21
$\|\cdot\rangle$	Dirac's *ket* notation	22
$\langle\cdot\|$	Dirac's *bra* notation	22
$\{\|0\rangle, \|1\rangle\}$	Computational basis	22
$\{\|+\rangle, \|-\rangle\}$	Hadamard basis	22
$\{\mathsf{X}, \mathsf{Y}, \mathsf{Z}\}$	Pauli gates	27
I	Identity gate	27
$\mathcal{W}_n$	Pauli group acting on n qubits	28
$\mathcal{K}_n$	Clifford group acting on n qubits	28
H	Hadamard gate	28
P	Phase gate	28
CNOT	CNOT gate	28
SWAP	SWAP gate	28
T	T gate	29
C	Quantum circuit	30
ρ	Density matrix	32
Tr	Trace	32
$\rho_{\max}$	Maximally mixed state	34

Symbol/ Abbreviation	Description	Page
ρ_{AB}	The density matrix of systems A and B	35
Tr_A	The partial trace over the A subsystem	35
$\|\mathsf{EPR}\rangle = \|\Phi^+\rangle$	EPR state	37
$\mathcal{M}$	Message space	42
$\mathcal{C}$	Ciphertext space	42
Gen	Key generation algorithm	42
Enc	Encryption algorithm	42
Dec	Decryption algorithm	43
pk	Public key	43
sk	Private key	43
PPT	Probabilistic Polynomial-time Algorithm	46
FHE	Fully homomorphic encryption scheme	48
CHE	Classical homomorphic encryption scheme	48
Eval	Evaluation algorithm	49
PHE	Partially homomorphic encryption scheme	50
MOHE	Multi-operational homomorphic encryption scheme	50
SHE	Somewhat homomorphic encryption scheme	50
LHE	Levelled homomorphic encryption scheme	51
LFHE	Levelled fully homomorphic encryption scheme	51
QOTP	Quantum one-time pad encryption scheme	52
QHE	Quantum Homomorphic Encryption	60
CL	Clifford quantum homomorphic encryption scheme	61
EPR	EPR quantum homomorphic encryption scheme	61
NISQ	Noisy Intermediate-Scale Quantum technology	68
$\gcd(a, b)$	Greatest common divisor of integers a and b	87
$\phi(n)$	Euler phi function for integer n	87
δ_{ij}	Kronecker delta function	92

1 Introduction

Suppose we want to perform a computation, but our available resources are insufficient to carry it out. One solution is to upload our data to cloud servers, such as Microsoft Azure, Amazon Web Services or Google Cloud Platform, which can provide the computational power needed. The server performs the computation and returns the result back to us. Although our objective of performing the computation was achieved in this scenario, the privacy of our data was not guaranteed. During the transmission and storage on the server, the data could be vulnerable to potential malicious servers or unauthorized access by bad actors. This is where homomorphic encryption comes into play. It is a cryptographic protocol that not only encrypts data but also allows meaningful computation to be performed on the encrypted data. When the encrypted data after computation is decrypted by us, we receive the correct computed result, provided the correctness of the protocol and proper execution of it. Homomorphic encryption, first introduced by [RAD78], addresses the privacy concerns mentioned, as the data remains encrypted during both the transmission and storage. Quantum homomorphic encryption extends this concept to the quantum setting, enabling homomorphic computation on quantum data and quantum circuits.

In this thesis, we will present an open-source software implementation [Gan24] of a quantum homomorphic encryption scheme, named "EPR scheme", developed by Broadbent and Jeffery [BJ15].

We continue this section with a brief overview of quantum theory and cryptography in Sections 1.1 and 1.2, respectively. We will then provide a summary of contribution and review the existing works in the literature in Sections 1.3 and 1.4. Finally, in Section 1.5, we provide an outline of the thesis detailing the organization of the remaining sections.

1.1 An Overview of Quantum Theory

With the invention of calculus and profound discoveries of Newton about gravity and other physical phenomena in the 16th century, many physicists spent the next three centuries pushing the frontiers of what is now known as classical physics. However, this perspective changed drastically at the beginning of the 20th century, when a series of discoveries, such as black-body radiation and the photoelectric effect, suggested Newtonian physics could not explain every physical phenomenon. These results indicated that there are other physical theories at play at the particle level. Later discoveries led by Bohr's models of atom, Heisenberg's uncertainty principle and Schrödinger equation paved the path for the development of quantum theory. The term 'quantum' originates from the Latin root, meaning 'something that has quantity' [Oxf24e]. In physics, its use was first associated to Planck and Einstein, appearing in German compound words such as 'Elementarquantum' (electric charge) and

'Energiequanta' (energy quanta) and 'Energieelement' (energy element)[1]. Overtime, 'quantum', and its plural form, 'quanta' became more prevalent. In this context, quantum means "a minimum amount of a physical quantity which can exist, and in multiples of which it can vary."

In contemporary times, quantum theory remains at the forefront of modern physics, with various applications such as magnetic resonance imaging (MRI), atomic clocks, quantum sensing, quantum computing and quantum cryptography. While quantum theory has seen significant advancements since the early 20th century, many important challenges remain in scaling quantum technologies and making them more accessible to the public. We will briefly discuss some of these challenges in the remainder of this work in the context of quantum computing and quantum cryptography.

1.1.1 Quantum Computing

By leveraging the laws of quantum mechanics, quantum computers offer a new model of computation that is capable of solving certain problems much more efficiently than classical computers. To motivate the importance of quantum computers, let us consider the integer factorization problem — a mathematical problem deemed infeasible to solve efficiently using classical computers, yet crucial for various applications, including internet security. A historic breakthrough that sparked significant interest in quantum computing was made possible by Shor's algorithm, which efficiently solves the integer factorization problem using a quantum computer [Sho94]. This example is particularly important because it showcases that quantum computers, theoretically, are capable of offering exponential speedups not just on artificial problems designed solely to demonstrate quantum capabilities without regard to practicality, but rather on problems with real-life applications. Studying the structure of problems where quantum speedups are possible remains an active area of research [BDCG+20].

While quantum computers are theoretically capable of such speedups, the task of building a quantum computer that provides utility by solving real-life problems is an ongoing multidisciplinary effort spanning fields such as mathematics, physics, computer science and engineering.

1.1.1.1 Simulation of Quantum Computing

Although building a powerful quantum computer is still an ongoing process, it does not mean that we must wait for them to be built before running experiments on quantum processes. Small-scale quantum algorithms and simulations of quantum computers can already be executed on classical computers. In fact, some research focuses on developing algorithms specifically designed for simulation on classical computers or small, noisy quantum computers. The main bottleneck in simulating quantum phenomena on classical computers is scalability. As the problems grow in size, classical computers quickly become infeasible for tackling

[1] In German, all nouns are captialized, so the terms 'Elementarquantum', 'Energiequanta' and 'Energieelement' are capitalized correctly.

them. This is one of the key motivations behind building powerful quantum computers, as they will provide the necessary tools for experimenting large-scale quantum phenomena. Further discussion on this subject can be found in Section 6.1.

1.2 An Overview of Cryptography

The word *cryptography*, derived from the Greek roots "crypto-" and "-graphy", combines the two components to mean "the art or practice of writing in cipher; the science of encryption". The prefix "crypto-" means "concealed or hidden", while the suffix "-graphy" refers to a style of writing. Thus, cryptography encompasses the science behind creating secure communications through encryption [Oxf24c, Oxf24b, Oxf24a].

In today's society, keeping secrets is normalized and part of our lives, both within larger societal structures like governments and companies and within smaller social units such as families. Governments might spy on each other to gain inside information without revealing their activities. Additionally, they do not want details of their attacks to be disclosed. In business, companies may finalize a trade agreement but keep the information confidential from the public until a predetermined date. On a personal level, you might purchase a birthday gift for a loved one and keep it a secret until their birthday. Secrets are part of life, whether we are consciously aware of them or not. Even in the animal kingdom, predators conceal their presence from prey until the moment of attack. In this way, animals also keep information away from each other. Thus, hiding information is not exclusive to human beings but is a natural behaviour.

This is where cryptography comes into play. Cryptography aims to study the science of encryption by defining relevant primitives and setting expectations for secure cryptographic protocols. In the modern world and with the advent of the internet, the impact of cryptography touches nearly everyone globally. Anyone who has done online banking has benefitted from cryptography, even if unaware of the processes behind the scenes. However, that is not to say that the study of cryptography has been completed and fully developed. While the field of cryptography has advanced significantly compared to 100 years ago, it still has a long way to go. Even now, the existing cryptographic protocols are not perfect, as cyberattacks and data breaches occur frequently, from highly sensitive environments like nuclear power plants to personal emails accounts. Having said that, the study of quantum mechanics has led to the emergence of a new branch of cryptography: quantum cryptography, which has opened new avenues of secure communication, such as quantum key distribution [BB84]. Consequently, modern cryptography has been divided into a dichotomy of classical and quantum cryptography.

Let us take a moment to reflect on the broader ethical considerations. While it can be argued that withholding some information may be unethical in certain situations, we leave that discussion to philosophers and policymakers. Nonetheless, cryptographers are encouraged to act ethically and consider who they are working for and why, keeping in mind that with great knowledge comes

great responsibility.

1.3 Summary of Contributions

This thesis details the software implementation of the EPR scheme through a Python package named `py-qhe-epr` [Gan24]. The main contributions of this work, outlined in Section 6, are based on joint work with Broadbent and Paddock. A pre-print of this research is available in [GPB24].

To the best of our knowledge, this work is the first open-source implementation of a quantum homomorphic encryption scheme, bridging theory and real-world applications. The aim of this work is to shed light on the challenges of simulating quantum processes and to demonstrate the practicality and viability of this technology. Great effort has been made to include extensive documentation for easier readability of the code. Additionally, careful consideration has gone into designing the code to minimize the risk of incompatibility with future versions of the dependencies used in this implementation, sustaining longer-term functionality. In addition, the code is designed with minimal assumptions about the input, as outlined in Section 6. Lastly, implementing the EPR scheme requires the use of a classical fully homomorphic encryption scheme (FHE) [Gen09], which is not a straightforward task. Therefore, this implementation of the EPR scheme relies on an external package found in [Era20a].

1.4 Related Works

While the concept of homomorphic encryption dates back to the late 1970s [RAD78], its quantum counterpart wasn't explored until the early 2010s. One of the earliest contributions came from the work of Rhode et al., where they introduced "a limited quantum homomorphic encryption using the Boson sampling and multi-walker quantum walk models for quantum computation" [RFG12]. In 2015, Broadbent and Jeffery further advanced the field by providing what they claimed to be the first definition of quantum homomorphic encryption in the computational setting and providing two schemes for universal quantum computing [BJ15]. Subsequent works, such as [DSS16], have made improvements to [BJ15] by including the compactness property[2]. Additionally, the work by Liang [Lia20] constructed a non-compact QHE scheme that has perfect security, as opposed to computational security described in [BJ15].

In the realm of experimental QHE, progress has already been made through both physical demonstration and software implementations. For example, as a physical demonstration, [TFBF+20] "implement[s] and experimentally demonstrate[s] for the first time the quantum homomorphic encryption scheme proposed by Broadbent and Jeffery". On the software side, [YUS22] implements Broadbent and Jeffery's work in the Qiskit library [JATK+24] over the Clifford gates only, without employing classical homomorphic encryption for the classical

[2]The compactness property is a feature of a homomorphic encryption scheme where the decryption process remains independent of the complexity of the evaluated circuit and depends solely the key. This concept will be defined in Section 4.

keys, as done in [BJ15]. Another pre-print by Fernández and Martin-Delgado [FMD24] implements the work of Liang using Qiskit. This thesis distinguishes itself from previous software implementations in three different ways. Firstly, it is theoretically capable of running any quantum circuit. Secondly, this work incorporates a classical homomorphic encryption scheme into the quantum scheme. Thirdly, it provides an open-source software implementation of quantum homomorphic encryption, which could support future research and verification. More details will be discussed in Section 6.

1.5 Outline

We outline the structure of the remainder of this thesis.

Section 2 covers the mathematical preliminaries including probability theory, algebraic structures and linear algebra.

Next, in Section 3, we introduce concepts from quantum information science relevant to this work, including the postulates of quantum theory, along with quantum circuits, density matrix representation, quantum operation on quantum subsystems, quantum entanglement and quantum teleportation.

In Section 4, we introduce cryptographic primitives and terminology, and explain the concepts of perfect security and computational security, followed by defining homomorphic encryption and its various types.

In the next section, Section 5, we extend these cryptographic concepts to the quantum setting. We will define the quantum one-time pad encryption scheme, define quantum homomorphic encryption scheme, and describe the Clifford scheme and the EPR scheme from [BJ15].

In Section 6, we synthesize our findings to describe the implementation of the EPR scheme as part of the `py-qhe-epr` package found in [Gan24]. Finally, in Section 7, we outline potential avenues for improving the existing software and propose future work as an extension of the EPR scheme. Parts of Sections 6 and 7 follow from [GPB24].

2 Mathematical Preliminaries

In this section, we develop the mathematical tools that will be used throughout this work. Section 2.1 introduces important concepts from probability theory, such as probability space and conditional probability, that are relevant to both quantum information science and cryptography. Section 2.2 delves into algebraic structures and homomorphism, which will be used in Sections 4 and 5 when describing homomorphic encryption. The remainder of the section then shifts focus to linear algebra in Section 2.3. We begin by rationalizing the choice of Hilbert space for describing quantum systems, followed by an introduction to tensor products for representing larger quantum systems. The section concludes with a review of important matrices in quantum information science, specifically unitary transformations and Hermitian matrices[3].

While an effort has been made to make this thesis as self-contained as possible, we recommend that interested readers review set theory, functions and real analysis in [Lay14], if needed.

2.1 Probability Theory

In quantum mechanics, quantum particles are subjects to the axioms of probability. Additionally, important cryptographic concepts, such as the notion of perfect security, are better understood and articulated using the language of probability theory. In this section, we will succinctly define the essential concepts from probability theory necessary for this work. This section follows from [Bil95, Dev12]. We begin by defining a probability space.

Definition 2.1 (Probability Space)**.**

Let Ω be a set, called the *sample space*, and let $\mathcal{F}$ be a subset of the power set of Ω, $\mathscr{P}(\Omega)$. Furthermore, the set $\mathcal{F}$ must contain Ω, $\emptyset$ and be closed under taking complements and countable unions. Then $\mathcal{F}$ is called a σ-algebra on Ω. Let Pr be a function that maps the σ-algebra $\mathcal{F}$ to the interval $[0, 1]$:

$$\Pr : \mathcal{F} \to [0, 1], \tag{2.1}$$

such that it satisfies the following three conditions, known as axioms of probability:

(1) $0 \leq \Pr(A) \leq 1$ for $A \in \mathcal{F}$

(2) $\Pr(\emptyset) = 0$, $\Pr(\Omega) = 1$

[3] In light of theoretical impossibility results, such as Gödel's incompleteness theorems [Gö31], we recognize that powerful axiomatic systems, like those in mathematics and physics, cannot prove their own consistency within their own respective framework. Consequently, and until further progress prompts a reassessment of this view, results derived from such systems should not be considered as absolute truths, but rather as true relative to the *unproven yet reasonable assumption* that such axioms themselves are consistent and correct. This comment applies to the entire work.

(3) If $A_1, A_2, \ldots$ is a disjoint sequence of $\mathcal{F}$-sets and if $\bigcup_{i=1}^{\infty} A_i \in \mathcal{F}$ then

$$\Pr\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \Pr(A_i). \tag{2.2}$$

The triple $(\Omega, \mathcal{F}, \Pr)$ is called a *probability space* and the function Pr is a *probability distribution* on the pair $(\Omega, \mathcal{F})$.

In this work, we restrict our focus to probability spaces where the underlying sample space Ω is finite.

Example 2.1. An example of a probability distribution is the uniform distribution. Let $\Omega = \{1, \ldots, n\}$ with $\mathcal{F} = \mathscr{P}(\Omega)$. Then, $\Pr : \mathcal{F} \to [0, 1]$ with $\Pr(\{\omega\}) = \frac{1}{n}$ for all $\omega \in \Omega$ induces a probability distribution on $(\Omega, \mathcal{F})$, known as the uniform distribution.

At times, additional knowledge about certain events changes the probability of another event occurring. For example, consider rolling two fair dice and determining the probability that their sum is less than 6. The probability is $\frac{10}{36} = \frac{5}{18}$, as out of the 36 possible outcomes, their sum is less than 6 in the following 10 outcomes:

$$(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (4,1).$$

However, if we know that the first die shows a 6, then the probability of the sum being less than 6 becomes 0. The concept of conditional probability allows us to analyze such situations, which we define next:

Definition 2.2 (Conditional Probability)**.**
The conditional probability of an event A, given that event B has occurred, is

$$\Pr(A|B) = \frac{\Pr(A \cap B)}{\Pr(B)}, \tag{2.3}$$

where $\Pr(B) > 0$.

If we let A denote the event that "the sum of the two dice is less than 6" and B the event that "the first die shows 6", then we see that $\Pr(A \cap B) = 0$ which implies that $\Pr(A|B) = 0$, as expected.

Using conditional probability, we derive Bayes' Theorem. Recall that the conditional probability is given by $\Pr(A|B) = \frac{\Pr(A \cap B)}{\Pr(B)}$, if $\Pr(B) > 0$. Similarly, $\Pr(B|A) = \frac{\Pr(A \cap B)}{\Pr(A)}$, if $\Pr(A) > 0$. Solving for $\Pr(A \cap B)$, we get $\Pr(A \cap B) = \Pr(B|A) \cdot \Pr(A)$. Substituting this expression into $\Pr(A|B)$ yields the Bayes' theorem:

$$\Pr(A|B) = \frac{\Pr(B|A) \cdot P(A)}{\Pr(B)}. \tag{2.4}$$

We formally state Bayes' theorem next.

Theorem 2.1 (Bayes' Theorem)**.**
Let A and B be events and $\Pr(B) > 0$. Then *Bayes' theorem* asserts

$$\Pr(A|B) = \frac{\Pr(B|A) \cdot P(A)}{\Pr(B)} \tag{2.5}$$

Finally, we conclude this section with the law of total probability, which provides an alternative method to calculate the probability of an event A by considering a partition of the sample space.

Theorem 2.2 (Law of Total Probability)**.**
Let $\{B_i : i = 1, \ldots, n\}$ form a partition on Ω. That is, $\bigcup_{i=1}^{n} B_i = \Omega$ and $B_i \cap B_j = \emptyset$ for any $i, j \in \{1, \ldots, n\}$ with $i \neq j$. Let A be any set. Then the law of total probability states:

$$\Pr(A) = \sum_{i=1}^{n} \Pr(A|B_i) \Pr(B_i). \tag{2.6}$$

2.2 Algebraic Structures

Algebraic structures are abstract mathematical objects defined over sets equipped with operations like addition and multiplications. In the context of homomorphic encryption, our goal is to establish a mapping between two algebraic structures that preserves the operations of the structures. Essentially, we aim to perform operations on encrypted data, which can be viewed as an algebraic structure, while ensuring the ability to decrypt them later. To begin, we introduce the notion of *arity* and proceed to formally define algebraic structures. This section follows from [BS12, AF15].

While humans communicate through natural languages, these languages can be encoded into binary sequences of 0s and 1s, enabling their representation within an abstract mathematical framework. In this framework, algebraic structures such as rings and groups become meaningful.

Definition 2.3 (arity)**.**
For a nonempty set A and a nonnegative integer n, we define $A^0 = \{\emptyset\}$, and, for $n > 0$, A^n is the set of n-tuples of elements from A. An n-ary operation or function on A is any function f such that $f : A^n \to A$; n is the arity or rank of f.

An operation f on A is unary, binary or ternary if its arity is 1,2, or 3, respectively.

Definition 2.4 (Algebraic Structure)**.**
An *algebraic structure* is a non-empty set A coupled with a collection of n-ary operation(s) on A and a finite set of axioms which describes the rules that the said operation(s) must follow.

An algebraic structure could be of any arity, including binary operations like addition and multiplication. Indeed, important algebraic structures, such as groups, rings and fields make use of binary operations. For this reason, going forward it is implicitly understood that by operation over an algebraic structure, we mean a binary operation, unless specified otherwise.

2.2.1 Groups, Rings and Fields

Now that we have familiarized ourselves with algebraic structures, we will define group, ring and field, which are the building blocks of homomorphic encryption. We begin by defining a group.

Definition 2.5 (Group)**.**
A *group* is an algebraic structure defined over the set G with a binary operation $\circ$ that satisfies the following axioms $(a, b, c \in G)$:

(1) **Associativity:**
$$(a \circ b) \circ c = a \circ (b \circ c). \tag{2.7}$$

(2) **Identity element:** There exists a unique element $e \in G$, such that for any $a \in G$, we have
$$a \circ e = e \circ a = a. \tag{2.8}$$
The element e is called the *identity* element of G.

(3) **Inverse element:** For each element $a \in G$, there exists an element $a^{-1} \in G$ such that
$$a \circ a^{-1} = a^{-1} \circ a = e. \tag{2.9}$$
The element a^{-1} is called the *inverse* of a. a^{-1} is unique.

Example 2.2. The set of integers $\mathbb{Z}$ equipped with the usual addition operation $+$ forms a group $(\mathbb{Z}, +)$. In this group, 0 is the identity element, and for any integer $a \in \mathbb{Z}$, $-a$ is its corresponding inverse element.

We briefly define some group theoretic concepts that will be useful later, particularly in the context of the Clifford scheme in Section 5.3. Consider a group G with binary operation $\circ$. A subset H of G is called a *subgroup*, if $(H, \circ)$ forms a group. The symbol $\leqslant$ is used to denote a subgroup relationship. Hence, $H \leqslant G$ means that "H is a subgroup of G" . The *normalizer* of a subgroup H in G is defined by
$$N_G(H) = \{x \in G : x \circ H \circ x^{-1} = H\}. \tag{2.10}$$
An important property of elements in $N_G(H)$ is that they conjugate the members of the subgroup H. That is, if $x \in N_G(H)$ and $a \in H$, then $x \circ a \circ x^{-1} \coloneqq b \in H$. It is worth noting that a does not necessarily equal b. Nonetheless, the conjugation property ensures that b remains within H. Furthermore, The normalizer

$N_G(H)$ forms a subgroup in G, and H is a subgroup of $N_G(H)$. Therefore, we have $H \leqslant N_G(H) \leqslant G$.

Lastly, we say that a subset S of G is a *generator* of G, if the elements of G can be constructed as a combination of elements from S and their inverses under the group operation. We denote this by $\langle S \rangle = G$.

We now turn our attention to rings, another important algebraic structure. Unlike groups, which are characterized by a single binary operation, rings incorporate two binary operations: addition and multiplication. These operations distinguish rings from groups[4].

Definition 2.6 (Ring)**.**
A *ring* $(R, +, \cdot)$ is a tuple consisting of a set R and two binary operations, namely addition $+$ and multiplication $\cdot$ that satisfy the following axioms:

(1) **Commutativity:**

$$\forall a, b \in R, a + b = b + a. \tag{2.11}$$

(2) **Associativity of $+$:**

$$\forall a, b, c \in R, (a + b) + c = a + (b + c). \tag{2.12}$$

(3) **Additive Identity:**

$$\exists 0 \in R \text{ such that } \forall a \in R, a + 0 = a. \tag{2.13}$$

(4) **Additive Inverse:**

$$\forall a \in R, \exists a' \in R, \text{ such that } a + a' = 0. \tag{2.14}$$

(5) **Associativity of $\cdot$:**

$$\forall a, b, c \in R, (a \cdot b) \cdot c = a \cdot (b \cdot c). \tag{2.15}$$

(6) **Multiplicative Identity:**

$$\exists 1 \in R \text{ such that } \forall a \in R, a \cdot 1 = 1 \cdot 1 = a. \tag{2.16}$$

(7) **Distribution of Multiplication over Addition:**

$\forall a, b, c \in R$

(a) $a \cdot (b + c) = a \cdot b + a \cdot c,$ (2.17)

(b) $(b + c) \cdot a = b \cdot a + c \cdot a.$ (2.18)

[4]It is worth noting that our definition of a ring will incorporate the existence of a multiplicative identity. However, there is no unanimous consensus on this matter in the mathematical literature. For instance, [Noe21] introduced a ring without requiring a multiplicative identity, while [Jac85] includes the multiplicative identity and reserves a distinct term "rng", for the structures which are not assumed to have multiplicative unit.

Example 2.3. The tuple $(\mathbb{Z}, +, \cdot)$ with the ordinary operations of addition and multiplication forms a ring, where 1 is the multiplicative identity.

It can be shown that the additive identity, the multiplicative identity and the additive inverse of every element of a ring R are unique. While the existence of an additive inverse is a requirement for a ring, the existence of a multiplicative inverse is not. That is, for an element a in the ring R, it is not required that there exists an element $a^{-1} \in R$ satisfying the property $a \cdot a^{-1} = 1$. Furthermore, the multiplication operation within a ring need not be commutative; there is no requirement that for elements $a, b \in R$, $a \cdot b = b \cdot a$. A ring that satisfies the property of commutativity of multiplication and the existence of multiplicative inverse for each element is called a field, which we are going to formally define next.

Definition 2.7 (Field)**.**
A *field* $(\mathbb{F}, +, \cdot)$ is a tuple consisting of a set $\mathbb{F}$ and two binary operations, where addition $+$ and multiplication $\cdot$ satisfy the ring axioms, with the additional requirement that multiplication also satisfies the following axioms:

(1) **Commutativity of Multiplication:**

$$\forall a, b \in \mathbb{F}, a \cdot b = b \cdot a. \tag{2.19}$$

(2) **Multiplicative Identity:**

$$\forall a \neq 0_{\mathbb{F}} \in \mathbb{F}, \exists b \text{ such that } a \cdot b = b \cdot a = 1_{\mathbb{F}}. \tag{2.20}$$

Example 2.4. An example of a field is the finite field with two elements, denoted by $\mathbb{F}_2$. This field is the smallest field in terms of the number of elements. $\mathbb{F}_2$ is particularly important in this work, especially in the Section 5 on quantum cryptography. We formally define the addition and multiplication operations for this field in Tables (1) and (2), respectively.

$\oplus$	0	1
0	0	1
1	1	0

Table 1: Addition in $\mathbb{F}_2$

$\times$	0	1
0	0	0
1	0	1

Table 2: Multiplication in $\mathbb{F}_2$

2.2.2 Homomorphism

Homomorphism is a type of a mathematical function that preserves operation between two algebraic structures, such as groups and fields. Indeed, the word homomorphism is made up of Greek words *homo-*, meaning "same" and *-morphism*, which is used to form nouns with the sense of "a transformation or correspondence of the kind specified by the first element". Therefore, homomorphism can be translated as "a transformation of one set into another

that preserves in the second set the operations or relations between the elements of the first" [Oxf24d]. Now, we define homomorphism formally. Here we provide a general definition of homomorphism for algebraic structures equipped with n operations. For the purposes of this work, we are interested in algebraic structures equipped with 1 and 2 operations.

Definition 2.8 (Homomorphism)**.**
Let $(A, \circ_{A_1}, \circ_{A_2}, \ldots, \circ_{A_n})$ and $(B, \circ_{B_1}, \circ_{B_2}, \ldots, \circ_{B_n})$ be two algebraic structures of the same type (such as two groups), each equipped with n binary operations. Then a function $f : A \to B$ is called a *homomorphism* if for all $a, b \in A$ and for each i where $1 \leq i \leq n$:

$$f(a \circ_{A_i} b) = f(a) \circ_{B_i} f(b). \tag{2.21}$$

Following Definition 2.8, one can easily work out definitions for group homomorphisms and ring homomorphisms with careful consideration. For example, a group homomorphism must map the identity elements between the two groups, as well as mapping the additive inverses. Similarly, a ring homomorphism must map the multiplicative identity between two rings.

We recall that our focus in this work is on quantum homomorphic encryption. At this stage, we have covered the essential components related to the term 'homomorphic' in this context. We now shift our attention to developing the mathematical tools for the quantum part of our work.

2.3 Linear Algebra

In this section, we introduce the concept of a Hilbert space, which provides the mathematical framework to reason about the space that quantum states occupy and their relations with each other spatially. We will define its components, including vector space, inner product, norm and completeness of a space. Following this, we introduce other essential mathematical tools, such as tensor products and unitary transformations, which are frequently used in quantum mechanics.

2.3.1 Vector Space

We begin by defining a vector space. Sections 2.3.1-2.3.4 follow from [Lan86, DM99].

Definition 2.9 (Vector Space)**.**
Let $\mathbb{F}$ be a field. A non-empty set V is called a *vector space*, if it is equipped with the following two operations:

(1) *Vector Addition:* a mapping

$$V \times V \mapsto V \tag{2.22}$$

defined by

$$(\boldsymbol{u}, \boldsymbol{v}) \mapsto \boldsymbol{u} + \boldsymbol{v}, \tag{2.23}$$

(2) *Scalar Multiplication:* a mapping

$$\mathbb{F} \times V \mapsto V \tag{2.24}$$

defined by

$$(a, \boldsymbol{u}) \mapsto a\boldsymbol{u} \tag{2.25}$$

such that it satisfies the following 8 axioms:

VS1. Given the elements $\boldsymbol{u}, \boldsymbol{v}$ and $\boldsymbol{w}$ of V, we have:

$$\boldsymbol{u} + (\boldsymbol{v} + \boldsymbol{w}) = (\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{w}. \tag{2.26}$$

VS2. For all elements $\boldsymbol{u} + \boldsymbol{v}$ of V

$$\boldsymbol{u} + \boldsymbol{v} = \boldsymbol{v} + \boldsymbol{u}. \tag{2.27}$$

VS3. There exists an element $\mathbf{0} \in V$, called the zero vector, such that

$$\boldsymbol{v} + \mathbf{0} = \boldsymbol{v} \tag{2.28}$$

for all $\boldsymbol{v} \in V$.

VS4. Given an element $\boldsymbol{u} \in V$, the element $-\boldsymbol{v}$ is such that

$$\boldsymbol{v} + (-\boldsymbol{v}) = \mathbf{0}. \tag{2.29}$$

VS5. If $a, b \in \mathbb{F}$ and $\boldsymbol{u} \in V$, then

$$a(b\boldsymbol{v}) = (ab)\boldsymbol{v}. \tag{2.30}$$

VS6. For all elements $\boldsymbol{u}$ of V, we have

$$1\boldsymbol{v} = \boldsymbol{v} \tag{2.31}$$

where 1 denotes the multiplicative identity in $\mathbb{F}$.

VS7. If $a \in \mathbb{F}$, then

$$a(\boldsymbol{u} + \boldsymbol{v}) = a\boldsymbol{u} + a\boldsymbol{v} \tag{2.32}$$

for all elements $\boldsymbol{u}$ and $\boldsymbol{v}$ of V.

VS8. For $a, b \in \mathbb{F}$ and all elements $\boldsymbol{u}, \boldsymbol{v} \in V$

$$(a + b)\boldsymbol{v} = a\boldsymbol{v} + b\boldsymbol{v}. \tag{2.33}$$

Elements of V are called vectors. If $\mathbb{F} = \mathbb{R}$, then V is called a *real vector space*, and if $\mathbb{F} = \mathbb{C}$, then V is called a *complex vector space*.

We briefly introduce some key terminologies related to a vector space. Given a set of vectors $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n\}$, any vector of the form

$$\boldsymbol{v} = a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2 + \cdots + a_n\boldsymbol{v}_n \tag{2.34}$$

for some scalars $a_1, a_2, \ldots, a_n \in \mathbb{F}$ is called a *linear combination* of $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n\}$. The said set is called *linearly independent* if $\sum_{i=1}^{n} a_i\boldsymbol{v}_i = 0$ only when $a_1 = a_2 = \cdots = a_n = 0$. Otherwise, the set is called *linearly dependent*. A *subspace* S of the vector space V is a non-empty subset of V that is closed under addition of vectors and scalar multiplication. The *span* of a set of vectors is defined as the set of all linear combinations of the vectors from that set. A minimal spanning set of a vector space is called a *basis*. The cardinality of basis of a vector space V is called the *dimension* of V, denoted by $\dim V$. Throughout this work, we assume that the dimension of V is finite, that is $\dim V < \infty$.

2.3.2 Inner Products and Norms

The mathematical framework of quantum mechanics must include the necessary tools to facilitate the calculation of probabilities and the determination of orthogonality of states, among other tasks. Quantum particles may exist in multiple states at once, also known as "superposition", and the outcome of a measurement collapses the wavefunction to a definite state, making the measurement outcomes subject to the axioms of probability. More precisely, the sum of the squared absolute value of all possible measurement outcomes must equal 1. Therefore, it is desirable to equip the mathematical framework with notions of distance and normalization to ensure the law of total probability is respected. Consequently, the vector spaces in quantum mechanics are endowed with inner product and normed structures to provide the necessary tools for these calculations and properties. We are going to define inner product space and normed vector spaces next.

Definition 2.10 (Inner Product Space)**.**
Let V be a complex vector space, that is V is a vector space that is defined over the field $\mathbb{C}$. A mapping

$$\langle \cdot, \cdot \rangle : V \times V \mapsto \mathbb{C}$$

is called an *inner product* in V if for any $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in V$ and $\alpha, \beta \in \mathbb{C}$ the following conditions are satisfied:

IP1. Conjugate symmetry:

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \overline{\langle \boldsymbol{v}, \boldsymbol{u} \rangle} \tag{2.35}$$

where $\bar{\cdot}$ denotes the complex conjugate of a complex number.

IP2. Linearity in the second argument:

$$\langle \boldsymbol{u}, \alpha\boldsymbol{v} + \beta\boldsymbol{w} \rangle = \alpha\langle \boldsymbol{u}, \boldsymbol{v} \rangle + \beta\langle \boldsymbol{u}, \boldsymbol{w} \rangle. \tag{2.36}$$

IP3. Positive-definiteness:

$$\langle \boldsymbol{u}, \boldsymbol{u} \rangle \geq 0 \text{ with equality if and only if } \boldsymbol{u} = 0. \tag{2.37}$$

A vector space with an inner product space is called a *inner product space*.

We note that in physics and quantum mechanics the inner product is typically defined to be linear in the second argument and conjugate linearly in the first argument. That is, $\langle \alpha \boldsymbol{u} + \beta \boldsymbol{v}, \boldsymbol{w} \rangle = \overline{\alpha} \langle \boldsymbol{u}, \boldsymbol{w} \rangle + \overline{\beta} \langle \boldsymbol{v}, \boldsymbol{w} \rangle$. This definition is aligned with the presentation in [NC10]; however, in mathematical contexts the convention is to define linearity in the first argument as in [DM99].

Example 2.5. An example of inner product space is the set of the complex numbers $\mathbb{C}$, equipped with an inner product defined by $\langle x, y \rangle = \overline{x} y$.

Generally speaking, given a vector space it is not meaningful to ask questions about the length or magnitude of a vector in that space. The concept of norm provides the tools to generalize the notion of length in a meaningful manner.

Definition 2.11 (Normed Vector Space)**.**
A function from a vector space V into $\mathbb{R}$:

$$\|\cdot\| : V \mapsto \mathbb{R} \tag{2.38}$$

defined by

$$\|\cdot\| : \boldsymbol{x} \mapsto \|\boldsymbol{x}\| \tag{2.39}$$

is called a *norm* if it satisfies the following conditions:

NV1. It is non-negative, meaning that $\|\boldsymbol{x}\| \geq 0$ for every vector $\boldsymbol{x}$.

NV2. It is positive on nonzero vectors, that is

$$\|\boldsymbol{x}\| = 0 \text{ implies } \boldsymbol{x} = 0 \tag{2.40}$$

NV3. For every vector $\boldsymbol{x}$, and every scalar α

$$\|\alpha \boldsymbol{x}\| = |\alpha| \|\boldsymbol{x}\| \tag{2.41}$$

NV4. The triangle inequality holds; that is, for every vectors $\boldsymbol{x}$ and $\boldsymbol{y}$:

$$\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\| \tag{2.42}$$

A vector space with a norm is called a *normed vector space*.

An inner product induces a norm in a natural way. Suppose, $\langle \cdot, \cdot \rangle$ denotes an inner product in some vector space. Then by a theorem, $\|\cdot\| \coloneqq \sqrt{\langle \cdot, \cdot \rangle}$ defines a norm.

Example 2.6. The function defined by $\|\boldsymbol{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$ for $\boldsymbol{x} = (x_1, x_2, \cdots, x_n) \in \mathbb{R}^n$ is a norm in $\mathbb{R}^n$. This norm is known as *Euclidean norm.*

Without delving into details, we remark that a normed vector space does not induce an inner product. Therefore, every inner product space induces a normed vector space, but not the other way around.

With the definitions of an inner product space and norm in place, we can now define what we mean by an orthonormal basis. A vector $\boldsymbol{v}$ in an inner product space V is called to be a *unit vector* if $\|\boldsymbol{v}\| = \sqrt{\langle \boldsymbol{v}, \boldsymbol{v} \rangle} = 1$. Furthermore, two vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ of V are *orthogonal*, if $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle = 0$. An *orthonormal basis* for an inner product space V is a basis whose vectors are orthonormal, meaning that the vectors are unit vectors and orthogonal to each other.

2.3.3 Complete Space

Before defining Hilbert spaces, we need to equip our vector space with another important property, known as completeness of the space. Specifically, we want any convergent sequence of elements within our vector space to converge to an element that also lies within the same vector space, as opposed to an element that may lie outside of it. This property is vital in quantum mechanics as it ensures that a convergent sequence of quantum states converge to another quantum state and similarly a sequence of convergent quantum operators converge to another quantum operator.

To define a complete vector space, we first need to introduce the concept of a Cauchy sequence.

Definition 2.12 (Cauchy Sequence)**.**
A sequence of vectors $(\boldsymbol{x_n})$ in a normed space is called a *Cauchy Sequence*,

$$\text{if } \forall \epsilon > 0, \exists M_\epsilon > 0 \text{ such that } \|\boldsymbol{x_m} - \boldsymbol{x_n}\| < \epsilon \text{ for all } m_{\epsilon,M}, n_{\epsilon,M} > M_\epsilon.$$

Definition 2.13 (Complete Vector Space)**.**
A vector space V is called a *complete vector space* if every Cauchy sequence in V converges to an element of V.

One might wonder if an inner product space is also a complete space; however, we remark without providing the details that there are inner product spaces that are not complete! Therefore, it is necessary to equip the mathematical framework in the context of quantum mechanics with complete vector space.

2.3.4 Hilbert Space

With inner product space, normed vector space and complete space defined, we now have all the necessary ingredients to describe a Hilbert space.

Definition 2.14 (Hilbert Space)**.**
A complete vector space $\mathcal{H}$ with an inner product $\langle \cdot, \cdot \rangle$, which induces the norm defined by $\|\cdot\| = \sqrt{\langle \cdot, \cdot \rangle}$ is called a *Hilbert Space.*

It is worth noting that Hilbert spaces are not exclusive to quantum mechanics. The definition of an inner product is context-dependent, and depending on the applications, one could define a different inner product for their Hilbert space. With this in mind, in the current context, we specifically define the appropriate inner product used in this quantum mechanical Hilbert space. Going forward, when "Hilbert space" is mentioned, we mean the Hilbert space as in Definition 2.15. We will denote it using the script letter $\mathcal{H}$.

Definition 2.15 (Hilbert Space for Quantum Mechanics)**.**
Let $\mathcal{H}$ denote a complete complex vector space of dimension n, equipped with an inner product defined by

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \sum_{i=1}^{n} \overline{u_i} \cdot v_i \tag{2.43}$$

where u_i and v_i denote the i^{th} coordinate of $\boldsymbol{u}$ and $\boldsymbol{v}$, respectively.

Furthermore, let $\mathcal{H}$ be equipped with the norm $\|\boldsymbol{u}\| = \sqrt{\langle \boldsymbol{u}, \boldsymbol{u} \rangle}$ induced by this inner product. $\mathcal{H}$ is the Hilbert space used in quantum mechanics.

To summarize, a Hilbert space is a vector space along with additional structures such as completeness of space, inner product and norm.

2.3.5 Tensor Products

The tensor product provides the mathematical tools for constructing larger quantum systems, which is crucial for describing multi-particle systems. More formal constructions of tensor products can be found in [Wat18, NC10].

Definition 2.16 (Tensor Product of Matrices)**.**
Let A be an $m \times n$ matrix and B be a $p \times q$ matrix:

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & \dots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{q1} & \dots & b_{qp} \end{bmatrix}, \tag{2.44}$$

then the tensor product of A and B, denoted by $A \otimes B$ is defined by

$$A \otimes B = \begin{bmatrix} a_{11}b_{11} & \dots & a_{11}b_{1q} & \dots & \dots & a_{1n}b_{11} & \dots & a_{1n}b_{1q} \\ \vdots & \ddots & \vdots & \dots & \dots & \vdots & \ddots & \vdots \\ a_{11}b_{p1} & \dots & a_{11}b_{pq} & \dots & \dots & a_{1n}b_{p1} & \dots & a_{1n}b_{pq} \\ \vdots & & \vdots & \ddots & & \vdots & & \vdots \\ \vdots & & \vdots & & \ddots & \vdots & & \vdots \\ a_{m1}b_{11} & \dots & a_{m1}b_{1q} & \dots & \dots & a_{mn}b_{11} & \dots & a_{mn}b_{1q} \\ \vdots & \ddots & \vdots & \dots & \dots & \vdots & \ddots & \vdots \\ a_{m1}b_{p1} & \dots & a_{m1}b_{pq} & \dots & \dots & a_{mn}b_{p1} & \dots & a_{mn}b_{pq} \end{bmatrix} \tag{2.45}$$

or more compactly in the block matrix

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix}. \tag{2.46}$$

After introducing the appropriate concepts and notation in quantum mechanics, we will discuss the properties of tensor products relevant to this work in Section 3.1.4.

2.3.6 Unitary Transformations

Unitary transformations are matrices that have an important property, namely preserving the norm of a vector. This is specially important in quantum mechanics, as the evolution of a closed quantum system is described by a unitary transformation. More details will be given in Section 3.1.2. For now, we define a unitary transformation. Before doing so, we remark that for a matrix U, $U^\dagger$ denotes the *conjugate transpose* of U. In other words, if

$$U = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \tag{2.47}$$

then

$$U^\dagger = \begin{bmatrix} \overline{a_{11}} & \dots & \overline{a_{1m}} \\ \vdots & \ddots & \vdots \\ \overline{a_{1n}} & \dots & \overline{a_{mn}} \end{bmatrix}. \tag{2.48}$$

Definition 2.17 (Unitary Matrix)**.**
A complex $n \times n$ matrix U is *unitary* if its conjugate transpose is equal to its inverse. That is, if

$$U^\dagger U = UU^\dagger = UU^{-1} = I \tag{2.49}$$

where I is the identity matrix, then U is a unitary matrix. In other words, $U^\dagger = U^{-1}$.

In $\mathcal{H}$, we have $\langle U\boldsymbol{u}, U\boldsymbol{v}\rangle = (U\boldsymbol{u})^\dagger U\boldsymbol{v} = \overline{\boldsymbol{u}}\underbrace{U^\dagger U}_{=I}\boldsymbol{v} = \overline{\boldsymbol{u}}\boldsymbol{v} = \langle \boldsymbol{u}, \boldsymbol{v}\rangle$, which shows that a unitary transformation preserves the inner product.

We remark that unitary matrices coupled with matrix multiplication form a group, which will be denoted by $\mathcal{U}_n$, where n is the dimension of the Hilbert space $\mathcal{H}$.

2.3.7 Hermitian Matrix

In this section, we are going to introduce another important and useful family of matrices in the realm of quantum computing known as Hermitian matrices, along with their subset, positive semi-definite matrices. We will conclude by discussing their motivation and applications within the field of quantum computing.

Definition 2.18 (Hermitian Matrix)**.**
A square complex matrix H is said to be *Hermitian matrix* or self-adjoint, if $H = H^\dagger$.

Note that Hermitian matrices and unitary matrices are not equivalent, although their definitions are very similar. A unitary matrix U demands $U^{-1} = U^\dagger$, while a Hermitian matrix H requires $H = H^\dagger$. In the next example, we provide instances of unitaries that are not Hermitian and Hermitian matrices that are not unitary.

Example 2.7.

(1) There are unitary matrices that are not Hermitian.

Consider the matrix U:

$$U = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \Rightarrow U^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} \tag{2.50}$$

$$\Rightarrow UU^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} \tag{2.51}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I \tag{2.52}$$

Therefore, U is unitary.

However, $U \neq U^\dagger$, and consequently, U is not a Hermitian matrix.

(2) There are Hermitian matrices that are not unitary.

Consider the matrix H:

$$H = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} = H^\dagger \tag{2.53}$$

However

$$\det(H) = 3 \Rightarrow H^{-1} = \frac{1}{\det(H)}\text{adj}(H) = \frac{1}{3}\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} \end{bmatrix} \tag{2.54}$$

Clearly, $H^\dagger \neq H^{-1}$, and therefore, H is not a unitary matrix, while H is a Hermitian matrix.

2.3.7.1 Positive Semi-Definite Matrix

Hermitian matrices have an important subclass, known as positive semi-definite matrices.

Definition 2.19 (Positive Semi-Definite Matrix)**.**
Let H be an $n \times n$ Hermitian matrix and let $\boldsymbol{u}$ be any n-dimensional vector in Hilbert space. Then the matrix H is said to be *positive semi-definite* if,

$$\boldsymbol{u}^\dagger H \boldsymbol{u} \geq 0. \tag{2.55}$$

In this case, we write $H \succeq 0$.

An important property of a positive semi-definite matrix $H \in \mathbb{C}^{n \times n}$ is that it has non-negative eigenvalues λ_i, and H can be represented by

$$H = \sum_{i=1}^{n} \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^\dagger \tag{2.56}$$

where each $\boldsymbol{u}_i$ is the eigenvector associated with the eigenvector λ_i. The non-negativity of the eigenvalues λ_i justifies the notation $H \succeq 0$, indicating that all the eigenvalues of H are non-negative.

Looking ahead, the representation of H in Equation 2.56 is particularly valuable in the context of quantum mechanics. As we will discuss in Section 3.3, positive semi-definite matrices provide a representation of quantum states. Additionally, we will see that certain quantum gates that are unitary are also Hermitian, notably Pauli matrices. It will be shown that that Pauli matrices form a basis for Hermitian matrices, a property that we will rely on to prove the security of quantum one-time pad in Section 5.1.

3 Quantum Information Science

In this section, we present relevant terms and concepts in the context of quantum information science (QIS) . In Section 3.1, we begin by introducing the postulates of quantum theory. We then proceed to Section 3.2, where we define various quantum gates and gradually build up to quantum circuits. In Section 3.3, we introduce the density matrix formalism, which facilitates the description of various quantum states, such as representing a maximally mixed state. In Section 3.4, we discuss the concepts of partial measurement and partial trace. Finally, in Section 3.5, we introduce the concepts of quantum entanglement and quantum teleportation, which do not have classical counterparts. These two concepts are an integral part of the EPR scheme, and they ensure the correctness of the decryption procedure.

3.1 Postulates of Quantum Theory

Quantum mechanics is used to explain physical phenomena that cannot be explained by classical physics, although classical and quantum physics are closely connected. Quantum mechanics exhibits classical behaviour in the large-scale limits, as formalized by the correspondence principle. Like classical physics, quantum mechanics is governed by a set of laws, which physicists tend to call them postulates, what mathematicians typically refer to them as axioms. As we have developed the necessary mathematical tools, we can introduce the postulates of quantum theory. The formulations of the postulates are adapted from the widely used source in QIS by Nielsen and Chuang [NC10].

3.1.1 Qubit

The first postulate of quantum mechanics specifies the space in which quantum state lie and the constraints on them, particularly that they must be unit vectors. This postulate justifies the use of Hilbert space $\mathcal{H}$ as the appropriate vector space used for quantum mechanics.

> **Postulate 1**: Associated to any isolated physical system is a complex vector space with inner product (that is, a Hilbert space) known as the *state space* of the system. The system is completely described by its *state vector*, which is a unit vector in the system's state space.

We can now define the term quantum bit or "qubit" for short, which is the counterpart of a bit from classical information science. A qubit is the fundamental unit of quantum information.

Definition 3.1 (Qubit)**.**
A *qubit* is a two-dimensional quantum system having length 1.

Using the language of linear algebra, a qubit can be described by an orthonormal basis, such as $\left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$. That is, a qubit, as vector a $\boldsymbol{v}$, can be

represented by

$$\boldsymbol{v} = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{3.1}$$

with the constraint $|\alpha|^2 + |\beta|^2 = 1$, to ensure the normalization condition is respected. Furthermore, the complex numbers α and β are called the *probability amplitudes*. As it will soon become evident, representing a qubit with vector notation $[\cdot]$ becomes cumbersome, especially when describing larger quantum states, which will be introduced later. For that reason, physicists adopt the *bra-ket Dirac notation*. With this notation, the vector $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ is represented by the *ket* $|0\rangle$. In other words, $|0\rangle \coloneqq \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. Similarly, $|1\rangle \coloneqq \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. The two states $|0\rangle$ and $|1\rangle$ are called the zero state and the one state, respectively. A *bra*, denoted by $\langle\cdot|$, represents the conjugate transpose of $|\cdot\rangle$. Therefore, $\langle 0| = \begin{bmatrix} 1 & 0 \end{bmatrix}$ and $\langle 1| = \begin{bmatrix} 0 & 1 \end{bmatrix}$. With this in mind, Equation 3.1 can be compactly be represented by

$$\alpha\,|0\rangle + \beta\,|1\rangle\,. \tag{3.2}$$

It is common to denote quantum states using Greek letters, such as $|\psi\rangle$ and $|\phi\rangle$.

By a result from linear algebra, the choice of a basis vector and similarly an orthonormal basis is not unique. In QIS, the orthonormal basis $\{|0\rangle, |1\rangle\}$ is referred to as *computational basis*. We will introduce another frequently used basis in quantum mechanics, known as the *Hadamard basis* in Example 3.1.

Example 3.1. The Hadamard basis consists of the *plus state* $|+\rangle \coloneqq \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and the *minus state* $|-\rangle \coloneqq \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. To see why $\{|+\rangle, |-\rangle\}$ forms an orthonormal basis, we observe that

$$\sqrt{\langle +, + \rangle} \coloneqq \sqrt{\langle + | + \rangle} = \sqrt{\left(\frac{1}{\sqrt{2}}\right)^2 + \left(\frac{1}{\sqrt{2}}\right)^2} = 1. \tag{3.3}$$

By similar reasoning and the fact that $\left(-\frac{1}{\sqrt{2}}\right)^2 = \left(\frac{1}{\sqrt{2}}\right)^2$, it follows that $\sqrt{\langle - | - \rangle} = 1$. Lastly,

$$\sqrt{\langle + | - \rangle} = \sqrt{\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \times \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}} \tag{3.4}$$

$$= \sqrt{\frac{1}{2} - \frac{1}{2}} \tag{3.5}$$

$$= 0. \tag{3.6}$$

Therefore, we have shown that $\{|+\rangle, |-\rangle\}$ also forms an orthonormal basis, by showing that $|+\rangle$ and $|-\rangle$ are unit vectors that orthogonal to each other. □

Before stating the next postulate, it is worthwhile to highlight that we have already arrived at a point that distinguishes quantum information science from classical information in a significant way. A quantum state is described as a linear combination of the states $|0\rangle$ and $|1\rangle$, which is often called *superposition*. This is different from the classical setting, where a classical bit is either 0 or 1, but not both. Nonetheless, it is important to clarify what superposition entails and what it does not. A qubit $|\psi\rangle$ cannot not exist in the states $|0\rangle$ and $|1\rangle$ both with 100% probability. Instead, $|\psi\rangle$ can exist in a state that is a combination of $|0\rangle$ and $|1\rangle$, such that the sum of absolute squares of the corresponding amplitudes add up to 1. Consequently, while classical bits are confined to two discrete states, the probability amplitudes of qubits can span a continuous range within that spectrum.

3.1.2 Evolution of a Closed Quantum System

The next natural question that may come up is what operations are permissible on isolated quantum states. Due to postulate 1, we know that a qubit must be represented by a unit vector. Therefore, any operations on qubits must respect that restriction. That is, we seek matrices U such that $\|U|\psi\rangle\| = \||\psi\rangle\| = 1$. These matrices are known as unitary matrices, which we introduced in Section 2.3.6. This leads us to the second postulate of quantum mechanics.

> **Postulate 2:** The evolution of a *closed* quantum system is described by a *unitary transformation*. That is, the state $|\psi\rangle$ of the system at t_1 is related to the state $|\psi'\rangle$ of the system at time t_2 by a unitary operator U which depends only on times t_1 and t_2,
>
> $$|\psi'\rangle = U|\psi\rangle\,. \tag{3.7}$$

We can verify that $\|U|\psi\rangle\| = 1$, by noting that

$$\|U|\psi\rangle\| = \sqrt{(U|\psi\rangle)^\dagger (U|\psi\rangle)} = \sqrt{\langle\psi|\underbrace{U^\dagger U}_{I}|\psi\rangle} = \sqrt{\langle\psi|\psi\rangle} = 1. \tag{3.8}$$

3.1.3 Measurement

> **Postulate 3:** Quantum measurements are described by a collection $\{M_m\}$ of *measurement operators*. These operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is $|\psi\rangle$ immediately before the measurement then the probability that result m occurs is given by
>
> $$p(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle\,. \tag{3.9}$$

The state of the system after the measurement is

$$\frac{M_m |\psi\rangle}{\sqrt{\langle\psi| M_m^\dagger M_m |\psi\rangle}}. \tag{3.10}$$

The measurement operators satisfy the *completeness equation*,

$$\sum_m M_m^\dagger M_m = I. \tag{3.11}$$

The completeness equation expresses the fact that probabilities sum to one:

$$1 = \sum_m p(m) = \sum_m \langle\psi| M_m^\dagger M_m |\psi\rangle . \tag{3.12}$$

This postulate implies that the act of measurement causes the wavefunction to collapse. Note that M_m, a measurement operator, is not unitary. Hence, $M_m |\psi\rangle$ is not necessarily a valid quantum state, and that is why Equation 3.10 is normalized to ensure that the post-measurement state is a valid quantum state with norm 1. Furthermore, the non-unitary nature of the measurement operator M_m implies that measurement is not a reversible quantum operation, highlighting its part in bringing about the collapse of the wavefunction. The precise mechanism underlying this collapse is a topic of interpretation and debate within quantum theory, with various interpretations offering different perspectives on the nature of measurement and the collapse process. We refer the interested reader to [LD24, Vai21, Fay24] for reference.

Additionally, the non-unitary nature of a measurement does not contradict the second postulate, as the second postulate pertains to the evolution of a *closed* quantum system. When a quantum system interacts with its external environment, the second postulate does not necessarily apply. During a measurement, the quantum system interacts with its surroundings, and hence the measurement operator is non-unitary. It is worth mentioning that a measurement is not the only non-unitary transformation; for instance, noise from the environment can cause a non-unitary transformation upon a quantum state. The general transformation of a quantum state is described by quantum channels. An interested reader that wishes to learn more about quantum channels, which are *completely positive trace-preserving* maps, is invited to consult the relevant sources such as [NC10, Wat18].

Example 3.2. Let us compute the probability of measuring different outcomes in the computational basis for an arbitrary quantum state $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$. First, we define the measurement operators:

$$M_0 = |0\rangle\langle 0| = \begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}1 & 0\end{bmatrix} = \begin{bmatrix}1 & 0\\0 & 0\end{bmatrix}, \quad M_1 = |1\rangle\langle 1| = \begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}0 & 1\end{bmatrix} = \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}. \tag{3.13}$$

It is straightforward to verify that $M_0 + M_1 = I$, and hence this collection of measurement operators satisfies the completeness equation, given by Equation 3.11.

Next, we observe that

$$M_0 |\psi\rangle = (|0\rangle \langle 0|)(\alpha |0\rangle + \beta |1\rangle) \tag{3.14}$$
$$= \alpha |0\rangle (\langle 0| 0\rangle) + \beta |1\rangle (\langle 0| 1\rangle) \tag{3.15}$$
$$= \alpha |0\rangle \tag{3.16}$$

From Equations 3.14 and 3.16, we can conclude that $(M_0 |\psi\rangle)^\dagger = \langle 0| M_0^\dagger = \langle 0| \overline{\alpha}$. We can calculate the probability of measuring 0:

$$p(0) = \langle \psi| M_0^\dagger M_0 |0\rangle \tag{3.17}$$
$$= \langle 0| \overline{\alpha}\alpha |0\rangle \tag{3.18}$$
$$= |\alpha|^2 \langle 0|0\rangle \tag{3.19}$$
$$= |\alpha|^2. \tag{3.20}$$

Mutatis mutandis, it follows that $M_1 |\psi\rangle = \beta |1\rangle$ and $\langle \psi| M_1^\dagger = \langle 1| \overline{\beta}$. Again, by similar reasoning, it follows that $p(1) = |\beta|^2$.

We conclude that the probability outcome of performing a measurement in the same basis in which a qubit is expressed yields a value that is proportional to the square of the absolute value of the corresponding probability amplitude.

We have now developed the tools to introduce another aspect of qubits: their equivalency under a *global phase*. Consider two states $e^{i\theta} |\psi\rangle$ and $|\psi\rangle$, where θ is a real number. The quantity $e^{i\theta}$ is called the global phase factor. To understand their equivalency, let us compute the statistics of an arbitrary measurement outcome for both states. Let $p_{e^{i\theta}|\psi\rangle}(m)$ and $p_{|\psi\rangle}(m)$ denote the probability of measuring m for the states $e^{i\theta} |\psi\rangle$ and $|\psi\rangle$, respectively. Then we have:

$$p_{e^{i\theta}|\psi\rangle}(m) = \left(\langle e^{i\theta}\psi|\right) M^\dagger M \left(e^{i\theta} |\psi\rangle\right) \tag{3.21}$$
$$= \langle \psi| e^{-i\theta} M^\dagger M e^{i\theta} |\psi\rangle \tag{3.22}$$
$$= \langle \psi| M^\dagger M e^{i(\theta-\theta)} |\psi\rangle \tag{3.23}$$
$$= \langle \psi| M^\dagger M |\psi\rangle \tag{3.24}$$
$$= p_{|\psi\rangle}(m). \tag{3.25}$$

Therefore, we observe that the probability of an arbitrary outcome m is the same for both $e^{i\theta} |\psi\rangle$ and $|\psi\rangle$. This establishes an equivalency class between the two states, and we can write $e^{i\theta} |\psi\rangle \equiv |\psi\rangle$. It is common in the literature to use the less precise equal sign $=$ for this equivalency class, and we will adopt this convention going forward. For example, we can implicitly infer that $-|\psi\rangle$ is equivalent to $|\psi\rangle$ and write $-|\psi\rangle = |\psi\rangle$.

3.1.4 Composite Systems

In Section 2.3.5, we introduced tensor products, which provide the mathematical structure for the fourth postulate. The fourth postulate describes how to construct composite quantum states from the constituent subsystems, while using the tensor product formalism to describe it.

> **Postulate 4:** The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered 1 through n, and system number i is prepared in the state $|\psi_i\rangle$, then the joint state of the total system $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$.

Note that the composite quantum state

$$|\phi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle = |\psi_1\rangle\,|\psi_2\rangle = |\psi_1\psi_2\rangle\,, \tag{3.26}$$

must still follow the first postulate, ensuring that $\||\phi\rangle\| = 1$. We will show its correctness shortly. Before doing so, let us introduce the properties of tensor products that are relevant to this work. Let $|\psi_1\rangle\,, |\psi_2\rangle \in \mathcal{H}_1$ and $|\phi_1\rangle\,, |\phi_2\rangle \in \mathcal{H}_2$. Let s by any scalar. Then the following properties hold:

$$(1)\ \ s(|\psi_1\rangle \otimes |\phi_1\rangle) = (s\,|\psi_1\rangle) \otimes |\phi_1\rangle = |\psi_1\rangle \otimes (s\,|\phi_1\rangle). \tag{3.27}$$

$$(2)\ \ (|\psi_1\rangle + |\psi_2\rangle) \otimes |\phi_1\rangle = |\psi_1\rangle \otimes |\phi_1\rangle + |\psi_2\rangle \otimes |\phi_1\rangle\,. \tag{3.28}$$

$$(3)\ \ |\psi_1\rangle \otimes (|\phi_1\rangle + |\phi_2\rangle) = |\psi_1\rangle \otimes |\phi_1\rangle + |\psi_1\rangle \otimes |\phi_2\rangle\,. \tag{3.29}$$

Now, to see why $\||\phi\rangle\| = \||\psi_1\rangle \otimes |\psi_2\rangle\| = 1$, note that if $|\psi_1\rangle = \alpha_1\,|0\rangle + \beta_1\,|1\rangle \in \mathcal{H}_1$ and $|\psi_2\rangle = \alpha_2\,|0\rangle + \beta_2\,|1\rangle \in \mathcal{H}_2$, then we have:

$$|\psi_1\rangle \otimes |\psi_2\rangle = (\alpha_1\,|0\rangle + \beta_1\,|1\rangle) \otimes (\alpha_2\,|0\rangle + \beta_2\,|1\rangle) \tag{3.30}$$

$$= \alpha_1\alpha_2\,|00\rangle + \alpha_1\beta_2\,|01\rangle + \beta_1\alpha_2\,|10\rangle + \beta_1\beta_2\,|11\rangle\,. \tag{3.31}$$

Next we observe that,

$$|\alpha_1\alpha_2|^2 + |\alpha_1\beta_2|^2 + |\beta_1\alpha_2|^2 + |\beta_1\beta_2|^2 \tag{3.32}$$

$$= |\alpha_1|^2 \cdot \left(\underbrace{|\alpha_2|^2 + |\beta_2|^2}_{=1}\right) + |\beta_1|^2 \cdot \left(\underbrace{|\alpha_2|^2 + |\beta_2|^2}_{=1}\right) \tag{3.33}$$

$$= |\alpha_1|^2 + |\beta_1|^2 = 1. \tag{3.34}$$

A similar argument holds for larger quantum states $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$.

3.2 Quantum Gates

In this section, we formally define a quantum gate and introduce different families of important quantum gates, namely Pauli and Clifford gates. Finally, we will explain the concept of universal quantum circuits. This section follows from [Sch19] and [GS22].

As mentioned in the second postulate of quantum theory, the evolution of a closed system is described by a unitary transformation. The unitary transformations are linear transformations on the state space, and those are the transformations that are regarded as quantum gates.

Definition 3.2 (Quantum Gate)**.**
A quantum n-qubit gate is a unitary operator U, such that $U : \mathcal{H}^{\otimes n} \to \mathcal{H}^{\otimes n}$.

Note that Definition 3.2 excludes non-unitary operations from being quantum gates. Consequently, the measurement operator does not count as a quantum gate. While terms like "measurement gate" are frequently used in literature, we will opt for the broader term "measurement operator" to account for its non-unitary nature.

3.2.1 Pauli Gates

An important class of unitary matrices are called Pauli matrices, named after physicist Wolfgang Pauli. Pauli matrices introduce operations such as bit flip, phase flip and a combination of both. Additionally, Pauli matrices form a basis for the space of Hermitian matrices. This is useful because density matrices, which will be introduced in Section 3.3, are Hermitian. Coupled with the fact that density matrices provide a representation of qubits, this means that a qubit can be represented as a linear combination of Pauli matrices. This fact will be useful when proving the security of a quantum encryption scheme in Section 5.1.

Definition 3.3 (Pauli Gates)**.**

$$\sigma_1 = \sigma_x = \mathsf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \sigma_2 = \sigma_y = \mathsf{Y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$
$$\sigma_3 = \sigma_z = \mathsf{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

The X gate, also known as the quantum NOT gate, is the quantum analog of the classical bit flip. Therefore, in the computational basis, this means $\mathsf{X}|0\rangle = |1\rangle$ and $\mathsf{X}|1\rangle = |0\rangle$. The Z gate, which does not have a classical counterpart, introduces a phase flip to the one state $|1\rangle$ while leaving the zero state $|0\rangle$ unchanged. In other words, $\mathsf{Z}|0\rangle = |0\rangle$ and $\mathsf{Z}|1\rangle = -|1\rangle$. Lastly, the Y gate combines both a bit flip and a phase flip. It can be represented as $\mathsf{Y} = i\mathsf{X}\mathsf{Z}$. Thus, $\mathsf{Y}|0\rangle = i|1\rangle$ and $\mathsf{Y}|1\rangle = -i|0\rangle$. Pauli matrices along with the identity gate $\sigma_0 = \mathsf{I}$ form a group.

Definition 3.4 (Pauli Group)**.**
Let

$$\mathcal{W} = \{i^a \sigma_\alpha | a, \alpha \in \{0, 1, 2, 3\}\}. \tag{3.35}$$

Then the set $\mathcal{W}$ equipped with a matrix multiplication operator $\times$ forms a group. The group $(\mathcal{W}, \times)$ is called the *Pauli* group.

While $\mathcal{W}$ has 16 members, the set $S_{\text{Pauli}} = \{\sigma_1, \sigma_2, \sigma_3\}$ with only three elements is a generator of the Pauli group $\mathcal{W}$. Hence, we can write $\langle S_{\text{Pauli}} \rangle = \mathcal{W}$

The Pauli group can be used to construct n-fold Pauli group, also known as the Pauli group on n-qubits, $\mathcal{W}_n$. $\mathcal{W}_n$ is the set of all n-qubit tensor products of elements of $\mathcal{W}$.

3.2.2 Clifford Gates

In Section 2.2.1, we introduced the concept of a group normalizer. The normalizer group of Paulis are simply called the *Clifford group*.

Definition 3.5 (Clifford Group)**.**
Let $\mathcal{W}_n$ denote the Pauli group acting on n qubits. The *Clifford group on n qubits*, is the group that normalizes $\mathcal{W}_n$. That is,

$$\mathcal{K}_n = \{\mathsf{K} : \mathsf{K}\,\mathsf{W}\,\mathsf{K}^\dagger \in \mathcal{W}_n, \forall\, \mathsf{W} \in \mathcal{W}_n\}. \tag{3.36}$$

Another equivalent way of characterizing a Clifford gate is to say for any Clifford gate K and any Pauli W, there exists W' such that $\mathsf{K}\,\mathsf{W} = \mathsf{W}'\,\mathsf{K}$. Furthermore, in the language of group theory developed in Section 2.2.1, we can write $\mathcal{W}_n \leqslant N_{\mathcal{U}_n}(\mathcal{W}_n) = \mathcal{K}_n \leqslant \mathcal{U}_n$.

While Definition 3.5 provides a description of Clifford gates and a method for verifying them, it does not define specific examples of Clifford gates. Here, we will briefly introduce some notable and important Clifford gates relevant to this work. These include the Hadamard gate $\mathsf{H} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, the phase gate[5] $\mathsf{P} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$ and the controlled-NOT gate $\mathsf{CNOT} = |0\rangle\langle 0| \otimes \mathsf{I} + |1\rangle\langle 1| \otimes \mathsf{X}$, which applies an X on the second register of a two-qubit system only if the first qubit is in the state $|1\rangle$. Another example of a two-qubit gate is the SWAP gate, defined by $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, which has the effect of swapping the order of qubits. That is, if $|\phi_1\rangle$ and $|\phi_2\rangle$ are single qubits in the computational basis, then $\mathsf{SWAP}(|\phi_1\rangle \otimes |\phi_2\rangle) = |\phi_2\rangle \otimes |\phi_1\rangle$.

[5] In some literature, such as [NC10], the phase gate is denoted by S instead of P. However, in this work, we follow the notation used in [BJ15].

In Example 3.3, we will verify that the Hadamard gate is a Clifford gate.

Example 3.3. The Hadamard gate is a Clifford gate, since $\mathsf{H}\mathsf{X} = \mathsf{Z}\mathsf{H}$, $\mathsf{H}\mathsf{Z} = \mathsf{X}\mathsf{H}$ and $\mathsf{H}\mathsf{Y} = -\mathsf{Y}\mathsf{H}$. These equalities only show that the H gate normalizes $S_{\text{Pauli}} = \{\mathsf{X}, \mathsf{Z}, \mathsf{Y}\}$. However, since $\langle S_{\text{Pauli}} \rangle = \mathcal{W}$, demonstrating that H is a normalizer of S_{Pauli} is sufficient to conclude that $\mathsf{H} \in \mathcal{K}$.

Lastly, before closing the section on Clifford gates, we highlight two important facts about them that motivate the structure of a quantum homomorphic encryption scheme. Firstly, the set $S_{\text{Cliff}} = \{\mathsf{H}, \mathsf{P}, \mathsf{CNOT}\}$ is not an exhaustive list of all Clifford gates; however, S_{Cliff} is of special interest, since it is a generator of all Clifford gates, as proven in [Got98b]. In the context of quantum homomorphic encryption scheme, this implies that when handling Clifford gates, a scheme only needs to account for the gates in S_{Cliff}, as any other Clifford gate can be decomposed into these gates. Secondly, another important and surprising fact is that Clifford gates can be efficiently simulated by classical computers. This result is known as the "Gottesman-Knill theorem" and a proof of this fact can be found in [Got98a]. Intuitively, this result suggests that handling Clifford gates in quantum homomorphic encryption may be more straightforward, and it might be advantageous to focus on these gates first before considering non-Clifford gates. We will elaborate more on this in Section 5.3.

3.2.3 Non-Clifford Gates

Up to this point we have introduced Pauli and Clifford gates; however, thanks to Gottesman-Knill theorem, we know that those gates are efficiently simulatable by classical computers, and if there were no other quantum gates, then it would have imposed a real threat to the motivation behind investing and studying quantum computing, as all of quantum computing would have been efficiently simulatable by classical computers. This is where non-Clifford gates come to the rescue of quantum computing[6]. A non-Clifford gate is any quantum gate that is not part of the Clifford group.

A notable and important non-Clifford gate is the T gate, which is defined as

$$\mathsf{T} = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix}. \tag{3.37}$$

A plethora of quantum algorithms, such as Shor's algorithm [Sho99] and Grover's search algorithm [Gro96], which are believed to outperform their best classical counterparts make use of non-Clifford gates in their implementations.

3.2.4 Quantum Circuit

Using quantum gates and measurement operators as fundamental building blocks, we can construct quantum circuits, which we will define next.

[6]Whether non-Clifford gates can be efficiently simulated by classical computers remains an open problem, with the widely held belief suggesting the answer is negative.

Definition 3.6 (Quantum Circuit)**.**
Let $\mathsf{L}_1, \mathsf{L}_2, \ldots, \mathsf{L}_d$ be a sequence of d operators acting on an n-qubit system, where each L_i consists of either unitary operators, measurement operators, or both. We define the *quantum circuit* C as the composition

$$\mathsf{C} = \mathsf{L}_d \circ \mathsf{L}_{d-1} \cdots \mathsf{L}_2 \circ \mathsf{L}_1 \tag{3.38}$$

constructed from the operators $\mathsf{L}_1, \mathsf{L}_2, \ldots, \mathsf{L}_d$. A quantum operator L_i is called a *layer*. To ensure that each layer L_i is an n-qubit operator, we assume all layers following a measurement operator include the identity gate on measured qubits to match the dimensionality. The *depth* of a circuit is the number of layers, d.

We note that in this definition of a quantum circuit, a measurement operator is included as an element of a quantum circuit, while it is not technically a quantum gate. The trade-off of including the measurement operator is that the resulting quantum circuits are not reversible according to Definition 3.6. Therefore, we reserve the term "quantum reversible circuit", when referring to quantum circuits where each layer consists only of quantum gates.

Quantum circuit diagrams are used as a visual representation of a quantum circuit, showing the sequence and types of quantum gates applied to qubits. Time progresses from left to right, illustrating the order in which operations are applied. A quantum wire, represented by horizontal lines, is associated with a qubit and illustrates how that qubit is affected throughout the circuit. Figure (1) shows a quantum circuit diagram and its various elements.

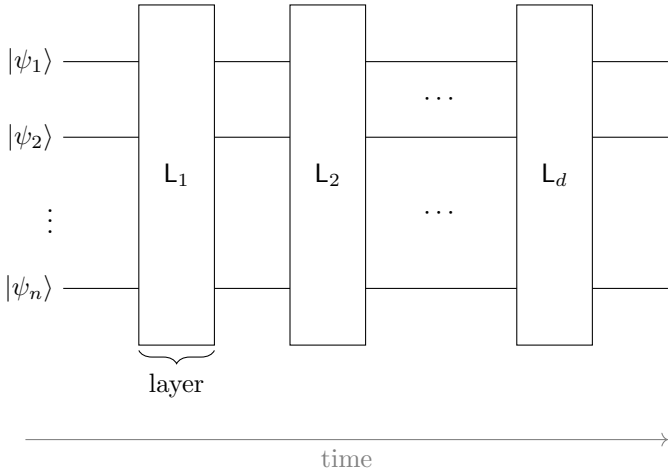


Figure 1: A quantum circuit diagram

We conclude this section with Table (3), which presents a compilation of frequently used quantum gates along with their circuit representation.

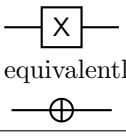
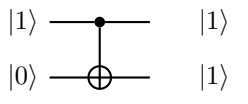
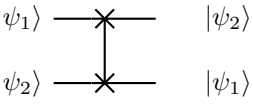
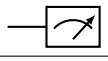
Gate	**# of Qubits**	**Matrix**	**Circuit Diagram**
X	1	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	 or equivalently
Y	1	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	
Z	1	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	
H	1	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	
P	1	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$	
T	1	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$	
CNOT	2	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	
SWAP	2	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	
Measurement	1 or more	A non-unitary operator	

Table 3: Frequently used quantum operators and their circuit representation

3.2.5 Universal Quantum Gate Set

A natural question to consider at this point is whether there are any other types of quantum gates to consider beyond Clifford and non-Clifford gates, and the answer is negative. To further elaborate on this topic, it is worthwhile to define a universal quantum gate set. A set S_{UQG} is called a *universal quantum gate set*, if S_{UQG}-gates can approximate any unitary acting on n qubits to any arbitrary precision. According to a fundamental result in quantum computing, any non-Clifford gate combined with a generator set of Clifford groups forms a universal quantum gate set [NC10]. For instance, $S_{\text{Cliff}} \cup \{\mathsf{T}\}$ forms a universal quantum gate set. This observation, coupled with the discussions made at the end of the Section 3.2.2, explains why [BJ15] first provides a scheme for Clifford gates and then introduces the "EPR scheme", which mainly differs from the Clifford scheme by including the T gate.

3.3 Density Matrix

So far we have used vectors to describe a quantum state $|\psi\rangle$. There is another equivalent representation of quantum states given by density operators or density matrices. This section follows from [NC10, Wol21].

Definition 3.7 (Density Matrix)**.**
A matrix ρ is called a *density matrix*, if ρ has the following properties:

(1) $\rho \succeq 0$, meaning that ρ is a positive semi-definite matrix, and also a Hermitian matrix.

(2) $\text{Tr}(\rho) = 1$.

The first property in Definition 3.7 ensures that a density matrix ρ has non-negative eigenvalues λ_i and the second condition ensures that the sum of the eigenvalues add up to 1, aligned with the fact that the sum of the squared absolute values of probability amplitudes of a quantum state add up to 1, making the density matrices suitable for representing a quantum state. Let us see how.

A quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ can be represented by a density matrix $\rho = |\psi\rangle\langle\psi| = \begin{bmatrix} |\alpha|^2 & \alpha\overline{\beta} \\ \overline{\alpha}\beta & |\beta|^2 \end{bmatrix}$. However, density matrices capture a broader class of quantum states that cannot be representable by a vector. To illustrate this, we will consider the following two scenarios:

(1) Alice applies the Hadamard gate H to the state $|0\rangle$ to obtain the state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. She sends the state $|+\rangle$ to Bob, who expects a 50-50 probability split in observing the states $|0\rangle$ and $|1\rangle$ upon measurement.

(2) Alice flips a fair coin. If the result is heads, she creates the state $|0\rangle$; if tails, she prepares the state $|1\rangle$. Let $|\phi\rangle$ represent the resulting state and suppose Bob knows how the state $|\phi\rangle$ is prepared, but he does not know the outcome of the coin flip. He still expects a 50-50 split in observing $|0\rangle$ and $|1\rangle$ upon measurement.

There is a key difference between the two scenarios. In the first scenario, the quantum state in Bob's view is the quantum state $|+\rangle$, which is in a superposition of the $|0\rangle$ and $|1\rangle$. In the second scenario, the quantum state according to Bob's view is a mixture of $|0\rangle$ and $|1\rangle$ with equal probability, which is not the same as the state $|+\rangle$. More formally, the state $|\phi\rangle$ exists as a classical statistical mixture of states $|0\rangle$ and $|1\rangle$ with equal probability, described by the ensemble $\{(p_1, |0\rangle), (p_2, |1\rangle)\}$, where $p_1 = p_2 = 0.5$. Using the language of density matrices, the state $|\psi\rangle$ can be written as $\frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$, which correctly captures Bob's expectation of measuring $|0\rangle$ or $|1\rangle$ with equal probability. Thus, density matrices allow us to represent quantum states that cannot be expressed as a single vector, in this case from Bob's point of view.

More abstractly, an *ensemble of quantum states* $\{(p_1, |\psi_1\rangle), \ldots, (p_k, |\psi_k\rangle)\}$ where each $|\psi_i\rangle$ is prepared with probability p_i is represented by a density matrix

$$\rho = \sum_{i=1}^{k} p_i |\psi_i\rangle\langle\psi_i| . \tag{3.39}$$

If $k = 1$, the state is precisely described and is called a *pure state*. Otherwise, if $k \geq 2$, then the quantum state is called a *mixed state*. A key property of the density matrix of a pure state is that $\rho^2 = \rho$, whereas this equality does not hold for mixed states.

Lastly, let us revisit the second scenario one more time, where Alice prepares her qubit by a coin flip. From Bob's point of view, the resulting state is a mixed state. However, since Alice knows the outcome of the coin flip, the resulting state is either definitely $|0\rangle$ or definitely $|1\rangle$. This is a crucial observation in quantum information theory. A pure quantum state from Alice's perspective may be a mixed quantum state to Bob. When proving the security of quantum cryptographic protocols, we would like to prove security from an adversary's point of view. Therefore, even if a quantum state is known to trusted parties, we are only concerned with how an adversary would view the state. By describing quantum states as density matrices, we can demonstrate the security of the scheme. That is another important reason for using density matrices to represent quantum states.

3.3.1 Quantum Operations on Density Matrix

We have seen the effects of applying a unitary gate and a measurement on a pure state when represented as vectors. The corresponding mathematical operations on density matrices are slightly different, and this section aims to clarify those differences.

Applying a unitary U to a quantum state $|\psi\rangle$ results in the state $U|\psi\rangle \rightarrow |\psi'\rangle$. Intuitively, one might expect that if ρ is the density matrix of $|\psi\rangle$, then $U\rho$ is equivalent to the density matrix of $|\psi'\rangle$. However, that is not the case. Indeed,

$$\rho' = |\psi'\rangle \langle\psi'| \tag{3.40}$$
$$= U |\psi\rangle (U |\psi\rangle)^\dagger \tag{3.41}$$
$$= U \underbrace{|\psi\rangle \langle\psi|}_{=\rho} U^\dagger \tag{3.42}$$
$$= U\rho U^\dagger \tag{3.43}$$

Therefore, applying U to ρ is equivalent to $U\rho U^\dagger$.

Furthermore, in Section 3.1.3, we observed that the post-measurement state of a pure state is given by Equation 3.10

$$\frac{M_m |\psi\rangle}{\sqrt{\langle\psi| M_m^\dagger M_m |\psi\rangle}},$$

where M_m is the corresponding measurement operator. The corresponding density operator, given that outcome m is observed, is given by:

$$\rho|_m = \frac{M_m \rho M_m^\dagger}{\mathrm{Tr}\left(M_m^\dagger M_m \rho\right)}. \tag{3.44}$$

Additionally, the probability of observing the outcome m is:

$$p(m) = \mathrm{Tr}\left(M_m^\dagger M \rho\right). \tag{3.45}$$

3.3.2 Maximally Mixed State

An important class of mixed states is a *maximally mixed state*, which is easily expressible in the density matrix formalism.

Definition 3.8 (Maximally Mixed State)**.**
A quantum state in a 2^n-dimensional Hilbert space that is an ensemble of orthogonal states $\{\frac{1}{2^n}, |\psi_m\rangle\}_{m=1}^{2^n}$ is called a *maximally mixed state*. The density operator corresponding to a maximally mixed state is given by:

$$\rho_{\max} = \frac{\mathsf{I}}{2^n} \tag{3.46}$$

where I is the identity matrix of dimension 2^n.

Example 3.4. A mixture of the states $|0\rangle$ and $|1\rangle$ or $|+\rangle$ and $|-\rangle$ weighted with equal probability is a maximally mixed state:

$$\frac{\mathsf{I}}{2} = \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|) = \frac{1}{2}(|+\rangle\langle +| + |-\rangle\langle -|) \tag{3.47}$$

A noteworthy feature of maximally mixed states is that measuring a maximally mixed state in any basis results in a uniform distribution of the basis elements. To see this, suppose we measure the maximally mixed state $\rho_{\max}$ in some orthonormal basis $\{|\psi_m\rangle\}_{m=1}^{2^n}$. The probability $p(m)$ of obtaining the outcome corresponding to the basis state $|\psi_m\rangle$ is given by: $p(m) = \mathrm{Tr}(M_m^\dagger M_m \rho_{\max})$, where $M_m = |\psi_m\rangle\langle\psi_m|$. Note that

$$M_m^\dagger M_m = (|\psi_m\rangle\langle\psi_m|)^\dagger (|\psi_m\rangle\langle\psi_m|) = |\psi_m\rangle \underbrace{\langle\psi_m|\psi_m\rangle}_{=1} \langle\psi_m| = |\psi_m\rangle\langle\psi_m| = M_m. \tag{3.48}$$

Thus:

$$\begin{aligned} p(m) &= \mathrm{Tr}(M_m \rho_{\max}) && (3.49)\\ &= \mathrm{Tr}(M_m \frac{\mathsf{I}}{2^n}) && (3.50)\\ &= \frac{1}{2^n}\mathrm{Tr}(M_m) && (3.51)\\ &= \frac{1}{2^n}\mathrm{Tr}(|\psi_m\rangle\langle\psi_m|) && (3.52)\\ &= \frac{1}{2^n}. && (3.53) \end{aligned}$$

This shows that the probability is $\frac{1}{2^n}$ for any outcome m, meaning that the measurement results in a uniform distribution over the basis states. This property makes maximally mixed states highly desirable for defining security primitives in the context of quantum cryptography, as we will see in Section 5.

3.4 Partial Measurement and Partial Trace

At times, when dealing with n-qubit quantum systems, we are more interested in a subsystem of the larger system, or we may want to measure only a part of the system rather than the whole. These concepts are explained through the formalism of partial trace and partial measurement. This section follows [LaR18].

Suppose Alice and Bob hold a quantum state, denoted by ρ_{AB} . We wish to describe the subsystem that is held by Alice, denoted as ρ_A.

Definition 3.9 (Partial Trace)**.**
Let A and B be two subsystems making up the composite system described by the density operator ρ_{AB}. The partial trace over the B subsystem, denoted by Tr_B is defined as

$$\mathrm{Tr}_B[\rho_{AB}] = \sum_i (\mathsf{I}_A \otimes \langle i|_B)\rho_{AB}(\mathsf{I}_A \otimes |i\rangle_B), \tag{3.54}$$

where $\{|i\rangle\}$ is any orthonormal basis for the Hilbert space $\mathcal{H}_B$ of subsystem B. We often write $\rho_A = \mathrm{Tr}_B[\rho_{AB}]$.

The identity operator I_A in Equation 3.54 reflects the fact that we are focusing on the subsystem A in isolation and wish to leave it unchanged.

Furthermore, Equation 3.54 can be extended to take the partial trace of multiple subsystems while tracing out multiple subsystems. Let $S_1 = \{A_1, A_2, \ldots, A_m\}$ and $S_2 = \{B_1, B_2, \ldots, B_k\}$ such that $S \coloneqq S_1 \cup S_2$ and $S_1 \cap S_2 = \emptyset$. S_1 is the set of subsystems to keep, and S_2 is the set trace out. Define the mapping $f : S \to \{1, 2, \ldots, k+m\}$ such that $f(x)$ indicates the order of subsystem x in the larger quantum system ρ_S. Then:

$$\rho_{S_1} = \mathrm{Tr}_{S_2}[\rho_S] = \sum_{i_1=1}^{\dim(B_1)} \sum_{i_2=1}^{\dim(B_2)} \cdots \sum_{i_k=1}^{\dim(B_k)} \left(\bigotimes_{x \in S} O_{f(x)}^\dagger \right) \rho_S \left(\bigotimes_{x \in S} O_{f(x)} \right), \tag{3.55}$$

where

$$O_{f(x)} = \begin{cases} \mathsf{I}_x & \text{if } x \in S_1 \\ |i_j\rangle & \text{if } x \in S_2 \text{ and } x = B_j \end{cases}. \tag{3.56}$$

Here, $\{|i_j\rangle\}$ is an orthonormal basis for the Hilbert space $\mathcal{H}_{B_j}$ of subsystem B_j, and $\dim(B_j)$ is the dimension of $\mathcal{H}_{B_j}$. Equation 3.55 is particularly useful for simulating quantum computers, as it allows the partial trace of a quantum system to be taken in any order.

Furthermore, the reduced density matrix of ρ_{S_1} can be used to compute measurements on that subsystem within a larger quantum system. For instance, if Alice and Bob share the state ρ_{AB} and we wish to perform a measurement only on Alice's system, we first compute ρ_A by taking the partial trace over Bob's subsystem. Then, we perform the measurement on Alice's system and the post-measurement state can be computed according to Equation 3.44. Such measurements are referred to as "partial measurements" as only a subsystem of the whole has been measured. In Section 3.5, we will see an example of partial measurement.

3.5 Quantum Entanglement

Quantum entanglement is a purely quantum phenomenon without a classical counterpart. A special feature of entangled quantum particles is that measuring one part of the system affects the measurement results of other parts. In other words, these quantum particles are correlated, and the state of one particle cannot be described independently from the states of the other particles. Entanglement plays a crucial role in various applications of quantum information science, such as quantum teleportation, non-local games, quantum algorithms, enabling capabilities that would otherwise be impossible within the classical framework.

An example of the advantage provided by entanglement can be illustrated with a game where a referee, named Charlie, generates two random bits a and b and sends them to Alice and Bob, respectively. Alice and Bob win the game if they output bits x and y, such that $a \oplus b = x \wedge y$. Without going over the details, the optimal classical strategy yields a 75% winning probability, whereas

an optimal quantum strategy using entanglement achieves approximately an 85% winning probability. This game is known as the CHSH game, named after the authors of the relevant work: Clauser, Horne, Shimony, and Holt [CHSH69]. The CHSH game is just one example of a larger class of games, known as non-local games. A survey of non-local games can be found in [BBT05].

We will now formally define entanglement, which is better understood through the concept of separable states.

Definition 3.10 (Separable and Entangled States)**.**
Let $\mathcal{H}_A$ and $\mathcal{H}_B$ be two Hilbert spaces and let $|\psi\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$. Then the state $|\psi\rangle$ is said to be *separable* if there are states $|\phi\rangle \in \mathcal{H}_A$ and $|\varphi\rangle \in \mathcal{H}_B$ such that

$$|\psi\rangle = |\phi\rangle \otimes |\varphi\rangle \,. \tag{3.57}$$

Otherwise, the state $|\psi\rangle$ is called an *entangled state*.

The definition of an entangled state captures the fact it cannot be described by considering its constituent parts independently. A simple and an important example of an entangled particle is the "EPR pair", named after Einstein, Podolsky, and Rosen [EPR35].

Definition 3.11 (EPR State)**.**
The *EPR pair* is defined by

$$|\mathsf{EPR}\rangle = \left|\Phi^+\right\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right). \tag{3.58}$$

A circuit that prepares an EPR pair is provided in Figure (2).

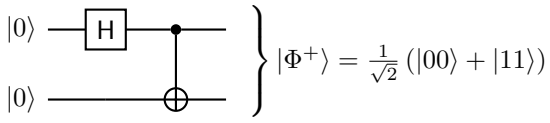


Figure 2: Quantum circuit that prepares an EPR pair.

As mentioned earlier, measuring one part of an entangled pair influences the result of measuring the other part, provided the measurements are performed in the same basis. In the next example, we will see the effect of measurement on the EPR state.

Example 3.5. Let $|\Phi^+\rangle = \frac{1}{\sqrt{2}}\left(|0_A 0_B\rangle + |1_A 1_B\rangle\right)$ be a shared entangled state between Alice and Bob. Firstly, tracing out Bob's system yields:

$$\rho_A = \mathrm{Tr}_B[\rho_{AB}] = \left(\mathsf{I}_A \otimes \langle 0|\right)\left|\Phi^+\right\rangle\left\langle\Phi^+\right|\left(\mathsf{I}_A \otimes |0\rangle\right) \tag{3.59}$$

$$+ \left(\mathsf{I}_A \otimes \langle 1|\right)\left|\Phi^+\right\rangle\left\langle\Phi^+\right|\left(\mathsf{I}_A \otimes |1\rangle\right) \tag{3.60}$$

$$= \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} = \frac{\mathsf{I}}{2}. \tag{3.61}$$

This means Alice's system in isolation corresponds to a maximally mixed state, reflecting the property of entangled particles: the description of the whole system requires all its constituent elements, and studying subsystems in isolation does not reveal precise information about them. Otherwise, it would be possible to describe the entangled pair as a separable state.

Secondly, suppose that Alice performs a measurement on her system. Since $\rho_A = \rho_{\max}$, the measurement results are 0 or 1 with equal probability. If she obtains the bit 0, then the post-measurement state corresponds to $|0_A 0_B\rangle$, which means that if Bob performs measurement on his part of the system in the computational basis, he also measures 0 on his part. Similarly, if Alice measures 1, then Bob will also measure 1. This demonstrates the importance of entangled pairs: a precise description of the system requires all its parts, and measuring one part affects the results of measurements on other parts.

3.5.1 Quantum Teleportation

According to a fundamental result in quantum information science, copying an arbitrary quantum state is impossible. This result, known as the "no-cloning theorem", is important because it highlights a key difference between classical and quantum information science. It must be noted that copying classical information is so straightforward nowadays that we take it for granted; however, it is a property of classical information that could have failed to exist, as that is the case for quantum information science. In contrast, a quantum state $|\psi\rangle$ can be teleported from Alice to Bob using entanglement. This is also a quantum phenomenon with no classical counterpart. It is important to note that teleporting a quantum state does not violate the no-cloning theorem, as the state $|\psi\rangle$ is not duplicated in the process. In other words, before and after the quantum teleportation process there is only one state $|\psi\rangle$. At times, the objective of employing a quantum teleportation gadget is to apply a quantum gate to the teleported state. In such cases, we refer to this process as gate teleportation. In this work, we view teleportation as an umbrella term encompassing both state teleportation and gate teleportation. For a more in-depth discussion on quantum teleportation, interested readers are encouraged to refer to [ZLC00].

In the next example, we provide a teleportation gadget that allows Alice to teleport her qubit to Bob by making use of an entangled pair.

Example 3.6. Suppose Alice possesses a qubit $|\psi\rangle$ and she wishes to teleport it to Bob. To achieve this, Alice and Bob are required to share an entangled pair. Figure (3) illustrates a teleportation circuit that enables Alice to teleport her state to Bob. First, Alice and Bob create an entangled state indicated by $|\Phi^+\rangle$. Then, Alice will perform a CNOT gate followed by a H gate on her system and make two measurements. Bob will subsequently perform conditional X and Z gates to recover the state $|\psi\rangle$.

Let us verify the correctness. After creating the entanglement, Alice and Bob hold a three-qubit system:

$$|\psi_0\rangle = |\psi\rangle \left|\Phi^+\right\rangle \tag{3.62}$$

where $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ is the state that Alice wants to send to Bob, and $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is the EPR pair shared between Alice and Bob.

In other words,

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}\left[\alpha|0\rangle(|00\rangle + |11\rangle) + \beta|1\rangle(|00\rangle + |11\rangle)\right]. \tag{3.63}$$

At this point, Alice applies a CNOT gate to her part of the system to obtain $|\psi_1\rangle$:

$$|\psi_1\rangle = (\mathsf{CNOT} \otimes \mathsf{I})(|\psi_0\rangle) \tag{3.64}$$

$$= \frac{1}{\sqrt{2}}\left[\alpha|0\rangle(|00\rangle + |11\rangle) + \beta|1\rangle(|10\rangle + |01\rangle)\right] \tag{3.65}$$

Then Alice applies the Hadamard gate H. To illustrate the effect of the Hadamard gate, let us take a brief detour and consider its impact on an arbitrary quantum state. This observation will help with the calculation that follow:

$$\mathsf{H}|\psi\rangle = \mathsf{H}(\alpha|0\rangle + \beta|1\rangle) \tag{3.66}$$

$$= \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot (\alpha|0\rangle + \beta|1\rangle) \tag{3.67}$$

$$= \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{3.68}$$

$$= \frac{1}{\sqrt{2}} \begin{bmatrix} \alpha + \beta \\ \alpha - \beta \end{bmatrix} \tag{3.69}$$

$$= \frac{1}{\sqrt{2}}\left[(\alpha + \beta)|0\rangle + (\alpha - \beta)|1\rangle\right] \tag{3.70}$$

$$= \frac{1}{\sqrt{2}}\left[\alpha(|0\rangle + |1\rangle) + \beta(|0\rangle - |1\rangle)\right]. \tag{3.71}$$

Now returning to the protocol:

$$|\psi_2\rangle = (\mathsf{H} \otimes \mathsf{I} \otimes \mathsf{I})|\psi_1\rangle \tag{3.72}$$

$$= \frac{1}{2}\left[\alpha(|0\rangle + |1\rangle)(|00\rangle + |11\rangle) + \beta(|0\rangle - |1\rangle)(|10\rangle + |01\rangle)\right] \tag{3.73}$$

We can rewrite the state as follows:

$$|\psi_2\rangle = \frac{1}{2}[\alpha(|0\rangle + |1\rangle)(|00\rangle + |11\rangle) + \beta(|0\rangle - |1\rangle)(|10\rangle + |01\rangle)] \tag{3.74}$$

$$= \frac{1}{2}[\,|00\rangle\,(\alpha\,|0\rangle + \beta\,|1\rangle) \tag{3.75}$$

$$+ |01\rangle\,(\alpha\,|1\rangle + \beta\,|0\rangle) \tag{3.76}$$

$$+ |10\rangle\,(\alpha\,|0\rangle - \beta\,|1\rangle) \tag{3.77}$$

$$+ |11\rangle\,(\alpha\,|1\rangle - \beta\,|0\rangle)] \tag{3.78}$$

Alice will then perform two measurements in the computational basis, obtaining classical bits a and b as shown in Figure (3). Suppose Alice measures $(a, b) = (0, 0)$, then Bob's post-measurement state corresponds to $\alpha\,|0\rangle + \beta\,|1\rangle$, as indicated by Equation 3.74. In total, there are four possible post-measurement outcomes $|\psi_3\rangle$, given the two measurements by Alice as shown in Table (4):

Measured Bits (a, b)	**Bob's Qubit** $\lvert\psi_3\rangle$	**Correction Gate**
(0,0)	$\alpha\,\lvert0\rangle + \beta\,\lvert1\rangle$	—
(0,1)	$\alpha\,\lvert1\rangle + \beta\,\lvert0\rangle$	X
(1,0)	$\alpha\,\lvert0\rangle - \beta\,\lvert1\rangle$	Z
(1,1)	$\alpha\,\lvert1\rangle - \beta\,\lvert0\rangle$	ZX

Table 4: Quantum Teleportation - Post-Measurement State

We can observe if the measured bits are $(0, 0)$, Bob's qubit, as indicated in Table (4) corresponds to the state $|\psi_4\rangle = |\psi\rangle$, which was originally in Alice's possession in the beginning of the protocol. If the measured bit is $(0, 1)$, applying the X gate results in:

$$\mathsf{X}\,(\alpha\,|1\rangle + \beta\,|0\rangle) = \alpha\,|0\rangle + \beta\,|1\rangle = |\psi\rangle\,. \tag{3.79}$$

This means Bob can recover the corresponding qubit $|\psi\rangle$. Similarly, upon measuring $(1, 0)$ and $(1, 1)$, applying gates Z and ZX, respectively, corrects the output so Bob can recover Alice's initial qubit to obtain $|\psi_4\rangle = |\psi\rangle$ precisely. More compactly, depending on the value of (a, b), $\mathsf{Z}^a\,\mathsf{X}^b\,|\psi_3\rangle = |\psi\rangle$. Hence, Alice has successfully teleported the state $|\psi\rangle$ to Bob.

As we will see in Section 5.4, another teleportation gadget plays a crucial role in security and correctness of the EPR quantum homomorphic encryption scheme.

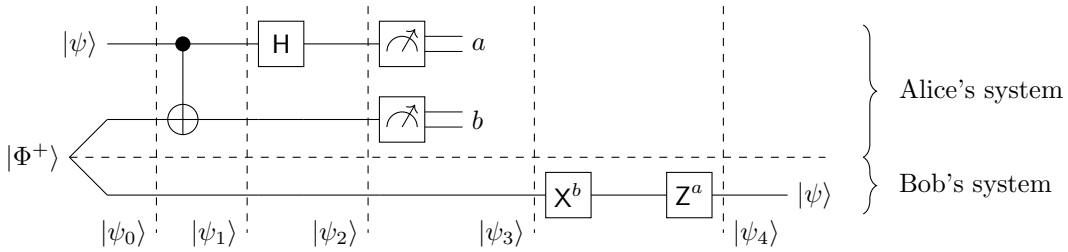


Figure 3: Quantum circuit for teleporting a qubit. The two lines following the measurement represent the measurement outcomes, which are now classical bits used for applying the conditional X^b and Z^a.

4 Cryptography

Cryptography is the study of secure communication. In this section, we outline various cryptographic concepts and protocols. In Section 4.1, we will introduce cryptographic terms and primitives that are relevant to both classical and quantum cryptography. The subsequent Sections 4.2 and 4.3 focus primarily on classical cryptography. As we will see in Section 5, both classical and quantum cryptography are needed for the development of a quantum homomorphic encryption scheme.

4.1 Semantics of Cryptography

To delve into cryptographic primitives, it is important to equip ourselves with the appropriate definitions and tools. This ensures that we have the appropriate language and framework to reason about cryptography. The terms introduced in this section are common in classical and quantum cryptography. This section follows from [KL21] and [SP19].

The fundamental objective of cryptography is to allow two or more parties to communicate over a channel, whether secure or insecure, in such a way that no adversary can understand the messages being sent over the channel. For simplicity, we consider the scenario where two parties, Alice and Bob, want to communicate.

- **Message Space:** The set of permissible messages to be communicated is called the *message space*, denoted by $\mathcal{M}$. Each message m is referred to as *plaintext*. In the classical setting, m is a bit string, where $m \in \mathcal{M} \subseteq \{0,1\}^*$. In the quantum setting, m is a quantum state and $\mathcal{M} \subseteq \mathcal{H}$.

- **Key Generation Algorithm and Key Space:** The *key generation algorithm* Gen is a probabilistic algorithm that returns a pair of keys $(k_{\text{enc}}, k_{\text{dec}})$[7]. The set of all such keys form the *key space* $\mathcal{K}$. For the purposes of this work, we consider keys to be classical bits, therefore, $k_{\text{enc}}, k_{\text{dec}} \in \{0,1\}^*$.

- **Encryption Algorithm and Ciphertext Space:** The *encryption algorithm* Enc is a function that encrypts the message m under the key k_{enc} and outputs the *ciphertext* c. That is,

$$\mathsf{Enc}_{k_{\text{enc}}}(m) = c. \tag{4.1}$$

  The set of all possible ciphertexts is referred to as the *ciphertext space*, denoted by $\mathcal{C}$. Similar to the message space, the ciphertexts are bit strings in the classical setting and quantum states in the quantum setting.

[7] In some cryptographic schemes, k_{enc} and k_{dec} may be equal, meaning the key generation algorithm only needs to generate a single key. This will be further discussed in Section 4.2.

- **Decryption Algorithm:** The *decryption algorithm* Dec takes as input a key k_{dec} and a ciphertext c and outputs a message m'. It is required that, except with negligible[8] probability over the randomness of Gen and Enc, we have $\mathsf{Dec}_{k_{\text{dec}}}(\mathsf{Enc}_{k_{\text{enc}}}(m)) = m$.

These terminologies provide a unified language when discussing classical and quantum cryptographic schemes.

4.2 Public-Key and Private-Key Cryptography

Encryption schemes are divided into two main categories: private-key and public-key encryption schemes. A private-key encryption scheme allows for symmetric communication by employing the same key for encryption and decryption, denoted as $k := k_{\text{enc}} = k_{\text{dec}}$. However, these schemes require that Alice and Bob share the secret key k securely before they can use the private-key encryption scheme. For example, one way to achieve this is by having Alice and Bob meet in-person to exchange the key k. The crucial point is that private-key encryption necessitates a secure channel for sharing the secret key, which can be impractical for applications over the internet and in other communication scenarios in the digital world.

This limitation is addressed by public-key cryptography, which facilitates asymmetric communication by using different keys during encryption and decryption. In the setting of public-key cryptography only one of the two communicating parties can send a message to the other party, hence the name asymmetric. Suppose Alice wants to send a message to Bob and they only have access to an insecure communication channel. Bob will generate a pair of keys $(k_{\text{enc}}, k_{\text{dec}})$. The encryption key k_{enc} is publicly announced and thus referred to as the *public key*, denoted by pk. Alice uses the public key to encrypt her message m, resulting in the ciphertext $c = \mathsf{Enc}_{pk}(m)$, which she sends to Bob. Then, Bob uses his secret, *private key* $k_{\text{dec}} = sk$ to decrypt the ciphertext to obtain Alice's original message m.

An example of public-key cryptography in action is online shopping, where Alice needs to provide her credit card information to Bob, the seller. Only Alice needs to send sensitive information to Bob, making this an instance of asymmetric cryptography. Given that the communication channel could be intercepted by an eavesdropper, public-key cryptography is an appropriate solution for such applications.

While primia facie public-key cryptography might initially seem preferable due to its flexibility and the lack of requirement for a pre-shared secret key, there are trade-offs to consider, which we will explore next.

[8] A function f from the natural numbers to the non-negative real numbers is *negligble* if for every polynomial p there is an N such that $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.

4.2.1 Perfect Secrecy

Ideally, we want an encryption scheme that is perfectly secure. Intuitively, we want an encryption scheme where, as long as the pair $(k_{\text{enc}}, k_{\text{dec}})$ remains secret, the encryption scheme remains secure regardless of any action taken by an adversary, Eve. Let us formalize the concept of a perfectly secure encryption scheme.

Definition 4.1 (Perfectly Secret Encryption Scheme)**.**
An encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ is *perfectly secret* if for every probability distribution for $\mathcal{M}$, every message $m \in \mathcal{M}$, and every ciphertext $c \in \mathcal{C}$ for which $\Pr[C = c] > 0$:

$$\Pr[M = m | C = c] = \Pr[M = m]. \tag{4.2}$$

An important theoretical result shows that perfectly secure encryption schemes do exist! One example of such a scheme is the one-time pad encryption scheme.

Definition 4.2 (One-time Pad Encryption Scheme)**.**
The one-time pad encryption scheme consists of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Let ℓ denote the length of the message $m \in \mathcal{M}$, where the message m is represented as a sequence of bits $m = (m_1, m_2, \ldots, m_\ell)$. The key generation algorithm Gen outputs a key k of length ℓ by sampling from the uniform distribution. The encryption algorithm Enc, encrypts the message m by performing a component-wise XOR (addition modulo 2) operation between the corresponding bits of m and k. That is,

$$\mathsf{Enc}_k(m) = m \oplus k = (m_1 \oplus k_1, m_2 \oplus k_2, \cdots, m_\ell \oplus k_\ell). \tag{4.3}$$

The decryption algorithm $\mathsf{Dec}_k(c)$ is defined by XORing the ciphertext c with the key k:

$$\mathsf{Dec}_k(c) = c \oplus k = (c_1 \oplus k_1, c_2 \oplus k_2, \cdots, c_\ell \oplus k_\ell). \tag{4.4}$$

The correctness of the one-time pad can be easily verified. Decrypting the ciphertext c with the key k results in

$$\mathsf{Dec}_k(c) = c \oplus k = (m \oplus k) \oplus k = m \oplus (k \oplus k) = m \oplus 2k = m, \tag{4.5}$$

as desired. Additionally, the one-time pad is perfectly secret, as proven in Appendix C. However, using the same key k more than once would render the encryption scheme insecure, which is why it is called the "**one**-time pad encryption". If the same key k is used to encrypt two distinct messages $m^{(1)}$ and $m^{(2)}$, an adversary that obtains the corresponding ciphertexts $c^{(1)}$ and $c^{(2)}$ can compute $c^{(1)} \oplus c^{(2)} = (m^{(1)} \oplus k) \oplus (m^{(2)} \oplus k) = m^{(1)} \oplus m^{(2)}$. Although, $m^{(1)} \oplus m^{(2)}$ does not directly reveal $m^{(1)}$ or $m^{(2)}$ individually, statistical analysis tools in cryptography can be used to recover $m^{(1)}$ and $m^{(2)}$. For instance, statistical attacks such as *index of coincidence* or testing common words and frequently-used letters can lead to partial information leakage and, with enough

persistence, to complete decryption. Historically, during the Cold War, the US Government decrypted Soviet Union messages encrypted with the one-time pad in the *VENONA* project, where the key was reused, demonstrating the importance of using the key only once [U.S21].

Furthermore, another important theoretical result states that in any perfectly secret encryption scheme, the key length of the key k must be at least as large as the message size. This is undesirable for practical purposes, as encrypting very large messages would require equally large keys. For example, to perfectly secure 4 gigabytes of data, a key of at least 4 gigabytes is needed. Additionally, another important theoretical result shows that achieving perfect secrecy in public-key cryptography is impossible. For a discussion around this impossibility result, see [KL21, 406]. In short, while perfect secrecy is theoretically achievable, it is too restrictive and impractical for real-world applications. Therefore, to achieve meaningful cryptography, we must adopt alternative approaches.

4.2.2 Computational Security

We have seen that achieving perfect secrecy is impractical for real-life scenarios. Instead, we trade perfect secrecy for a weaker form of secrecy by making certain assumptions. One such assumption is the computational complexity of solving difficult problems. This is motivated by the fact that some mathematical problems become "easy" to solve with additional information, which is referred to as a private key in cryptographic terms. Without this key, solving the problem remains inefficient. This is analogous to real-life scenarios, such as a public mailbox, where anyone can deposit mail, but only authorized postal personnel with a special key can access and distribute the mail. The security of the mailbox relies on the assumption that no adversary can duplicate the special key. Similarly, in cryptographic protocols, we assume that without the private key, no adversary can efficiently break the cryptosystem. To formalize the concept of an efficient adversary and a security framework based on computational security, we will introduce the concept of probabilistic polynomial-time algorithm. This will be explained next.

Definition 4.3 (Polynomial-Time Algorithm)**.**
A function $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$ from the natural numbers to the non-negative real numbers is *polynomial* if there is a constant c such that $f < n^c$ for all n. An algorithm $\mathcal{A}$ runs in polynomial-time if there exists a polynomial p such that, for every input $x \in \{0,1\}^*$, the computation of $\mathcal{A}(x)$ terminates within at most $p(|x|)$ steps. Here, $|x|$ denotes the length of the string x.

Polynomial-time algorithms are considered efficient. We remark that the notion of efficiency of algorithms is based on their practical utility in the field of computer science and cryptography. It is widely accepted in the field that polynomial-time algorithms are efficient due to their reasonable runtime for most applications. For more detailed discussions, see references such as [Gol08, KL21, BS23].

Furthermore, we allow all polynomial-time algorithms to be probabilistic. That is, the algorithm can access a sequence of unbiased and independent random bits. As noted in [KL21], "randomization ... - as far as we know - gives attackers additional power"[9]. We denote probabilistic polynomial-time algorithms as PPT.

Equipped with these concepts, we say that an encryption scheme is *computationally secure* if any PPT adversary succeeds in breaking the scheme with at most negligible probability.

4.2.2.1 RSA

Computationally secure encryption schemes form the foundation of secure communication over the internet. Many of these encryption schemes are based on the public-key RSA encryption scheme, named after its developers: Rivest, Shamir and Adleman [RSA78]. RSA itself is based on number-theoretic primitives, and we have included a section in Appendix B to outline the necessary technical concepts for completeness. We note that RSA has historical significance and is presented as a motivating example for the development of post-quantum security in Section 4.2.2.2.

Definition 4.4 (RSA Encryption Scheme)**.**
The RSA encryption scheme consists of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. The Gen algorithm generates two distinct large primes p and q, computes $n = p \cdot q$ and $\phi(n) = (p-1) \cdot (q-1)$. Furthermore, Gen randomly generates $e > 1$ such that $\gcd(e, \phi(n)) = 1$ and computes $d \equiv e^{-1} \pmod{\phi(n)}$. The Gen algorithm outputs the public key (n, e) and the private key (n, d). To encrypt the message m, the encryption algorithm Enc, using the public key pair (n, e) computes $m^e \pmod{n} = c$. The decryption of the ciphertext c is performed by $\mathsf{Dec}_{(n,d)}(c) = c^d \pmod{n}$.

The Miller-Rabin primality test can be used for generating the primes [Mil76, Rab80]. If for a given e, $\gcd(e, \phi(n)) > 1$, then the Gen algorithm can try a new value of e. The Euclidean algorithm efficiently computes the greatest common divisor of two numbers. Once an appropriate e is chosen, $d = e^{-1} \pmod{\phi(n)}$ can be efficiently computed by the extended Euclidean algorithm.

To see the correctness of RSA, suppose the message m is encrypted with the public key (n, e) to obtain $c = m^e \pmod{n}$. The ciphertext is decrypted using the private key (n, d) by computing c^d. Note that $c^d = (m^e)^d = m^{e \cdot d} \pmod{n}$. Since $e \cdot d \equiv 1 \pmod{\phi(n)}$, we have $m^{e \cdot d} \equiv m^{k \cdot \phi(n)+1} \pmod{n}$ for some integer k. Finally, $m^{k \cdot \phi(n)+1} \equiv m^{k \cdot \phi(n)} \cdot m \equiv m \pmod{n}$, where we used the Euler's theorem, which asserts $m^{k \cdot \phi(n)} \equiv 1 \pmod{n}$.

[9] An example of the practical advantage of probabilistic over deterministic algorithms is found in primality testing. The Miller-Rabin test [Mil76, Rab80], a probabilistic polynomial-time algorithm, is preferred for implementation purposes over its deterministic polynomial-time counterpart, the AKS primality test [AKS04]. Cryptographic textbooks such as [KL21, Gol08, SP19] focus on the Miller-Rabin test.

As mentioned, the RSA is an instance of a computationally secure encryption scheme. Specifically, its security relies on the hardness of the RSA problem, which is defined as follows: given an RSA public key (n, e) and a ciphertext $c = m^e \pmod{n}$, the challenge is to compute m. It is assumed that no efficient classical adversary can solve the RSA problem. While closely related to the integer factorization problem, it remains an open question whether the two are equivalent. Solving the integer factorization problem does solve the RSA problem, but it is not known if solving the RSA problem would also solve the integer factorization. Intuitively, the public exponent e might enable alternative attacks, potentially breaking RSA without factoring n.

To understand why solving the integer factorization leads to solving the RSA problem, note that if an adversary can factorize n into its prime components p and q, then the adversary can compute $\phi(n) = (p-1) \cdot (q-1)$. With $\phi(n)$ and e known, the adversary can efficiently compute d, which is part of the private key (n, d), allowing the adversary to decrypt the message m.

4.2.2.2 Post-Quantum Security

In Section 4.2.2, we outlined our expectations for a computationally secure scheme: an encryption scheme is computationally secure if any PPT adversary can break it with negligible probability. However, this definition has a drawback: it is independent of any computational model. While a rigorous treatment of a computational model is outside the scope of this current work, interested readers can refer to sources such as [AB09, Mar11] for more details. Here, we highlight important results.

Modern classical computers are approximations of the theoretical Turing machines framework [Tur37]. It is in this computational model that solving the RSA problem is assumed to be intractable. However, Turing machines are not the only model of computation. Quantum computers, modeled by quantum Turing machines, open up new possibilities for computation. The earliest proposed models of quantum computation can be traced back to 1985, due to Deutsch [Deu85]. Within the quantum model of computation, Shor developed an efficient quantum algorithm for solving the integer factorization problem, known as Shor's algorithm [Sho94]. As discussed in Section 4.2.2.1, integer factorization leads to solving the RSA problem. Therefore a powerful quantum computer is capable of breaking the RSA cryptosystem[10].

Therefore, post-quantum security refers to a revised definition of computational security, where adversaries are assumed to have access to powerful quantum computers. Despite the theoretical potential for quantum computers to break RSA and certain other number-theoretic encryption schemes like Diffie-Hellman [DH76], the challenge of post-quantum security is to identify problems that are hard for quantum computers. In public-key cryptography, lattice-based schemes

[10] Philosophically, the advent of quantum computers as a new computational model prompts questions about the existence of other, potentially more powerful, models of computation yet to be discovered. While intriguing, there is no definite answer to this. For more information, see [Aar06].

like NTRU [HPS98] and learning with errors (LWE) [Reg05] are considered strong candidates for post-quantum cryptography. While a detailed treatment of such schemes is beyond the scope of this work, the concept of post-quantum cryptography plays an important role in the contexts of quantum homomorphic encryption and the simulation of quantum homomorphic encryption. These topics will be further discussed in Sections 5 and 6. It is worth noting that the National Institute of Standards and Technology (NIST) in the United States began the standardization process for post-quantum secure cryptographic schemes in 2016 [Nat16]. After several rounds of formal reviews and improvements, NIST recently announced three new standards that are ready for immediate use [Nat24][11].

4.3 Classical Homomorphic Encryption

In Section 2.2.2, we defined the concept of homomorphism, which involves not only mapping elements of one algebraic structure but also mapping the binary operations between these two spaces. In Section 4 and up to this point, we have introduced various cryptographic primitives, including message space and ciphertext space. Homomorphic encryption combines the concepts of homomorphism and cryptography. The goal of homomorphic encryption is to enable computations to be performed on encrypted data which can then be decrypted to obtain the correct result.

In the literature, the first known introduction of the concept of homomorphic encryption, originally called *privacy homomorphism*, is attributed to a paper by Rivest, Adleman and Dertouzos in 1978 [RAD78]. Although significant progress was made over the years, it was not until 2009 that Craig Gentry proposed the first fully homomorphic encryption (FHE) scheme in his PhD thesis [Gen09]. FHE is considered the major breakthrough in classical homomorphic encryption as it allows for arbitrary computation on encrypted data. It is worth noting that although FHE schemes have been realized in theory, they require substantial computational resources and are not yet practical for most applications. Developing more efficient FHE schemes is an active area of research. For a brief discussion on this topic, refer to [Yac21].

In the remainder of this section, we will define what we mean by a classical homomorphic encryption (CHE) scheme. Then, we will provide a brief survey that includes a high-level summary of different types of classical homomorphic encryption schemes. CHE is required for performing key updates in the quantum homomorphic encryption scheme that we are considering in this work. This topic will be further discussed in Section 5. This section follows from [ABC+15, AAUC19, MSM+22, DMGD23].

[11] The information is accurate as of August 2024. Post-quantum cryptography is an evolving field and one must continuously monitor the latest developments to ensure a given post-quantum scheme still remains secure. For a historical example, see [SIK22], where the Supersingular Isogeny Key Encapsulation (SIKE) protocol was ultimately found to be insecure.

Definition 4.5 (Classical Homomorphic Encryption)**.**
A public-key homomorphic encryption scheme consists of four PPT algorithms $(\mathsf{CHE.Gen}, \mathsf{CHE.Enc}, \mathsf{CHE.Eval}, \mathsf{CHE.Dec})$ such that:

- The public-key generation algorithm $\mathsf{CHE.Gen}$ takes as input the security parameter λ and outputs the secret key sk, the public key pk and the public evaluation key evk[12].
- The public encryption algorithm $\mathsf{CHE.Enc}$ takes as input the public key pk and a message m and it outputs the ciphertext c.
- The decryption algorithm $\mathsf{CHE.Dec}$ takes as input the secret key sk and a ciphertext c. Next, it outputs a message m or $\perp$, if the decryption algorithm cannot successfully recover the encrypted message m.
- The evaluation algorithm $\mathsf{CHE.Eval}$ takes as input the evaluation key evk, a function f and n-tuple of ciphertexts $(c_1, c_2, \cdots, c_n)$. It outputs a ciphertext c_f, such that it decrypts to the result of evaluation $(m_1, m_2, \cdots, m_n)$ under f. That is, $c_f = \mathsf{CHE.Eval}_{evk}(f, (c_1, c_2, \cdots, c_n))$ and $\mathsf{CHE.Dec}_{sk}(c_f) = f(m_1, m_2, \cdots, m_n)$.

The main difference between homomorphic encryption and non-homomorphic encryption schemes lies in the Eval algorithm. Non-homomorphic encryption schemes do not have an evaluation algorithm, whereas homomorphic encryption schemes do. The evaluation algorithm is responsible for performing computation on encrypted data.

4.3.1 A Survey of Classical Homomorphic Encryption

Before delving into the classifications of classical homomorphic encryption schemes, we remark that a major challenge in developing FHE schemes is noise handling, as highlighted in [FV12]. Noise is added during encryption to maintain security, and it accumulates during homomorphic computations, potentially causing decryption to fail. Craig Gentry introduced "bootstraping" technique to address this issue, allowing the evaluator to refresh the ciphertext and reduce noise. This ensures that during decryption, substantial computation is not required by the client. In other words, the length of the ciphertext is independent of the evaluated circuit f. This concept is captured by the notion of compactness, which we will define next, adopted from [BV11]. We note that the compactness property is designed to rule out the trivial case where the evaluation algorithm simply outputs the circuit description without performing the computation, thereby offloading the evaluation procedure onto the client during decryption.

[12] Not all homomorphic encryption schemes require an evaluation key. For an example, see [GSW13].

Definition 4.6 (Compactness)**.**
A homomorphic encryption scheme is *compact* if there exists a polynomial p such that the output length of Eval is at most $p(\lambda)$.

Below we provide a brief survey of different types of classical homomorphic encryption schemes. We categorize these schemes into two classifications: partially homomorphic encryption (PHE) and multi-operational homomorphic encryption (MOHE) schemes[13]. Note that not all MOHE schemes need to satisfy the compactness property.

- **Partially Homomorphic Encryption:**

  The two main operations that a CHE scheme aims to preserve are addition and multiplication on encrypted data. PHE schemes support either addition or multiplication, but not both.

  - **Additive Homomorphic Encryption:**

    A CHE scheme that allows addition on encrypted data is called an *additive homomorphic encryption scheme*. An example of such a scheme is the Paillier cryptosystem [Pai99].

  - **Multiplicative Homomorphic Encryption:**

    A CHE scheme that allows multiplication on encrypted data is called a *multiplicative homomorphic encryption scheme*. For example, the RSA cryptosystem can be modified to handle multiplication on encrypted data.

- **Multi-Operational Homomorphic Encryption:**

  Unlike PHE that supports one type of operation, MOHE supports both addition and multiplication. However, there are different MOHE schemes depending on the types of circuits and the circuit depth they can handle.

  - **Somewhat Homomorphic Encryption:**

    A *somewhat homomorphic encryption* (SHE) scheme permits addition and multiplication on encrypted data, but only for a specified subset of circuits. Let $\mathcal{A}$ denote the set of all classical circuits. Then, an SHE scheme can perform computation on encrypted data for circuits $\mathcal{B}$, where $\mathcal{B} \subset \mathcal{A}$. An example of an SHE scheme can be found in [IP07].

[13] We note that the term "multi-operational homomorphic encryption" is not commonly used in the literature. This term is introduced here as an umbrella term to encompass various homomorphic encryption schemes that support more than one type of operation on encrypted data, aiming to facilitate more precise language.

– **Levelled**[14] **Homomorphic Encryption and Fully Levelled Homomorphic Encryption**:

A *levelled homomorphic encryption* (LHE) is a type of somewhat homomorphic encryption scheme where the Gen algorithm takes an additional parameter d, which specifies the maximum depth of circuits that can be evaluated. Furthermore, LHE schemes must satisfy the compactness property. An example of an LHE scheme is found in [Bra12].

A *levelled fully homomorphic encryption* (LFHE) scheme is one where there is no restriction on the set of evaluated circuits but is still constrained by the depth d.

– **Fully Homomorphic Encryption**:

A *fully homomorphic encryption* scheme is a homomorphic encryption scheme that is compact and capable of evaluating any circuits. Craig Gentry's construction is an example of an FHE scheme [Gen09].

Having reviewed different types of CHE, we will now shift our focus to cryptographic concepts in the quantum setting in Section 5. The concepts from classical cryptography will be useful in this context.

[14] We note that in the literature the word "levelled" also appears with the spelling "leveled" as in [Bra12]. Here, we adopt the spelling used in [BJ15].

5 Quantum Cryptography

Quantum cryptography is the study of cryptographic methods in the quantum setting. Using the laws of quantum mechanics, quantum cryptography enables new techniques for secure communication using quantum data.

In Section 5.1, we introduce the quantum one-time pad encryption scheme, the quantum counterpart of the classical one-time pad. We define our expectations for a secure quantum encryption scheme and then prove the security of the quantum one-time pad. In Section 5.2, we define quantum homomorphic encryption, similar to the classical homomorphic encryption introduced in Section 4.3. In subsequent Sections 5.3 and 5.4, we construct a quantum homomorphic encryption scheme, known as the "EPR scheme", which is capable of performing universal quantum computation on encrypted data.

5.1 Quantum One-Time pad

Quantum one-time pad (QOTP) is the quantum counterpart of the classical one-time pad introduced in Section 4.2.1, where all the bits of the message m are XORed with random key k:

$$\mathsf{Enc}(m) = m \oplus k = c. \tag{5.1}$$

The key k is used only once, since using it more than one time would render the encryption insecure. The decryption is performed by XORing the ciphertext with the key k to obtain the original message:

$$\mathsf{Dec}(c) = c \oplus k = m \underbrace{\oplus k \oplus k}_{=2k=0} = m. \tag{5.2}$$

First, we will define the quantum one-time pad for an n-qubit message. Then we will show its correctness and prove its security for a single qubit, noting that these results could be extended to an n-qubit message as well. This section follows from [BR03, WN18].

Definition 5.1 (Quantum One-time Pad Scheme)**.**
The quantum one-time pad encryption scheme for an n-qubit system consists of 3-tuple of quantum algorithms $(\mathsf{QOTP.Gen}, \mathsf{QOTP.Enc}, \mathsf{QOTP.Dec})$:

- **Key Generation:** The $\mathsf{QOTP.Gen}$ algorithm generates a classical secret key k uniformly at random as tuple (a, b), each component being of length n. Thus, k is expressed as $((a_1, \ldots, a_n), (b_1, \ldots, b_n))$.
- **Encryption:** The $\mathsf{QOTP.Enc}_{a,b}$, encrypts the n-qubit state $|\psi\rangle$ using the tuple (a, b) as follows:

$$\mathsf{QOTP.Enc}_{a,b}(|\psi\rangle) = \left[\bigotimes_{i=1}^{n} \mathsf{X}^{a_i}\, \mathsf{Z}^{b_i}\right] |\psi\rangle = \left|\tilde{\psi}\right\rangle. \tag{5.3}$$

  The encrypted state $\left|\tilde{\psi}\right\rangle$ is referred to as the *cipherstate*.

- **Decryption:** The decryption algorithm $\mathsf{QOTP.Dec}_{a,b}$ takes as input the cipherstate $\left|\tilde{\psi}\right\rangle$ and the key (a, b) and performs the following operation to yield the original input $|\psi\rangle$.

$$\mathsf{QOTP.Dec}_{a,b}\left(\left|\tilde{\psi}\right\rangle\right) = \left[\bigotimes_{i=1}^{n} \mathsf{Z}^{b_i}\,\mathsf{X}^{a_i}\right]\left|\tilde{\psi}\right\rangle = |\psi\rangle\,. \tag{5.4}$$

Note the order of operations: $\mathsf{X}^a\,\mathsf{Z}^b$ during encryption and $\mathsf{Z}^b\,\mathsf{X}^a$ during decryption. This order ensures the correctness of the scheme, as will be shown shortly. We also note that while quantum one-time pad is a quantum encryption scheme, it relies on classical secret keys a and b. The interplay of these classical keys within a quantum encryption scheme motivates the use of classical homomorphic encryption scheme in the context of the quantum homomorphic encryption scheme, as will be discussed in Sections 5.3 and 5.4[15].

Going forward we will refer to the components of the secret key k, (a, b), as *sub-keys*. To prove the correctness of the QOTP for a single-qubit, we let the sub-keys a and b be one-bit long.

Let $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix}$. Then we will encrypt the state $|\psi\rangle$ by applying $\mathsf{X}^a\,\mathsf{Z}^b$, where $a, b \in \{0, 1\}$. Applying $\mathsf{X}^a\,\mathsf{Z}^b$ to $|\psi\rangle$ results in one the four possibilities as described in Table (5).

(a, b)	$\mathsf{X}^a\,\mathsf{Z}^b\,\|\psi\rangle$
(0,0)	$\alpha\,\|0\rangle + \beta\,\|1\rangle$
(1,0)	$\beta\,\|0\rangle + \alpha\,\|1\rangle$
(0,1)	$\alpha\,\|0\rangle - \beta\,\|1\rangle$
(1,1)	$-\beta\,\|0\rangle + \alpha\,\|1\rangle$

Table 5: Encryption of the single qubit $|\psi\rangle$ using QOTP

Therefore, $\mathsf{QOTP.Enc}_{a,b}(|\psi\rangle) = \mathsf{X}^a\,\mathsf{Z}^b\,|\psi\rangle$. To decrypt, Bob applies $\mathsf{Z}^b\,\mathsf{X}^a$ to $\mathsf{QOTP.Enc}_{a,b}(|\psi\rangle)$. After applying $\mathsf{Z}^b\,\mathsf{X}^a$ to $\mathsf{QOTP.Enc}_{a,b}(|\psi\rangle)$, Bob will retrieve the original qubit:

$$\begin{aligned}
\mathsf{QOTP.Dec}_{a,b}(\mathsf{QOTP.Enc}_{a,b}(|\psi\rangle)) &= \mathsf{QOTP.Dec}_{a,b}(\mathsf{X}^a\,\mathsf{Z}^b\,|\psi\rangle) && (5.5)\\
&= \mathsf{Z}^b\,\mathsf{X}^a\left(\mathsf{X}^a\,\mathsf{Z}^b\,|\psi\rangle\right) && (5.6)\\
&= \mathsf{Z}^b\,\mathsf{X}^a\,\mathsf{X}^a\,Z^b\,|\psi\rangle && (5.7)\\
&= \mathsf{Z}^b\,\underbrace{\mathsf{X}^a\,\mathsf{X}^a}_{\mathbb{I}}\,\mathsf{Z}^b\,|\psi\rangle && (5.8)\\
&= |\psi\rangle && (5.9)
\end{aligned}$$

[15] For encryption schemes where keys are allowed to be quantum states, an interested reader may refer to [BGHD+23].

5.1.1 Security of the Quantum One-Time Pad

From the point of view of an adversary, Eve, who does not have the full knowledge of the state $|\psi\rangle$, the encrypted message appears as one of the states $\{|\psi\rangle, \mathsf{X}|\psi\rangle, \mathsf{Z}|\psi\rangle, \mathsf{X}\mathsf{Z}|\psi\rangle\}$. The best strategy at Eve's disposal is to apply $\mathsf{X}^a\mathsf{Z}^b$ to $\mathsf{QOTP.Enc}(|\psi\rangle)$ where $a, b \in \{0, 1\}$. After measurement in the computational basis, on average, Eve would observe the bit 0 with a probability of $\frac{|\alpha|^2+|\beta|^2+|\alpha|^2+|\beta|^2}{4} = \frac{2(|\alpha|^2+|\beta|^2)}{4} = \frac{1}{2}$ according to the frequency of occurrence of the probability amplitudes α and β associated to $|0\rangle$ as shown in Table (5). A similar argument shows that the bit 1 is also observed 50% of times. This implies that to an Eve that is only capable of making measurements in the computational basis, under perfect conditions, the encrypted message looks completely random. The best she can do in that case is to rely on a random strategy and luck. However, making measurements in the computational basis is only one possible attack available to Eve. A powerful adversary can make measurements in any basis, and our goal is to show that the QOTP scheme remains secure against such a powerful adversary. To achieve this, we will first outline the requirements for a secure QOTP and then provide a general proof using the language of density matrices. This approach allows us to show the security of QOTP independent of the choice of basis.

Let $\rho = |\psi\rangle\langle\psi|$ be the density matrix of the input message and ρ_{enc} denote the encrypted message. The QOTP protocol is secure if for every message ρ, the output state ρ_{enc} is the totally mixed state. There are two reasons why this makes it a good security requirement as pointed out in [BR03]:

(1) The encrypted state must be independent of the input message.

(2) Applying any unitary operation U, including all the X and Z operators, to the maximally mixed state $\rho_{\max}$ maps the state to itself $\rho_{\max}$:

$$\mathsf{U}\,\rho_{\max}\,\mathsf{U}^\dagger = \mathsf{U}\,\frac{\mathsf{I}}{2^n}\,\mathsf{U}^\dagger = \frac{1}{2^n}\,\mathsf{U}\,\mathsf{I}\,\mathsf{U}^\dagger = \frac{1}{2^n}\,\mathsf{U}\,\mathsf{U}^\dagger = \frac{\mathsf{I}}{2^n} = \rho_{\max}. \tag{5.10}$$

This implies that Eve will not gain any new information by applying any unitary to the encrypted state, assuming that the encrypted state is the maximally mixed state.

Having established a notion of security, to prove the security of QOTP, we should show that to an adversary that does not know the pair of keys (a, b), the encrypted state appears as a maximally mixed state, which implies that the result of the measurement in *any* basis of measurement is going to lead to the a uniform distribution of basis states. To do so, we will first show that the set $\{\sigma_i\}$ forms a basis over 2×2 Hermitian matrices.

Theorem 5.1.
Pauli matrices coupled with the identity matrix I form a basis over 2×2 Hermitian matrices.

$$\mathcal{B} = \{\mathsf{I}, \mathsf{X}, \mathsf{Y}, \mathsf{Z}\} = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\} \tag{5.11}$$

In particular, an arbitrary 2×2 density matrix ρ can be expressed in terms of the Pauli matrices along with the identity matrix as

$$\rho = \frac{1}{2}\left(c_0\,\mathsf{I} + c_1\,\mathsf{X} + c_2\,\mathsf{Y} + c_3\,\mathsf{Z}\right) \tag{5.12}$$

where

$$c_0 = \mathrm{Tr}(\rho), \quad c_1 = \mathrm{Tr}(\rho\,\mathsf{X}) \quad c_2 = \mathrm{Tr}(\rho\,\mathsf{Y}) \quad c_3 = \mathrm{Tr}(\rho\,\mathsf{Z}) \tag{5.13}$$

Proof. First, we want to show the matrices σ_i are linearly independent. That is, for any $c_i \in \mathbb{C}$, we would like to show that $c_0\sigma_1 + c_1\sigma_1 + c_2\sigma_2 + c_3\sigma_3 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ implies that $c_0 = c_1 = c_2 = c_3 = 0$. Expanding $c_0\sigma_1 + c_1\sigma_1 + c_2\sigma_2 + c_3\sigma_3$ yields:

$$c_0\sigma_1 + c_1\sigma_1 + c_2\sigma_2 + c_3\sigma_3 = \begin{bmatrix} c_0 & 0 \\ 0 & c_0 \end{bmatrix} + \begin{bmatrix} 0 & c_1 \\ c_1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -c_2 i \\ c_2 i & 0 \end{bmatrix} + \begin{bmatrix} c_3 & 0 \\ 0 & -c_3 \end{bmatrix} \tag{5.14}$$

$$= \begin{bmatrix} c_0 + c_3 & c_1 - c_2 i \\ c_1 + c_2 i & c_0 - c_3 \end{bmatrix} \tag{5.15}$$

Equating the Equation 5.15 with the zero matrix yields, $c_0 + c_3 = 0$ and $c_0 - c_3 = 0$, which implies $c_0 = c_3 = 0$. By similar reasoning, it follows $c_1 = c_2 = 0$. Therefore, $c_0 = c_1 = c_2 = c_3 = 0$ as desired.

Next, we need to show that $\mathcal{B}$ spans an arbitrary 2×2 density matrix. Let $\rho = \begin{bmatrix} b_0 & b_1 \\ b_2 & b_3 \end{bmatrix}$ be such a matrix. Then we need to find coefficients c_0, c_1, c_2 and c_3 such that $\sum_{i=0}^{3} c_i\sigma_i = \rho$. Using Equation 5.15 we get:

$$\begin{bmatrix} c_0 + c_3 & c_1 - c_2 i \\ c_1 + c_2 i & c_0 - c_3 \end{bmatrix} = \begin{bmatrix} b_0 & b_1 \\ b_2 & b_3 \end{bmatrix}. \tag{5.16}$$

Equation 5.16 implies that $c_0 + c_3 = b_0 \Rightarrow c_3 = b_0 - c_0$. Substituting $b_0 - c_0$ for c_3 in $c_0 - c_3 = b_3$ yields $c_0 + c_0 - b_0 = b_3 \Rightarrow c_0 = \frac{b_0 + b_3}{2}$. Note that $b_0 + b_3 = \mathrm{Tr}(\rho)$. Similar reasoning will reveal that $c_1 = \frac{\mathrm{Tr}(\rho\,\mathsf{X})}{2}$, $c_2 = \frac{\mathrm{Tr}(\rho\,\mathsf{Y})}{2}$ and finally $c_3 = \frac{\mathrm{Tr}(\rho\,\mathsf{Z})}{2}$. Defining a new variable $c_i' = \mathrm{Tr}(\rho\sigma_i)$ shows that

$$\rho = \frac{1}{2} \cdot \sum_{i=0}^{3} c_i'\sigma_i, \tag{5.17}$$

□

Example 5.1. Let

$$\rho = |+\rangle\langle+| = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} \times \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

We are going to use Theorem 5.1 to express ρ as a linear combination of matrices in the set $\{\mathsf{I}, \mathsf{X}, \mathsf{Y}, \mathsf{Z}\}$.

First we need to compute $\rho\mathsf{X}, \rho\mathsf{Y}, \rho\mathsf{Z}$ and then compute half the traces of $\rho, \rho\mathsf{X}, \rho\mathsf{Y}, \rho\mathsf{Z}$:

$$\rho\mathsf{X} = \frac{1}{2}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} \times \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} = \frac{1}{2}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} \tag{5.18}$$

$$\rho\mathsf{Y} = \frac{1}{2}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} \times \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix} = \frac{1}{2}\begin{bmatrix}i & -i\\ i & -i\end{bmatrix} \tag{5.19}$$

$$\rho\mathsf{Z} = \frac{1}{2}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} \times \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}1 & -1\\ 1 & -1\end{bmatrix} \tag{5.20}$$

It follows that

$$c_0 = \mathrm{Tr}(\rho) = \mathrm{Tr}\left(\frac{1}{2}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\right) = \frac{1}{2}(1+1) = 1 \tag{5.21}$$

$$c_1 = \mathrm{Tr}(\rho\mathsf{X}) = \mathrm{Tr}\left(\frac{1}{2}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\right) = \frac{1}{2}(1+1) = 1 \tag{5.22}$$

$$c_2 = \mathrm{Tr}(\rho\mathsf{Y}) = \mathrm{Tr}\left(\frac{1}{2}\begin{bmatrix}i & -i\\ i & -i\end{bmatrix}\right) = \frac{1}{2}(i-i) = 0 \tag{5.23}$$

$$c_3 = \mathrm{Tr}(\rho\mathsf{Z}) = \mathrm{Tr}\left(\frac{1}{2}\begin{bmatrix}1 & -1\\ 1 & -1\end{bmatrix}\right) = \frac{1}{2}(1-1) = 0 \tag{5.24}$$

Therefore by the result of Theorem 5.1

$$\rho = \frac{1}{2}\left(c_0\,\mathsf{I} + c_1\,\mathsf{X} + c_2\,\mathsf{Y} + c_3\,\mathsf{Z}\right) \tag{5.25}$$

and substituting the corresponding values of c_i, we have:

$$\rho = \frac{1}{2}\left(\mathsf{I} + \mathsf{X}\right). \tag{5.26}$$

Let us verify its correctness:

$$\frac{1}{2}\left(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} + \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\right) = \frac{1}{2}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} = \rho \tag{5.27}$$

as expected.

The set $\{\mathsf{I}, \mathsf{X}, \mathsf{Y}, \mathsf{Z}\}$ is a basis for 2×2 Hermitian matrices. In Section 3.2.1, we previously established that $\mathsf{X}\,\mathsf{Z} \equiv \mathsf{Y}$. With this in mind, we observe that the set $\{\mathsf{X}^\alpha\,\mathsf{Z}^\beta \,|\alpha, \beta \in \{0, 1\}\}$, also forms a basis over 2×2 Hermitian matrices, up to global phase factor. Thus in general, any density matrix can be written as

$$\rho = \sum_{\alpha,\beta} c_{\alpha,\beta}\,\mathsf{X}^\alpha\,\mathsf{Z}^\beta \tag{5.28}$$

where

$$c_{\alpha,\beta} = \frac{\mathrm{Tr}(\rho\,\mathsf{X}^\alpha\,\mathsf{Z}^\beta)}{2}. \tag{5.29}$$

Now we have all the necessary ingredients to prove the security of the quantum one-time pad. We can express the input message $|\psi\rangle$ as $\rho = |\psi\rangle\langle\psi|$. Moreover, since the set $\{\mathsf{X}^\alpha\,\mathsf{Z}^\beta : \alpha, \beta \in \{0, 1\}\}$ is a basis, we can represent ρ as $\rho = \sum_{\alpha,\beta} c_{\alpha,\beta}\,\mathsf{X}^\alpha\,\mathsf{Z}^\beta$ where $c_{\alpha,\beta} = \frac{\mathrm{Tr}(\rho\,\mathsf{X}^\alpha\,\mathsf{Z}^\beta)}{2}$.

Theorem 5.2. The quantum one-time pad is secure.

Proof. Let ρ denote the input message. Recall from the discussion in Section 3.3 that a density matrix, depending on one's knowledge, may correspond to different ensembles of quantum states. For Alice, who encrypts ρ using a pair of secret keys (a, b) and knows the values of the pair (a, b), the message is encrypted as $\rho_{\text{enc,Alice}} = \mathsf{X}^a\,\mathsf{Z}^b\,\rho\,\mathsf{Z}^b\,\mathsf{X}^a$. We wish to show that for an adversary without the knowledge of a and b, $\rho_{\text{enc,adv}} = \rho_{\max}$. To an adversary, the encrypted state $\rho_{\text{enc,adv}}$ appears as one of the four possible states:

$$\{\rho, \mathsf{X}\,\rho\,\mathsf{X}, \mathsf{Z}\,\rho\,\mathsf{Z}, \mathsf{X}\,\mathsf{Z}\,\rho\,\mathsf{Z}\,\mathsf{X}\}. \tag{5.30}$$

Thus, the encrypted state appears as the equal average of these four possibilities to an adversary:

$$\rho_{\text{enc,adv}} = \frac{1}{4} \sum_{a,b\in\{0,1\}} \mathsf{X}^a\,\mathsf{Z}^b\,\rho\,\mathsf{Z}^b\,\mathsf{X}^a \tag{5.31}$$

$$= \frac{1}{4} \sum_{\alpha,\beta\in\{0,1\}} \sum_{a,b\in\{0,1\}} \mathsf{X}^a\,\mathsf{Z}^b \left(c_{\alpha,\beta}\,\mathsf{X}^\alpha\,\mathsf{Z}^\beta\right) \mathsf{Z}^b\,\mathsf{X}^a \quad (\text{since } \rho = \sum_{\alpha,\beta} c_{\alpha,\beta} X^\alpha Z^\beta) \tag{5.32}$$

$$= \frac{1}{4} \sum_{\alpha,\beta\in\{0,1\}} c_{\alpha,\beta} \sum_{a,b\in\{0,1\}} \mathsf{X}^a\,\mathsf{Z}^b\,X^\alpha\,\mathsf{Z}^\beta\,\mathsf{Z}^b\,\mathsf{X}^a \tag{5.33}$$

Next, we will use the fact that $\mathsf{X}\,\mathsf{Z} = -\,\mathsf{Z}\,\mathsf{X}$ to move X^a and Z^b to the same side. For instance, as Z^b on the right of $X^\alpha Z^\beta$ moves to the left, it obtains a factor of $(-1)^{\alpha\cdot b}$. The factor $(-1)^{\beta\cdot a}$ comes from moving X^a to the left of Z^β. Thus, continuing from Equation 5.33, we obtain:

$$\frac{1}{4}\sum_{\alpha,\beta\in\{0,1\}} c_{\alpha,\beta} \sum_{a,b\in\{0,1\}} \mathsf{X}^a \mathsf{Z}^b \mathsf{X}^\alpha \mathsf{Z}^\beta \mathsf{Z}^b \mathsf{X}^a \tag{5.33}$$

$$=\frac{1}{4}\sum_{\alpha,\beta\in\{0,1\}} c_{\alpha,\beta} \sum_{a,b\in\{0,1\}} (-1)^{\alpha\cdot b\oplus\beta\cdot a} \mathsf{X}^\alpha \mathsf{Z}^\beta \tag{5.34}$$

There are four cases to consider based on the possible values of the pair (α,β):

(1) $(\alpha,\beta)=(0,0)$:

$$\frac{1}{4}\sum_{a,b\in\{0,1\}} (-1)^{0\cdot b\oplus 0\cdot a} \mathsf{X}^0 \mathsf{Z}^0 \tag{5.35}$$

$$=\frac{1}{4}(4\,\mathsf{I}) \tag{5.36}$$

$$=\mathsf{I} \tag{5.37}$$

(2) $(\alpha,\beta)=(1,0)$:

$$\frac{1}{4}\sum_{a,b\in\{0,1\}} (-1)^{1\cdot b\oplus 0\cdot a} \mathsf{X}^1 \mathsf{Z}^0 \tag{5.38}$$

$$=\frac{1}{4}\left((-1)^0\mathsf{X}+(-1)^1\mathsf{X}\right) \tag{5.39}$$

$$=\frac{1}{4}(\mathsf{X}-\mathsf{X})=0. \tag{5.40}$$

(3) $(\alpha,\beta)=(0,1)$:

$$\frac{1}{4}\sum_{a,b\in\{0,1\}} (-1)^{0\cdot b\oplus 1\cdot a} \mathsf{X}^0 \mathsf{Z}^1 \tag{5.41}$$

$$=\frac{1}{4}\left((-1)^0\mathsf{Z}+(-1)^1\mathsf{Z}\right) \tag{5.42}$$

$$=\frac{1}{4}(\mathsf{Z}-\mathsf{Z})=0 \tag{5.43}$$

(4) $(\alpha,\beta)=(1,1)$:

$$\frac{1}{4}\sum_{a,b\in\{0,1\}} (-1)^{1\cdot b\oplus 1\cdot a} \mathsf{X}\mathsf{Z} \tag{5.44}$$

$$= \frac{1}{4}\left((-1)^0 \mathsf{X}\mathsf{Z} + (-1)^1 \mathsf{X}\mathsf{Z} + (-1)^1 \mathsf{X}\mathsf{Z} + (-1)^0 \mathsf{X}\mathsf{Z}\right) \tag{5.45}$$

$$= \frac{1}{4}\left(\mathsf{X}\mathsf{Z} - \mathsf{X}\mathsf{Z} - \mathsf{X}\mathsf{Z} + \mathsf{X}\mathsf{Z}\right) = 0 \tag{5.46}$$

Therefore, we observe that the only non-zero term that remains is when $(\alpha, \beta) = (0, 0)$, which results in I. Going back to the initial expression, we have shown that:

$$\rho_{\text{enc,adv}} = \frac{1}{4} \sum_{a,b \in \{0,1\}} \mathsf{X}^a \mathsf{Z}^b \rho \mathsf{Z}^b \mathsf{X}^a \tag{5.47}$$

$$= \frac{1}{4} \sum_{\alpha,\beta \in \{0,1\}} c_{\alpha,\beta} \sum_{a,b \in \{0,1\}} \mathsf{X}^a \mathsf{Z}^b \mathsf{X}^\alpha \mathsf{Z}^\beta \mathsf{Z}^b \mathsf{X}^a \tag{5.48}$$

$$= c_{0,0}\, \mathsf{I}\,. \tag{5.49}$$

By Theorem 5.1, $c_{\alpha,\beta} = \frac{\text{Tr}(\rho \mathsf{X}^\alpha \mathsf{Z}^\beta)}{2^n}$. In particular, $c_{0,0} = \frac{\text{Tr}(\rho)}{2}$. Note that ρ is a Hermitian matrix and it is a property of Hermitian matrices that $\text{Tr}(\rho) = 1$. Therefore, it follows that:

$$\rho_{\text{enc,adv}} = c_{0,0}\, \mathsf{I} \tag{5.50}$$

$$= \frac{\text{Tr}(\rho)}{2}\, \mathsf{I} \tag{5.51}$$

$$= \frac{\mathsf{I}}{2} = \rho_{\max}. \tag{5.52}$$

This implies that, to the adversary, the encrypted state appears as a maximally mixed state, as desired. Hence, the quantum one-time pad is secure. □

We conclude this section by noting that, similar to the classical one-time pad, the secret key in QOTP should not be reused for encrypting multiple messages, as this would compromise the security of the scheme. If the same key (a, b) is used to encrypt multiple messages, an adversary can learn information about sub-keys a and b by performing measurements in different bases. For example, if $|\psi\rangle = |0\rangle$, then $\mathsf{QOTP.Enc}_{a,b}(|\psi\rangle) = \mathsf{X}^a \mathsf{Z}^b |\psi\rangle = \mathsf{X}^a \mathsf{Z}^b |0\rangle = \mathsf{X}^a |0\rangle = |a\rangle$. Measuring the encrypted state in the computational basis will reveal a. Similarly, if $|\psi\rangle = |+\rangle$, then:

$$\mathsf{Z}^b |+\rangle = \mathsf{Z}^b \left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right] \tag{5.53}$$

$$= \frac{1}{\sqrt{2}}\left(|0\rangle + (-1)^b |1\rangle\right). \tag{5.54}$$

Therefore,

- if $b = 0$, $\mathsf{Z}^b |+\rangle = |+\rangle$, and
- if $b = 1$, $\mathsf{Z}^b |+\rangle = |-\rangle$

Similarly, analyzing the effects of the H gate on $|b\rangle$ shows the following:

- if $b = 0$, $\mathsf{H} |0\rangle = |+\rangle$, and
- if $b = 1$, $\mathsf{H} |1\rangle = |-\rangle$.

Combining our observation yields:

- when $b = 0$, $\mathsf{Z}^b |+\rangle = |+\rangle = \mathsf{H} |b\rangle$, and
- when $b = 1$, $\mathsf{Z}^b |+\rangle = |-\rangle = \mathsf{H} |b\rangle$.

Hence, we can conclude that $\mathsf{Z}^b |+\rangle = \mathsf{H} |b\rangle$. Finally, we note that $\mathsf{X} |+\rangle = |+\rangle$ and $\mathsf{X} |-\rangle = - |-\rangle \equiv |-\rangle$ under the global phase factor. Hence, $\mathsf{X}^a \mathsf{Z}^b |+\rangle \equiv \mathsf{H} |b\rangle$. Measuring this state in the Hadamard basis will reveal b. Therefore, this simple attack shows a weakness of QOTP if the same key is reused for encrypting multiple messages.

5.2 Quantum Homomorphic Encryption

Quantum Homomorphic Encryption (QHE) serves as the quantum counterpart to classical homomorphic encryption. While the QHE has been studied for less time compared to its classical counterpart, there are many similar ideas between the two settings. As one might expect, the QHE enables the evaluation of encrypted data in quantum settings. Moreover, the study of classical homomorphic encryption schemes is not only important for motivating the problem, but it also plays a pivotal role in enabling QHE.

The following definition of a quantum homomorphic encryption scheme is a synthesis of [BJ15] and [ML22, 12]. In particular, the definition provided in [ML22] is independent of quantum channels.

Definition 5.2 (Quantum Homomorphic Encryption)**.**
A quantum homomorphic encryption scheme is a 4-tuple of quantum algorithms $(\mathsf{QGen}, \mathsf{QEnc}, \mathsf{QEval}, \mathsf{QDec})$:

- **Key Generation.** The algorithm QGen takes as input the security parameter λ and outputs a public encryption key pk and a public evaluation key evk, and a secret key sk.
- **Encryption.** The algorithm QEnc takes the public key pk and a state $|\psi\rangle$ as input and outputs a quantum cipherstate $\left|\tilde{\psi}\right\rangle$[16].

[16] In this work, we restrict our attention to unitary transformations. For a more general definition involving quantum channels, refer to [BJ15].

- **Homomorphic Evaluation.** The algorithm QEval takes the evaluation key evk and a quantum circuit C with n-fold cipherstate $\left|\tilde{\psi}\right\rangle$ and outputs m-fold cipherstate qubits, $\left|\tilde{\psi}'\right\rangle$.
- **Decryption.** The QDec algorithm takes the secret key sk and a quantum cipherstate $\left|\tilde{\psi}\right\rangle$ and outputs a state $|\psi_{\text{eval}}\rangle$.

5.3 Clifford Scheme

In this section, we describe the Clifford scheme CL , a compact[17] quantum homomorphic encryption scheme for circuits consisting of Clifford circuits. Clifford scheme is a building block for EPR scheme, which will be introduced in Section 5.4.

Clifford circuits, consisting exclusively of Clifford gates, introduced in Section 3.2.2, are gates that conjugate with Pauli operators. That is, for any Clifford gate C and a Pauli operator W, there exists a Pauli W' such that $\mathsf{CW} = \mathsf{W}'\mathsf{C}$.

In the context of the CL scheme, both encryption and decryption are performed using the QOTP, which is constructed from Pauli gates. The QOTP encrypts the quantum message by applying the Pauli operators X^a and Z^b, where a and b are classical keys. Specifically, when a Clifford gate C acts on an encrypted state, it transforms the keys such that $\mathsf{C}\,\mathsf{X}^a\,\mathsf{Z}^b = \mathsf{X}^{a'}\,\mathsf{Z}^{b'}\,\mathsf{C}$. The key update rules are provided in Table (6). To ensure that the key updates are performed securely by the server, the CL scheme employs a post-quantum secure homomorphic encryption scheme. We will elaborate on the correctness and the security of the CL scheme next.

- **Correctness:** The correctness of the scheme follows from the Clifford-Pauli conjugation property. When a Clifford gate C is applied to an encrypted quantum state $\mathsf{W}\,|\psi\rangle$, the resulting state is $\mathsf{W}'(\mathsf{C}\,|\psi\rangle)$. It is important to note that the Clifford-Pauli conjugation transforms W into W'. Here, W is a tensor product of Pauli operators, $\mathsf{W} = \mathsf{X}^{a_1}\,\mathsf{Z}^{b_1} \otimes \cdots \otimes \mathsf{X}^{a_n}\,\mathsf{Z}^{b_n}$, represented by a pair of keys $((a_1, \ldots, a_n), (b_1, \ldots, b_n))$. After applying the Clifford gate C, the transformed Pauli operator W' is described by the updated keys $((a'_1, \ldots, a'_n), (b'_1, \ldots, b'_n))$. The key update process can be described compactly by a *key update rule*, denoted by a function $f^{\mathsf{C}} : \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$. The function depends on the Clifford circuit C, and the operations are performed over the field with two elements. More succinctly, the action of a single-qubit Clifford gate on the ith wire can be described by functions $f_a^{\mathsf{C}}, f_b^{\mathsf{C}} : \mathbb{F}_2^2 \to \mathbb{F}_2$, where each function takes two bits (a_i, b_i) as input and returns the single bits a'_i and b'_i, respectively. Similarly, the key update rule for the two-qubit CNOT gate can be described by

[17] The concept of compactness in the quantum setting is similar to the concept of compactness in the classical setting: we require that the circuit complexity of the decryption algorithm to be independent from the the evaluated circuit C.

$f_a^{\mathsf{C}}, f_b^{\mathsf{C}} : \mathbb{F}_2^4 \to \mathbb{F}_2^2$, since the CNOT gate acts on two wires. Table (6) shows the key update rules for the Clifford gates. The correctness of the key update rules is shown in Appendix D.

- **Security:** To ensure the security of the scheme, we observe that the keys are classical and the key updates are also purely classical operations, despite the fact that the key themselves are used for QOTP, a quantum process. This interesting interplay between classical and quantum processes enables the Clifford scheme to incorporate a post-quantum secure homomorphic encryption scheme within a quantum homomorphic encryption scheme. This enables the evaluator to update encrypted keys *without having knowledge of the decrypted key values*. The client can then decrypt keys during the decryption process.

We note that the only binary operation in Table (6) is the XOR operation, denoted by $\oplus$. This implies that an additive homomorphic encryption scheme would suffice for the purposes of the CL scheme.

We are now ready to define the CL scheme.

Key Generation. $\mathsf{CL}.\mathsf{Gen}$: Let n denote the size of the input quantum message $|\psi\rangle$. The $\mathsf{CL}.\mathsf{Gen}$ algorithm generates two n-bit keys $(a, b) = ((a_1, \ldots, a_n), (b_1, \ldots, b_n))$ uniformly at random. Additionally, given the security parameter λ, $\mathsf{CHE}.\mathsf{Gen}$ generates a secret key sk, a public key pk, and an evaluation key evk[18].

Encryption. $\mathsf{CL}.\mathsf{Enc}$: The state $|\psi\rangle$ is encrypted using QOTP's encryption algorithm:

$$\mathsf{QOTP}.\mathsf{Enc}_{a,b}(|\psi\rangle) = \left[\bigotimes_{i=1}^{n} \mathsf{X}^{a_i}\,\mathsf{Z}^{b_i}\right]|\psi\rangle = \left|\tilde{\psi}\right\rangle. \tag{5.55}$$

Furthermore, the classical keys a and b are encrypted using $\mathsf{CHE}.\mathsf{Enc}$, resulting in $(\tilde{a}, \tilde{b}) = (\mathsf{CHE}.\mathsf{Enc}_{pk}(a), \mathsf{CHE}.\mathsf{Enc}_{pk}(b))$.

The quantum density matrix of the cipherstate $\left|\tilde{\psi}\right\rangle$, ρ, along with the encrypted classical keys $(\tilde{a}, \tilde{b})$, are then sent to the server.

Evaluation. $\mathsf{CL}.\mathsf{Eval}^{\mathsf{C}}$: Let C denote a Clifford circuit consisting of ℓ layers, represented as $\mathsf{C} = \mathsf{C}_\ell \circ \mathsf{C}_{\ell-1} \circ \cdots \circ \mathsf{C}_2 \circ \mathsf{C}_1$.

(1) For $i \in \{1, 2, \ldots, n\}$, set $f_{a,i} \coloneqq a_i$ and $f_{b,i} \coloneqq b_i$.

(2) For $j = 1, \ldots, \ell$ such that each C_j is an n-qubit Clifford gate:

[18] The security parameter λ determines the security of the scheme against brute-force attacks and it provides computational security up to 2^λ computational steps. Based on the chosen λ, the values of sk, pk and evk are set appropriately. Therefore, to be precise, one could denote them sk_λ, pk_λ and evk_λ.

Table 6: Key Updates Process for Clifford Gates

Gate	Before	After
$\mathsf{I}, \mathsf{X}, \mathsf{Z}$	(a_i, b_i)	(a_i, b_i)
H	(a_i, b_i)	(b_i, a_i)
P	(a_i, b_i)	$(a_i, a_i \oplus b_i)$
CNOT	$(a_i, b_i), (a_{i+1}, b_{i+1})$	$(a_i, b_i \oplus b_{i+1}), (a_i \oplus a_{i+1}, b_{i+1})$

(a) Apply the gate C_j to the state ρ, to obtain $\mathsf{C}_j \, \rho \, \mathsf{C}_j^{-1}$.

(b) Update the key values for the ith wire according to the key update rule given in Table (6). If the Clifford gate acting on the ith wire is a single-qubit gate, denoted by $\mathsf{C}_j^{(i)}$, then update $(f_{a,i}, f_{b,i}) \coloneqq (f_{a,i} \circ f_a^{\mathsf{C}_j^{(i)}}, f_{b,i} \circ f_b^{\mathsf{C}_j^{(i)}})$. Otherwise, if the acting Clifford gate is a CNOT gate acting on wires i and $i+1$, update $(f_{a,i}, f_{a,i+1}, f_{b,i}, f_{a,i+1})$.

(3) Update the classical encryptions:

$$e_i = \left(\mathsf{CHE}.\mathsf{Eval}_{evk}^{f_{a,i}}(\tilde{a}_i), \mathsf{CHE}.\mathsf{Eval}_{evk}^{f_{b,i}}(\tilde{b}_i) \right). \tag{5.56}$$

(4) The server outputs $(e_1, \ldots, e_\ell, \rho)$.

Decryption. $\mathsf{CL}.\mathsf{Dec}$: The encrypted classical bits $\tilde{a}, \tilde{b}$ are first decrypted using $\mathsf{CHE}.\mathsf{Dec}_{sk}$. Then, with the decrypted keys, the evaluated message ρ is decrypted using the decryption component of the QOTP: $\mathsf{QOTP}.\mathsf{Dec}_{\mathsf{CHE}.\mathsf{Dec}_{sk}(\tilde{a}), \mathsf{CHE}.\mathsf{Dec}_{sk}(\tilde{b})}$.

5.4 EPR Scheme

As noted in Section 3.2.5, adding any non-Clifford in addition to the generator of a Clifford group will create a universal quantum gate set. Therefore, to extend the CL scheme for evaluating a universal quantum gate set, it only needs to incorporate a single non-Clifford gate. The EPR scheme achieves this by including a T gate.

By definition, non-Clifford gates do not commute with Pauli operators and as such evaluating a T gate homomorphically is more challenging. More specifically,

$$\mathsf{T}\,\mathsf{X}^a\,\mathsf{Z}^b = \mathsf{X}^a\,\mathsf{Z}^{a \oplus b}\,\mathsf{P}^a\,\mathsf{T}. \tag{5.57}$$

That is, an application of T introduces a conditional phase gate P dependent on the application of X upon the application of T gate. This observation has also been made by Childs in [Chi05]. The evaluation algorithm could be made to correct the P gate, when $a = 1$, and as long as the evaluator does not know whether the correction has been applied or not the security holds. It is important

to note that the solution in [Chi05] involves interaction between the client and server during the evaluation. The solution proposed by [BJ15] involves delaying the correction induced by the T gate by using entanglement[19]. Specifically, the EPR scheme makes use of a quantum gate teleportation gadget known as "T-gate gadget", as shown in Figure (4). The correctness of T-gate gadget is shown in [BJ14]. Due to the fact that the complexity of the decryption depends on the number of T gates, the EPR scheme is not compact.

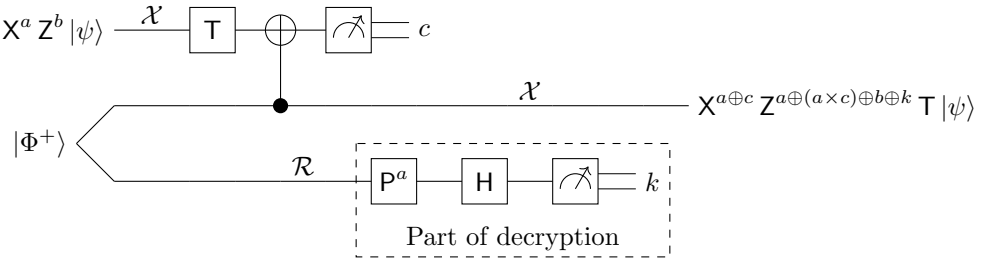


Figure 4: T-gate gadget

The important observation from the T-gate gadget is its effect at the end of the decryption procedure. Abstractly, $\mathsf{T}\,\mathsf{X}^g\,\mathsf{Z}^h\,|\psi\rangle$ before the evaluation is transformed into $\mathsf{X}^{g'}\,\mathsf{Z}^{h'}\,\mathsf{T}\,|\psi\rangle$ after decryption, where g, h, g', h' are polynomials in one or more variables. In particular, g' and h' gain new variables c and k, compared to g and h. The functionality of the T-gate gadget not only allows the T gate to commute with Pauli operators but also ensures that this is done securely, so the evaluator remains unaware of whether a P gate has been applied. To reiterate, the security of the T-gate gadget within the EPR scheme follows from the conditional application of P^a by the client during the decryption, ensuring that the server does not learn the value of a, while still satisfying the correctness of the protocol.

It is important to highlight the exponent of the Z gate at the end of the T-gate gadget: $a \oplus (a \times c) \oplus b \oplus k$. The operators $\times$ and $\oplus$ necessitate a classical FHE scheme, as they involve two operators, unlike the CL scheme, which only requires handling $\oplus$. Furthermore, the variable k is not known during the evaluation and will be measured by the client during the decryption process.

Next, we will define the EPR scheme. We remark that the key generation $\mathsf{EPR.Gen}$, and encryption $\mathsf{EPR.Enc}$ are defined exactly as $\mathsf{CL.Gen}$ and $\mathsf{CL.Enc}$, respectively. We will proceed to define $\mathsf{EPR.Eval}$ and $\mathsf{EPR.Dec}$.

Evaluation. $\mathsf{EPR.Eval}^{\mathsf{C}}$: Let C represent a universal quantum circuit consisting of ℓ layers of Clifford+T gates in terms of $\{\mathsf{I}, \mathsf{X}, \mathsf{Z}, \mathsf{H}, \mathsf{P}, \mathsf{CNOT}\} \cup \{\mathsf{T}\}$. This circuit is denoted as $\mathsf{C} = \mathsf{C}_\ell \circ \mathsf{C}_{\ell-1} \circ \cdots \circ \mathsf{C}_2 \circ \mathsf{C}_1$. The aim is to evaluate the circuit C on the quantum state ρ. Label wires holding the information of ρ as $\mathcal{X}_i$ for $1 \leq i \leq n$ where n is the size of the message ρ. For each Clifford

[19] In this work, we assume that homomorphic encryption is a non-interactive protocol. For an interactive QHE protocol that does not require shared entanglement, refer to [Lia15].

gate in the circuit, where $1 \leq j \leq \ell$ apply the gate to the corresponding wires and update the keys according to the CL scheme. Specifically, if $\mathsf{C}_j^{(i)}$ is a single-qubit Clifford gate, apply it to $\mathcal{X}_i$. If $\mathsf{C}_j^{(i,i+1)}$ is a CNOT gate, then apply it to $\mathcal{X}_i$ and $\mathcal{X}_{i+1}$.

Let R denote the number of T gates in C. For each T gate, use the T-gate gadget shown Figure (4). After performing the measurement on $\mathcal{X}_i$ wire, relabel the first half of the EPR pair as $\mathcal{X}_i$, and the second half as $\mathcal{R}_t$, where $1 \leq t \leq R$. The $\mathcal{R}_t$ wire then is returned to the client for decryption. Additionally, to determine the placement of the tth T in terms of its acting wire i and the layer j, we define the function π. This function takes t as input and outputs the tuple (i, j) that specifies the ith wire and jth layer:

$$\pi : \{1, \ldots, R\} \to \{1, \ldots, n\} \times \{1, \ldots, \ell\} \tag{5.58}$$

$$\pi(t) = (i, j). \tag{5.59}$$

The evaluation procedure is as follows.

(1) Let $N = \{1, \ldots, n\}$. Set $V \coloneqq \{a_{(i,0)}, b_{(i,0)}\}_{i \in N}$, and $\forall i \in N$, set $f_{(a,i,0)}, f_{(b,i,0)} \in \mathbb{F}_2[V]$ as $f_{(a,i,0)} \coloneqq a_{(i,0)}$, $f_{(b,i,0)} \coloneqq b_{(i,0)}$.

(2) Let $\mathsf{C}_j^{(i)}$ denote the single-qubit gate acting on wire i in the jth layer of the circuit C, and let $\mathsf{C}_j^{(i,i+1)}$ denote the $\mathsf{CNOT}_{i \to i+1}$ gate acting on wires i and $i+1$ in the jth layer. Apply each gate sequentially by iterating over the layers $j = 1$ to ℓ and then within each layer, follow the order of single or two-qubit gates, using the appropriate gadget depending on whether it is a Clifford gate or a T gate.

(3) Let N be the set of indices of the output wires. Let $\mathcal{V}$ be the set of labels

$$\mathcal{V} = \{(s, i, j) : (s, i, j) \in \{a, b\} \times N \times \{1, 2, \ldots, \ell\}\}. \tag{5.60}$$

For each $\alpha \in \mathcal{V}$, we want to homomorphically evaluate f_α to obtain the encrypted keys. By homomorphic properties and linearity of key update rules, we have $f_\alpha = f_\alpha^k + f_\alpha^{ab}$ where $f_\alpha^k \in \mathbb{F}_2[k_1, \ldots, k_R]$ and $f_\alpha^{ab} \in \mathbb{F}_2[a_1, \ldots, a_n, b_1, \ldots, b_n]$ (since c is a known constant). We can only evaluate the part of f_α that is in the known variables $\{a_{(i,j)}, b_{(i,j)}\}_{(i,j)}$ — the $\{k_{(i,j)}\}_{(i,j)}$ are unknown. Compute $\widetilde{f_\alpha^{ab}} \coloneqq \mathsf{CHE}.\mathsf{Eval}_{evk}^{f_\alpha^{ab}}\left(\tilde{a}_{(1,\ell)}, \ldots, \tilde{a}_{(n,\ell)}, \tilde{b}_{(1,\ell)}, \ldots, \tilde{b}_{(n,\ell)}\right)$ in the last layer of the circuit when $j = \ell$.

(4) The evaluator outputs:

(a) The $n = |N|$ qubit registers $\{\mathcal{X}_i : i \in N\}$ corresponding to the encrypted output of the circuit;

(b) The R qubit registers $\mathcal{R}_1, \ldots, \mathcal{R}_R$ corresponding to auxiliary states created by T-gate gadgets;

(c) The polynomials $\{f^k_\alpha\}_{\alpha\in\{\{a\}\times\pi(\{1,\ldots,R\})\}} \subset \mathbb{F}_2[k_1,\ldots,k_R]$ and the homomorphically evaluated polynomials $\{\widetilde{f^{ab}_\alpha}\}_{\alpha\in\{\{a,b\}\times N\times\{\ell\}\}}$.

Decryption. $\mathsf{EPR}.\mathsf{Dec}$: The client will perform the decryption component of the T-gate gadget by measuring the R auxiliary wires and proceed to decrypt the encrypted evaluated message. Formally:

(1) For $t = 1\ldots,R$:

(a) Decrypt $\mathsf{CHE}.\mathsf{Dec}_{sk}\left(\widetilde{f^{ab}_{(a,\pi(t))}}\right)$ to obtain $f^{ab}_{(a,\pi(t))}$.

(b) Compute $a_{\pi(t)} \coloneqq f^k_{(a,\pi(t))}(k_1,\ldots,k_{t-1}) \oplus f^{ab}_{(a,\pi(t))}$ and then apply $\mathsf{H}\,\mathsf{P}^{a_{\pi(t)}}$ to $\mathcal{R}_t$.

(c) Measure $\mathcal{R}_t$ to get k_t.

(2) Let N be the set of indices of the output registers. For $i \in N$:

(a) Decrypt $\mathsf{CHE}.\mathsf{Dec}_{sk}\left(\widetilde{f^{ab}_{(a,i,\ell)}}\right)$ and $\mathsf{CHE}.\mathsf{Dec}_{sk}\left(\widetilde{f^{ab}_{(b,i,\ell)}}\right)$ to obtain $f^{ab}_{(a,i,\ell)}$ and $f^{ab}_{(b,i,\ell)}$, respectively.

(b) Compute $a_{(i,\ell)} \coloneqq f^k_{(a,i,\ell)}(k_1,\ldots,k_t) \oplus f^{ab}_{(a,i,\ell)}$ and $b_{(i,\ell)} \coloneqq f^k_{(b,i,\ell)}(k_1,\ldots,k_t) \oplus f^{ab}_{(b,i,\ell)}$.

(3) To each register $\mathcal{X}_i$, apply $\mathsf{QOTP}.\mathsf{Dec}_{a_{i,\ell},b_{i,\ell}}$ and output registers $\mathcal{X}_1,\ldots,\mathcal{X}_n$.

This concludes the section on the EPR scheme, a quantum homomorphic encryption scheme for the universal quantum gate set. Next, we will develop the tools required for simulating the EPR scheme.

6 Simulation of EPR

In this section, we study the simulation of the EPR scheme, which has been developed as a package named `py-qhe-epr` found in [Gan24]. For the purposes of self-containment and completeness, the most-up-to date version of the package's source code at the time of writing this work is included in Appendix E.

In Section 6.1, we discuss the concept of simulation as a scientific method. Then, in Section 6.2, we introduce the design choices that facilitated the simulation of the EPR scheme, including the choice of programming language and the existing libraries. In Section 6.3, we examine the structure of the package and the various modules that make up the `py-qhe-epr` package. Section 6.5 addresses additional challenges encountered during the development and the solutions implemented to overcome them. Finally, in Section 6.6, we report on the results of running the simulation of the EPR scheme and analyze our findings, providing insights and evidence that the primary computational cost arises from simulating quantum processes, while the classical components contribute minimally to the overall performance.

This section is based on joint work with Broadbent and Paddock [GPB24].

6.1 Simulation as a Scientific Method

At the heart of every scientific endeavour is the understanding of the Universe and the physical laws that govern it, whether in physics, chemistry, biology, mathematics or any other field of science. A curious scientist makes observations in the real world and then questions why a certain phenomenon occurred the way it did. After studying the subject, they develop a reasonable hypothesis or model to explain the phenomenon. The next step involves verifying the model in a controlled environment, which can either be done through physical experiments, simulations or other testing methods.

Each method has its own benefits and downsides. Conducting accurate and precise physical experiments can be costly and may not be easily replicated. However, when done correctly, they provide higher confidence as they occur in the real physical world. For instance, the Higgs boson was first introduced in the 1960s [Hig64, EB64, GHK64] but was not confirmed until 2012 [ATL12]. The delay between theory and confirmation highlights the difficulty of conducting complex experiments, which sometimes depends on the available technology at the time.

On the other hand, simulations test theoretical models with fewer resources and are more reproducible than physical experiments. For instance, a flight simulator can train pilots without the need for operating an actual plane and incurring costly fuels expenses, all within a safe environment. The limitation is that a flight simulator cannot fully replicate potential real-life scenarios such as heavy storms, bird strikes or other unpredictable parameters.

6.1.1 Quantum Simulation on Classical Machines

In the study of quantum mechanics, physicists realized that the computations required to understand quantum phenomena are extremely demanding, as pointed out by Richard Feynman in [Fey82]. He stated that "nature isn't classical... if you want to make a simulation of nature, you'd better make it quantum mechanical." This insight was one of the key motivations behind the development of quantum computers. However, building quantum computers presents its own set of challenges such as mitigating noise-induced information loss, preventing qubit decoherence, maintaining high-quality qubits, and scaling up the number of qubits [Pre18]. Currently, we are in the *Noisy Intermediate-Scale Quantum* (NISQ) era, and there is still a long journey ahead to build a powerful quantum computer.

Another intermediate solution to the challenge of building a powerful quantum computer is the use of *quantum simulators*. These are classical computers that simulate quantum circuits as if they were running on a quantum computer [Ion24]. Nevertheless, due to the limitation of classical computers and the exponential growth in the number of classical bits required to simulate qubits, classical computers can only simulate quantum circuits to a limited extent. It is understood that even simulating a 50-qubit algorithm on a classical computer is considered infeasible [Ion24].

To put things into perspective, it is estimated that factoring RSA-2048 would require 20 million noisy qubits [GE21][20]. Therefore, classical computers are well-suited for simulating small-scale applications of quantum computing. They can provide evidence for the viability of quantum computers and maintain interest in investing in quantum technologies. However, we should not expect them to replace quantum computers entirely[21].

6.2 Design Choices

Designing a software simulation requires making a series of choices, such as selecting programming languages and libraries, whether they are built-in or from external sources. In this section, we touch on these design choices.

6.2.1 Programming Language: Python

We have chosen to implement the EPR scheme in Python programming language, version 3.8 [Pyt24]. There are various reasons that make Python a desirable choice for implementing a QHE scheme. Python is a high-level programming

[20]The subject of resource estimation is a controversial topic. For instance, one manuscript estimated that RSA-2048 could be broken with at least 372 qubits [YTW+22]. This claim was later challenged in [KY23] citing issues with scalability and the difficulty of factoring "random 80 bit integers and beyond". One should exercise caution when studying the literature on resource estimates and preferably refer to peer-reviewed publications.

[21]An interested reader who would like to know more about the simulation of quantum computing and explore surveys of simulators is encouraged to consult references [JCJ14, YSE23].

language known for its readable and straightforward syntax. Additionally, Python has a large and active community, extensive documentation, and a wide range of well-maintained libraries. It is reasonable to think that software developed in Python will remain functional and maintainable in the foreseeable future. Furthermore, the code developed in this work has been designed to be likely to remain compatible with future versions of Python, which is an important consideration for the initial development of an open-source QHE simulation package. In the context of quantum computing, many important quantum libraries, such as Google's Cirq [Cir24], IBM's Qiskit [JATK+24] and Xanadu's PennyLane [BIS+22] are Python-based. Therefore, Python is a natural choice for developing the software simulation of the EPR scheme.

6.2.2 Dependencies

To implement a simulation of the EPR scheme in Python, we rely on existing packages to handle both quantum operations, such as running quantum circuits, and classical operations, such as performing classical homomorphic encryption scheme.

6.2.2.1 Linear Algebra

We approached this work by simulating quantum circuits using matrices and linear algebra. A quantum gate can be represented by a matrix, and a density matrix, as the name suggests, is also a matrix. Tensor products are operations involving matrices. Consequently, we chose to perform these operations using a package that handles linear algebra and matrix operation robustly. The NumPy package [HMvdW+20] is specifically designed for this purpose. For example, the tensor product of matrices A and B, which results in $A \otimes B$, can be computed using the `kron(A,B)` function, and the matrix multiplication can be performed using `matmul` function or the `@` operator from NumPy.

6.2.2.2 Symbolic Computation

Implementing a theoretical work often involves addressing underlying issues that may not be immediately obvious. A challenging component of the implementation was carrying out the conditional phase gate used in the T-gate gadget. At first glance, performing P^a as part of the decryption component of the T-gate gadget may seem straightforward as only the b component of the homomorphic key update for T appears to contain any yet-to-be-measured bits k. However, this is not the case, as unmeasured bits can appear in the a component of the key. Explicitly, decryption following homomorphic evaluation of the circuit $\mathsf{T} \circ \mathsf{H} \circ \mathsf{T}$, shown in Figure (5a), would result in a k value in the a component before the application of the second T, since the H gate swaps the components a and b. Similarly, one might think that the key components of an encrypted qubit on a wire containing only Clifford gates will not contain these unmeasured values of k. However, this is not necessarily the case either. For instance,

the circuit $\mathsf{CNOT} \circ (\mathsf{I} \otimes \mathsf{T})$, shown in Figure (5b), will result in unmeasured k value to be "spilled over" to the b component of the first qubit at the end of the evaluation, despite no T being applied on the first qubit. These examples illustrate the important role of symbolic computation for keeping track of the values in the key updates. In our case, this was achieved through the use of the SymPy package [MSP+17] in combination with Python's built-in `re` package, which is used for searching text based on patterns. In the simulation of EPR, we make no assumptions about the input circuit, given each gate is chosen from the set of permissible quantum gates. Consequently, it was essential to develop a robust method for evaluating the key polynomials to achieve the objectives of this project.

(a) $\mathsf{T} \circ \mathsf{H} \circ \mathsf{T}$ circuit (b) $\mathsf{CNOT} \circ (\mathsf{I} \otimes \mathsf{T})$ circuit

Figure 5

6.2.2.3 FHE Scheme

Another crucial design choice was the selection of the classical FHE scheme. It is important to acknowledge the practical limitations of FHE implementations, as highlighted in [ABC+15], "It is fair to say that FHE mostly exists on paper. However, there also exist implementations", and in [FV12], "As to whether any of these proposals is really practical, the answer is simply "no"." Therefore, we approached the search for an FHE scheme with the understanding that it would primarily serve for demonstration purposes in our simulation. This work illustrates that while practical FHE implementations may not currently exist, they have potential use cases in quantum settings, especially within the constraints of the NISQ era.

Given that we decided to perform this simulation with Python, we focused on FHE implementations in Python. We opted for the implementation found in [Era20a] with a package named `py-fhe`, which was developed as part of the same student's master's thesis [Era20b]. The `py-fhe` package includes the Brakerski-Fan-Vercauteren (BFV) FHE scheme developed in [Bra12] and [FV12]. We used the `py-fhe` along with Python's built-in `copy` module to create independent copies of ciphertexts without altering originals.

We conclude the section on design choices with Table (7) listing the packages used for the simulation of the EPR scheme.

Package Name	Version
NumPy	1.24.4
SymPy	1.6.2
re	Built-in
py-fhe	1.0
copy	Built-in

Table 7: List of packages and their versions

6.2.3 Documentation

The `py-qhe-epr` library contains extensive documentation for each function, clearly outlining its purpose, the required input parameter(s), and the expected output(s). Additionally, in-line comments are provided to offer more insights. A great effort has been made to ensure the package adheres to Python's official guidelines, as outlined in the Python enhancement proposals (PEP) 8 style guide [vRWC13]. For instance, function names are descriptive, using lowercase words separated by underscores.

Given that this package aims to serve as a software simulation library for the quantum community and introduces the first open-source software for quantum homomorphic encryption, considerable effort has been made to ensure the code is readable for future use.

6.3 Implementation Structure

The `py-qhe-epr` package consists of seven modules designed to facilitate the implementation of the EPR scheme. Each module addresses a specific aspect of simulating the EPR scheme. Figure (6) shows the connection and dependencies of the seven modules that simulate the EPR scheme.

Below, we provide a high-level summary of the seven modules that make up the `py-qhe-epr` package, with implementation details deferred to Section 6.4:

- `che_initialization`: Importing the BFV scheme from the `py-fhe` package, `che_initialization` module manages the classical homomorphic encryption component of the EPR scheme. It encrypts the classical keys of the quantum one-time pad using the public key during encryption, applies homomorphic key updates during evaluation, and decrypts the classical keys with the secret key in the decryption phase.

- `basic_quantum_operations`: This module provides a range of functions and variables essential for quantum simulation. It defines the quantum gates in the universal gate set $\{\mathsf{I}, \mathsf{X}, \mathsf{Z}, \mathsf{H}, \mathsf{P}, \mathsf{CNOT}, \mathsf{T}\}$ and provides functions for creating various quantum states. These include the arbitrary quantum states with amplitudes α and β using `gen_qubit` function, as well as specific states like the zero state $|0\rangle$, the one state $|1\rangle$ and the EPR pair through functions such as `zero_state, one_state, gen_epr`, respectively.

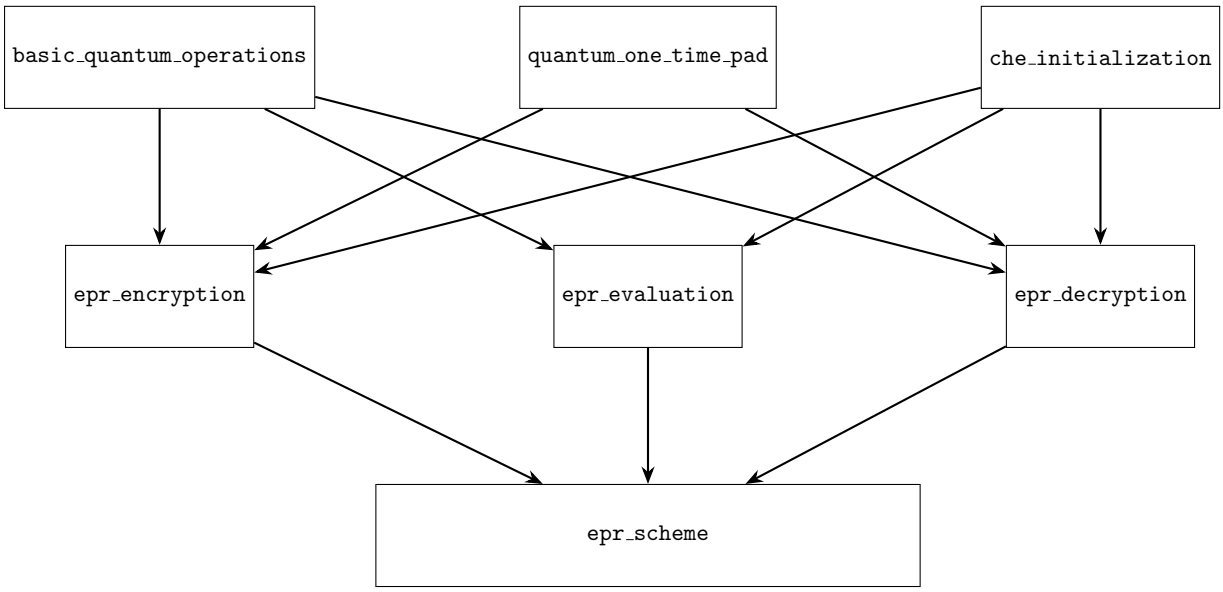


Figure 6: Connections and dependencies between different modules, with arrows indicating which modules depend on others.

Additionally, it includes quantum operations like `measurement` and `post_measurement_state`, following the mathematical formalism detailed in Section 3.

- `quantum_one_time_pad`: The `quantum_one_time_pad` module defines functions for quantum one-time pad encryption and decryption, which will be used during encryption and decryption of the EPR scheme.
- `epr_encryption`: This module encrypts the input message $|\psi\rangle$ using the classical secret keys (a, b) of the QOTP. Additionally, it encrypts the classical keys (a, b) to obtain $(\tilde{a}, \tilde{b})$.
- `epr_evaluation`: The `epr_evaluation` module carries out the necessary computations on the encrypted quantum data and updates classical keys homomorphically.
- `epr_decryption`: Handling the decryption component of the EPR scheme, the `epr_decryption` module decrypts the classical keys, and if necessary, runs the the decryption component of the T-gate gadget and traces out unnecessary wires, before decrypting the evaluated state using QOTP.
- `epr_scheme`: Integrating all the modules, the `epr_scheme` module encrypts, evaluates and decrypts the input message and then outputs the density matrix of the decrypted evaluated state.

6.4 Implementation Details

We now provide a detailed description of the `py-qhe-epr` package, closely following the documentation provided in [Gan24].

Key Generation — The implementation uses the key generation function `BFVKeyGenerator` from the `py-fhe` scheme implemented in [Era20a]. Using this package, we generate a public key pk and a secret key sk to be used in the encryption and decryption of the classical one-time-pad keys[22].

Encryption — Given an input n-qubit quantum state $|\psi\rangle$ and a pair of randomly generated n-bit keys (a, b), the client uses the public key pk to produce the encryption pair $(\tilde{a}, \tilde{b})$ along with an encrypted state $|\tilde{\psi}\rangle$, where the state $|\tilde{\psi}\rangle$ is obtained from the quantum one-time-pad, which takes as input the pair (a, b).

Homomorphic Evaluation — The server's description of the quantum circuit is represented by $\mathcal{C}$, a list of strings in the alphabet $\{\mathsf{I}, \mathsf{X}, \mathsf{Z}, \mathsf{H}, \mathsf{P}, \mathsf{CNOT}, \mathsf{T}\}$. Each element of the list corresponds to a layer of the quantum circuit and each symbol in the string represents the gate applied to qubit(s) i in the jth layer, namely $\mathcal{C}_j^{(i)}$, where $1 \leq j \leq \ell$ ranges through the ℓ layers of the circuit, and $1 \leq i \leq n$ for the n-qubits. Before evaluating the circuit, the server appends R EPR pairs to the input of the computation, where R is the total number of T in $\mathcal{C}$. Once completed, the server applies a series of SWAP gates so that all the wires associated with EPR pairs used for computing any T gates appear in the required order (see Figure (4)). The server applies the circuit $\mathcal{C}$ layer by layer. Each layer $\mathcal{C}_j$ is implemented by a unitary $\mathcal{L}_j$ on $n + 2R$-qubits, consisting of at most n non-identity gates.

The server implements $\mathcal{L}_j$ gate by gate, inserting identity gates where required. If the gate $\mathcal{C}_j^{(i)}$ is Clifford, then the keys $\left(\tilde{a}_j^{(i)}, \tilde{b}_j^{(i)}\right)$ are updated according to Table (6). On the other hand, if the gate $\mathcal{C}_j^{(i)}$ is a T gate, the server stores the ciphertext $\tilde{a}_j^{(i)}$ in a list $\tilde{\mathcal{P}}$, and applies the T gate followed by $\mathsf{CNOT}_{2\to1}$ and measures the wire that initially held the encrypted state obtaining the bit $c_j^{(i)}$ (see Figure (4)). Storage of the ciphertext in the list $\tilde{\mathcal{P}}$ will be used as part of the decryption. The server updates the wire labels holding the evaluated qubit as the encrypted qubit has been "teleported" to the 1st wire of the EPR pair.

To make the homomorphic update for the T, the measured bit $c_j^{(i)}$ is encrypted using the BFV scheme public key pk. The resulting ciphertext $\tilde{c}_j^{(i)}$ is stored in a list $\tilde{\mathcal{M}}$. Now, the server performs the homomorphic evaluation by computing $\tilde{a}_{j-1}^{(i)} \mapsto \tilde{a}_{j-1}^{(i)} \oplus \tilde{c}_j^{(i)}$ and $\tilde{b}_{j-1}^{(i)} \mapsto \tilde{a}_{j-1}^{(i)} \oplus \left(\tilde{a}_{j-1}^{(i)} \cdot \tilde{c}_j^{(i)}\right) \oplus \tilde{b}_{j-1}^{(i)}$, which accounts for the portion of the homomorphic T gate key update with *known* variables. To account for the *unknown* variables in the homomorphic T gate key update, the server also performs a symbolic computation tracking the remaining unknown values $k_j^{(i)}$. Here the encrypted keys are treated as unknown symbolic variables. Specifically, for a T gate, $\hat{a}_{j-1}^{(i)} \mapsto \hat{a}_{j-1}^{(i)} \oplus \hat{c}_j^{(i)}$ as usual.

[22] More information about parameter selection can be found in [Era20b].

However, $\hat{b}_{j-1}^{(i)} \mapsto \hat{a}_{j-1}^{(i)} \oplus \left(\hat{a}_{j-1}^{(i)} \cdot \hat{c}_j^{(i)}\right) \oplus \hat{b}_{j-1}^{(i)} \oplus \hat{\boldsymbol{k}}_{\boldsymbol{j}}^{(\boldsymbol{i})}$. We use the notation $\hat{(\cdot)}$ to denote symbolic versions of the variables. We need to perform symbolic updates for the Clifford gates as well, according to Table (6) as expected; however, as we discussed in Section 6.2.2.2, the unknown values of $k_j^{(i)}$ can propagate into the keys updates of subsequent layers of Clifford key updates.

In a list $\mathcal{S}$, the server stores the symbolic value of $\hat{a}$ and $\hat{b}$ at the end of the circuit. The list also includes a record of each value of $\hat{a}$ before the application of a T gate. Likewise, the order in which T gates appear in the circuit is transcribed in a list $\mathcal{T}$. Both lists are used in the decryption.

Decryption — Using the secret key sk, the client decrypts the ciphertexts $(\tilde{a}, \tilde{b})$ as well as the encrypted bits in $\tilde{\mathcal{M}}$ and $\tilde{\mathcal{P}}$. We denote these decrypted lists as $\mathcal{M}$ and $\mathcal{P}$, respectively. The client begins decrypting by the order specified in the list $\mathcal{T}$. If $\mathcal{T}$ is empty, the client can recover the evaluated quantum state by applying the quantum one-time pad using the decrypted keys (a, b). Otherwise, decryption proceeds as follows: the first conditional phase gate exponent does not depend on any unmeasured values. Therefore, in this case the client can perform the conditional $\mathsf{P}^{a_j^{(i)}}$ by reading the value of $a_j^{(i)}$ from the decrypted values of $\mathcal{P}_j^{(i)}$. The client applies the H gate and then proceeds to measure $k_j^{(i)}$, storing this value in a list $\mathcal{K}$ (see Figure (4)).

For the decryption of any subsequent T gates, the client may need to apply corrections to the values in $\mathcal{P}$, since those values could depend on yet-to-be-measured values of k, which were unknown during the evaluation. To apply theses corrections, the client looks up the $\mathcal{S}_j^{(i)}$ entry holding the symbolic expression of the key updates. The client substitutes 0 for expressions involving $\hat{a}$ and $\hat{b}$ and the corresponding values of $\hat{c}$ and $\hat{k}$ from the appropriate entries of lists $\mathcal{M}$ and $\mathcal{K}$ to evaluate the polynomial, which outputs the bit necessary to correct the exponent value of the P gate. Only after this correction, can the client apply the decryption part of the T-gate gadget. The new measured $k_j^{(i)}$ value is then appended in the $\mathcal{K}$ list. Upon completing this process for all the T gates, the client repeats the classical bit correction procedure to obtain the decrypted a and b values. Lastly, the client traces out all the unnecessary wires. It is only now that the client can apply the quantum one-time-pad to obtain the decrypted quantum state.

6.4.1 Minimal Working Example

While the available code in [Gan24] contains a tutorial on getting started with the `py-qhe-epr` package, we provide a minimal working example to showcase the package's input and output.

py-qhe-epr Code Example

```
1 %run epr_scheme.ipynb
2
3 a = [0]
4 b = [1]
5 psi = zero_state()
6 circuit = ['X']
7
8 print(epr_quantum_homomorphic_scheme(psi, a, b, circuit))
```

Program Output

```
1 [[0 0]
2  [0 1]]
```

In this example, we first load the `epr_scheme` module. We then choose the classical keys a, b, which can be generated using cryptographically secure random number generators provided by specialized Python packages. For the sake of demonstration, we choose $a = 0$ and $b = 1$, which will be used during the QOTP procedure. We also specify the input message $|\psi\rangle = |0\rangle$ and the circuit we would like to apply. In line 8, the function `epr_quantum_homomorphic_schme` takes $|\psi\rangle, a, b$ and the circuit X as input. Internally, the function performs QOTP on $|\psi\rangle$, encrypts classical keys (a, b) using the BFV scheme, evaluates the circuit homomorphically using the density matrix formalism, and updates the classical keys. Finally, it decrypts the classical keys and decrypts the encrypted message using the QOTP. In density matrix formalism, $\mathsf{X}|0\rangle\langle 0|\mathsf{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$, which corresponds to the output.

6.5 Implementation Challenges

In Section 6.2, we discussed the challenges involved in evaluating the polynomial of key updates and how symbolic computation was used to address the issue. Here we provide additional discussions of the implementation. We highlight some of the more challenging components, as well as provide insight into some of the more complex parts of the software design.

Wire Relabelling — The original EPR scheme in [BJ15] assumes that the appended Bell pairs appear adjacent to the corresponding qubit(s). However, in practice one needs to account for the relabelling of wires. We achieve this by performing SWAP gates to place the EPR pairs adjacent to the corresponding qubit as in Figure (4). Although we perform SWAP operations as needed, one could alternatively track these within the implementation and synthesize the required permutation of the output registers before decryption. In either case, one could be concerned that the SWAP gates need to be performed

homomorphically. SWAP gate is a Clifford and the key updates match our intuition. That is, the keys $(a_1, b_1), (a_2, b_2)$ get mapped to $(a_2, b_2), (a_1, b_1)$ after a SWAP is performed[23]. The use of SWAP gates in the implementation highlights one of the important challenges in translating a theoretical result, where auxiliary EPR pairs and the corresponding qubits are arranged in a specific order by default, into implementable code, which requires a programmatic approach to achieve that configuration in practice.

As an example, consider an n-qubit system $|\psi\rangle$ where a T gate is applied on the first qubit and a single-qubit Clifford unitary U is applied on the remaining wires as shown in Figure (7a). In the context of the `py-qhe-epr` package, we first append an EPR pair to create the evaluation circuit, as shown in Figure (7b). Next, SWAP gates are used to reorder wires, placing the appended EPR pair immediately after the first qubit, as illustrated in Figure (7c), in line with the EPR scheme described in [BJ15]. To ensure the correctness of the implementation, it is crucial that the added EPR pairs for a qubit appear before those for subsequent qubits.

Generalized Controlled-NOTs — In the software implementation, we develop tools for tracking the wire(s) where gates were applied. This is a practical way to account for the fact that the T-gate gadget teleports the encrypted state to another wire. Tracking this information is crucial not only during the evaluation and decryption procedures but also in cases where multiple T gates are applied to the same qubit, particularly when the T-gate gadget requires a CNOT gate between non-adjacent wires of different EPR pairs. This is not possible with the standard $\mathsf{CNOT}_{2\to1}$. To address this, we constructed a function `controlled_not_constructor(control, target, number_of_qubits)` for performing CNOTs between non-adjacent wires. In the case `target` $<$ `control`, the function creates the following n-qubit gate

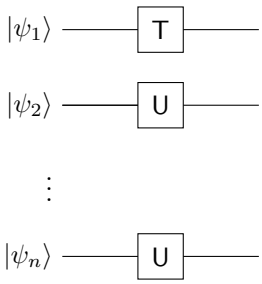
$$\begin{aligned}\mathsf{CX}_{c,t,n} = &\left[\mathsf{I}^{\otimes(c-1)} \otimes |0\rangle\langle 0| \otimes \mathsf{I}^{\otimes(n-c)}\right] \\ &+ \left[\mathsf{I}^{\otimes(t-1)} \otimes \mathsf{X} \otimes \mathsf{I}^{\otimes(c-t-1)} \otimes |1\rangle\langle 1| \otimes \mathsf{I}^{\otimes(n-c)}\right],\end{aligned} \tag{6.1}$$

where we use the convention that $\mathsf{I}^{\otimes 0} = 1$.

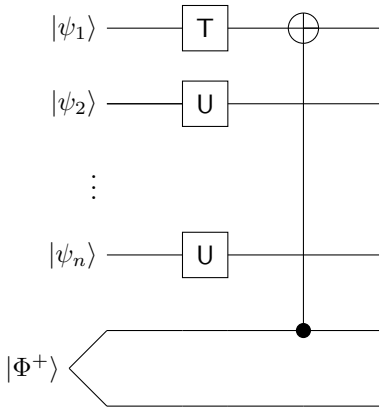
As an example, consider the homomorphic evaluation of $\mathsf{T} \circ \mathsf{T}$ as shown in Figure (8a)[24]. In the context of the `qy-qhe-erp`, the evaluation circuit corresponds to one shown in Figure (8b). Specifically, the second application the T-gate gadget requires applying $\mathsf{CNOT}_{4\to2}$ within a 5-qubit system, with the corresponding unitary operation being $\mathsf{CX}_{4,2,5}$.

[23] The correctness of the key update rule for the SWAP gate is shown in Appendix D, where the SWAP gate is decomposed into $\mathsf{CNOT}_{1\to2}$ and $\mathsf{CNOT}_{2\to1}$ gates, and the key update rules of the decomposition are tracked to show its correctness.

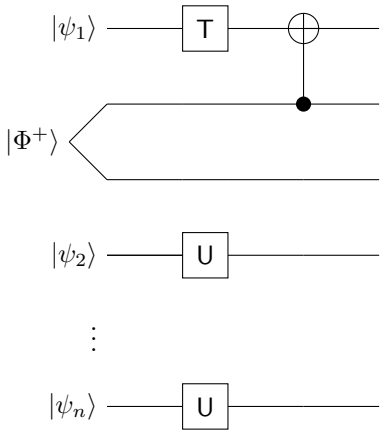
[24] While $\mathsf{T}^2 = \mathsf{P}$, this example is only intended to motivate the need for generalized CNOT gates.



(a) n-qubit system $|\psi\rangle$ with a T on the first qubit and U on the remaining qubits.



(b) EPR pair added in preparation for the evaluation circuit.



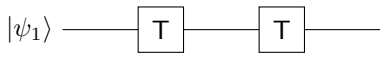
(c) Evaluation circuit after SWAP gates reorder wires, placing the EPR pair after first qubit.

Figure 7

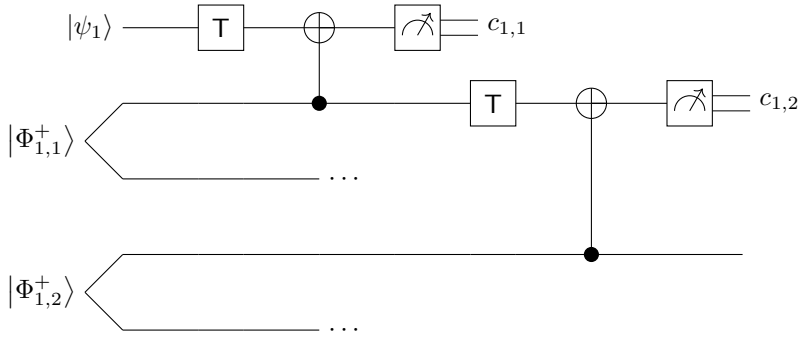
6.6 Test Cases and Analysis

In this section, we provide details of our test implementations and some analysis of our findings. Our simulations were conducted on an Intel core i9-10885H with 32GB RAM using Python 3.8. The security parameters for the BFV scheme include a polynomial ring of degree 8, a plaintext modulus of 17, and a ciphertext modulus of 8×10^{12}, as used by the example provided in `py-fhe`.

Figure (9) displays the average runtimes of the simulated EPR scheme for a circuit consisting of $R \in \{1, \ldots, 6\}$ T gates on a single qubit. The test aims to establish a benchmark for the cost of simulating the quantum operations by quantifying the time taken by the simulation of the EPR scheme with and without the homomorphic key updates. In the test runs without the encrypted



(a) $\mathsf{T} \circ \mathsf{T}$ circuit.



(b) The evaluation circuit for $\mathsf{T} \circ \mathsf{T}$ circuit in the EPR scheme.

Figure 8

key updates[25], we assume the circuit is known to the client and all the updates are performed by the client during the decryption. We stress that this is not the case in the EPR scheme with the encrypted key updates outlined in Section 6.4. The data in Figure (9) from the circuits with 1 to 5 T gates are averages from a 10 sample runs, whereas in the 6 T gate circuit the average was only taken from 3 sample runs due to computational limitations. The error bars in Figure (9) represent a standard deviation.

Recall that if the n-qubit quantum circuit has R T gates, the scheme requires an additional $2R$ qubits. This implies that the simulation amounts to performing quantum operations on an $n + 2R$-qubit system, *i.e.* multiplying $2^{n+2R} \times 2^{n+2R}$ matrices, which requires an exponential number of operations in the parameters n and R. The exponential nature of operations manifests itself in Figure (9) by noting the marked increase in the computational time between the runs with 5 and 6 T gates, from around 1 minute to almost 1 hour and a half, respectively.

On its own, the data presented in Figure (9) is too coarse to capture the relationship between the cost of the classical and quantum operations. More specifically, we want to better understand how the classical components of the code, such as homomorphic key updates, wire tracking and symbolic computations contribute to the overall runtime of the simulation. Figure (10) displays the proportion of computational resources used with the classical homomorphic encryption. The data in Figure (10) affirms our suspicion that the

[25] Details of the EPR scheme with unencrypted keys are included in our software library [Gan24], including the implementation that was used to collect the data in Figure (9).

classical components of the code do not have much impact on the runtime of the simulation. As expected Figure (10) shows that as the number of T gates increase, a larger portion of the runtime is used in simulating the quantum operations.

Interestingly, Figure (10) shows that the decryption takes slightly more computational resources than the evaluation. A plausible explanation is by noting that for every T gate in the circuit, there are two gates applied during evaluation: a T followed by a CNOT, and two more in the decryption: a conditional P and an H. Additionally, in the decryption, the client needs to perform the quantum operation to trace out unnecessary wires. From the data in Figure (9) for the EPR scheme with encrypted keys, and the data in Figure (10), we can deduce that the runtime of the decryption grows as the number of T gates increases, providing evidence that the scheme is not compact.

The attentive reader will note that in the case of the 6 T gate experiment in Figure (9), on average the EPR scheme without encrypted keys took slightly longer than the EPR scheme with encryption. This is surprising since one expects the addition of HE to require more resources. Nonetheless, this anomaly can be explained by the fact that the classical operations use a negligible amount of time in comparison to the quantum operations as evident in Figure (10). This observation, along with the overlap of the error bars from the 5 and 6 T gate experiments (with and without encrypted keys) supports this claim. Variability of the runtime in each test can be affected by factors such as background tasks and CPU variability.

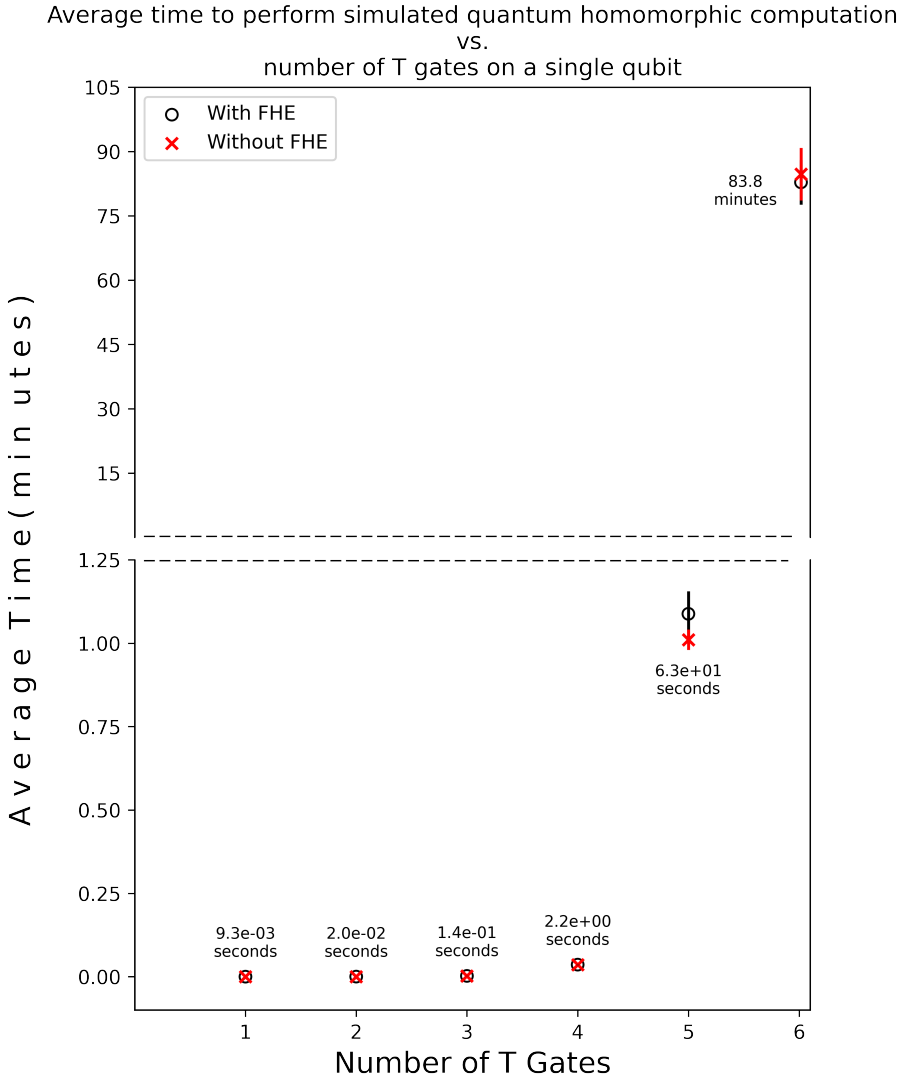


Figure 9: Runtime of the simulation of quantum homomorphic encryption based on the number of T gates with and without classical homomorphic encryption. The displayed runtime is the average of the two schemes.

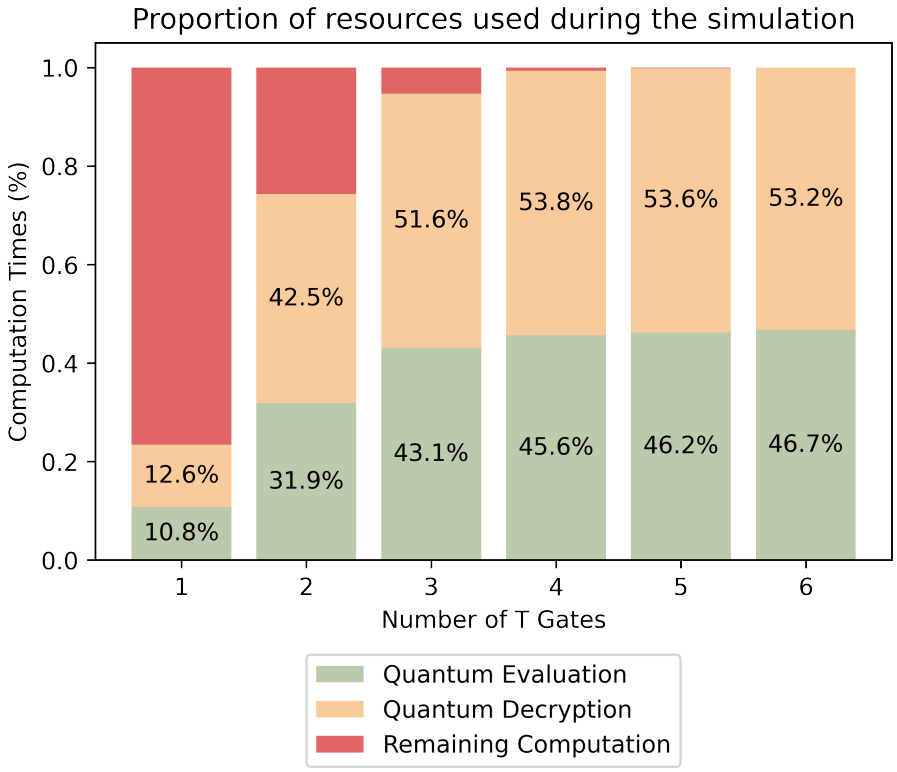


Figure 10: Proportion of computational resources used during the simulation. Remaining Computation includes any purely classical computations during the protocol, including evaluation and decryption, as well as the quantum one-time pad operation during the encryption.

7 Conclusion

In this work, we developed a Python implementation capable of simulating QHE. In particular, the software implementation of the EPR scheme from [BJ15] works for universal quantum circuits. We discussed the details and challenges of bridging the theoretical details of the scheme into practice. We provided some preliminary data which suggests that the cost of classical homomorphic encryption is nominal relative to the quantum simulation costs. We hope that the quantum community and enthusiasts will benefit from this library, towards advancing the frontiers of quantum cloud computing.

In this concluding section, we will propose improvements to the existing implementation in Section 7.1, and in Section 7.2, we will explore potential follow-up work and extensions that could stem from this work.

This section is based on joint work with Broadbent and Paddock [GPB24].

7.1 Improvements to the Implementation

There are several developments that could improve the performance of the simulation toolkit. The first of which could be improving the performance of the quantum operations. One method could be synthesizing the unitary circuit for an entire layer, so that all the gates in a layer are performed simultaneously, rather than one gate at a time. Another way would be to reduce the dimension of the quantum operations by tracing out the measured "extra" wires during the evaluation of the T-gate gadget. Doing this could reduce the wires from $n + 2R$ to just $n + R$ at the end of the evaluation, saving a factor of 2^R in the decryption runtime.

Further improvements are possible in other areas. For example, the current implementation requires the quantum circuit to be described using the quantum gate set $\{\mathsf{I}, \mathsf{X}, \mathsf{Z}, \mathsf{H}, \mathsf{P}, \mathsf{CNOT}, \mathsf{T}\}$. In particular, the description of the input circuit is restricted to CNOT gates acting on adjacent qubits. A useful improvement would be to extend this functionality to handle generalized controlled-NOT gates, where the control and target qubits can be specified on any wires in the circuit. While the internal mechanisms for implementing non-adjacent CNOT gates are already in place, as explained in Section 6.5, the input circuit description only supports CNOT gates between adjacent qubits. In other words, although non-adjacent CNOTs are supported within the T-gate gadget, the input circuit format is still designed to handle only adjacent CNOTs. To address this, the circuit description would need to explicitly allow for the specification of arbitrary control and target qubits. Implementing this change would require modifications to several subroutines across the `py-qhe-epr` package.

Moreover, the set of accepted quantum gates could be expanded to include additional quantum gates. The primary challenge in adding a new quantum gate lies in determining its correct key update rule. A straightforward solution would be to decompose the new quantum gate into a combination of gates for which the key update rules are already known, and then compose these rules to derive the correct key update rule for the new gate. An example of this technique is

provided in Appendix D, where the key update rule for the SWAP gate was found by decomposing it into the set $\{\mathsf{CNOT}_{1\to2}, \mathsf{CNOT}_{2\to1}\}$.

Another potential direction for exploration is to incorporate the `py-qhe-epr` package with existing quantum software frameworks like Cirq, PennyLane or Qiskit, introduced in Section 6.2.1, which are widely used platforms for quantum computing, helping it reach a broader audience within the quantum community.

7.2 Future work

While in Section 7.1, we focused on potential improvements and enhancing the efficiency of the existing implementation, in this section we explore alternative theoretical frameworks, allowing the current software to serve as a broader tool for theoretical exploration.

A next step would be to expand the `py-qhe-epr` library using techniques from follow-up works to the EPR scheme. For instance, in [DSS16] the EPR scheme was improved by introducing an alternative T-gate gadget which results in a compact levelled scheme for performing non-Clifford gates homomorphically. Implementing this new gadget would allow the simulation of QHE on quantum circuits with a polynomial number of T gates.

Verifiability is an important ingredient for secure delegated quantum computing. The addition of a verification procedure is a desirable feature, as often the client may want to delegate a quantum circuit to the server while ensuring that the output is correct. In [ADSS17] the QHE scheme of [DSS16] was made verifiable. Adding a verification component to the implementation would be an interesting next step.

Finally, the EPR scheme could be improved by incorporating additional security features, such as circuit privacy. Specifically, in the honest case, the client receives a qubit for each T gate to measure during the decryption procedure. If the server's objective is to keep the circuit private, as suggested in [DSS16], the server can take measures to maintain circuit privacy by adding randomization layers to the circuit. Future work towards including such features in the `py-qhe-epr` library could provide additional utility to the QHE scheme, particularly in the context of proprietary quantum algorithms.

8 Epilogue

The study of quantum information science requires mastery across multiple disciplines, including mathematics, physics and computer science, with a deep understanding of specific subfields within each. For example, in mathematics, familiarity with linear algebra, group theory, number theory and Hilbert spaces is beneficial—each of which is typically covered in an advanced undergraduate or graduate course.

Furthermore, quantum information science challenges our intuition, often yielding counter-intuitive results that have perplexed the greatest minds in human history, such as Albert Einstein. It also forces us to question our philosophical views towards life. For instance, if the act of measurement is an irreversible action, then does that imply that we live in a non-deterministic world? The deeper one delves into this subject, the more questions arise than answers. While I have found studying quantum information science to be extremely challenging, I have also found it to be a rewarding and humbling experience at the same time.

It is my hope that the challenges in unlocking the potentials of quantum information science will not be seen as impediments, but rather as a light guiding us toward the inspiration needed to overcome them, which is part of human's enduring endeavour. As us, humans, the inhabitants of the "pale blue dot" [The24e], strive to push the boundaries of knowledge in the quest of unraveling the mysteries of the Universe.

A Biographies

In this auxiliary section, we offer brief biographies of a few scientists mentioned in this thesis. We note that this list is neither exhaustive nor fully representative of all the scientists referenced. The scientists who have dedicated their lives to their fields of study are the unsung heroes of our time, who usually do not receive the recognition they deserve. The purpose of this section is to simply honour and acknowledge the contributions of not only the scientists listed here, but all those who made a positive impact on our world.

Charles Hermite (1822–1901) was a French mathematician who made major contributions to various fields of mathematics, including number theory and algebra. The Hermitian matrix, introduced in Section 2.3.7, is named after him.

Hermite was a mathematics professor in Paris, and was the first to prove that the number e is a transcendental number in 1873. Henri Poincaré, an influential figure in physics and mathematics, was one of Hermite's students. In his personal life, Hermite married Louise Bertrand, the sister of mathematician Joseph Bertrand [The24a, OR01].

William Kingdon Clifford (1845–1879) was an English mathematician and philosopher. Clifford gates, introduced in Section 3.2.2, are named after him. Despite his short life, passing away at age of 33, he made significant contributions to algebra, leading to the naming of a branch of algebra after him, now known as Clifford algebra. In 1871, Clifford was appointed to the chair of Mathematics and Mechanics at University College London, and in 1874, he was elected as a Fellow of the Royal Society [The24c, OR15].

Jacques Salomon Hadamard (1865–1963) was a French mathematician whose research spanned number theory, complex analysis and differential geometry, among others. Hadamard matrices, introduced in Section 3.2.2, are named after him. He also provided a proof of the prime number theorem, a fundamental result in number theory. Hadamard tragically lost two of his sons during World War I and his third son in the Second World War. In response to these personal losses, he devoted himself even more vigorously to mathematics and became an active figure in international peace movements. In his book *Psychology of Invention in the Mathematical Field*, Hadamard reflected on the nature of mathematical creativity and the workings of the mathematical mind [The24b, OR03].

Wolfgang Ernst Pauli (1900–1958) was an Austrian theoretical physicist, most notably recognized for his contributions to quantum physics. The Pauli gates and Pauli groups, introduced in Section 3.2.1, are named after him. He received the Nobel Prize in Physics in 1945. Pauli also conjectured the existence of neutral particles, now known as neutrinos [The24d, Nob24].

Anne Lise Broadbent is a Canadian mathematician specializing in quantum information science and quantum cryptography. She is one of the authors of the work titled *Quantum homomorphic encryption for circuits of low T-gate complexity*, in which the EPR scheme was introduced. Broadbent earned her PhD from the Université de Montréal under the supervision of renowned researchers Gilles Brassard and Alain Tapp. Among a multitude of papers, Broadbent has made significant contributions by publishing work on quantum blind computation, quantum homomorphic encryption and the verification of quantum computation, among others. Broadbent has been recognized with numerous awards, including the André Aisenstadt Prize in Mathematics in 2016, which honours outstanding young Canadian mathematicians [Bro19].

B Number Theory

Number theory is a branch of mathematics that studies integers and their properties, such as prime number and prime factorization. Number theory has important applications in cryptography. In Section 4.2.2, we introduce the RSA cryptosystem [RSA78], which is based on number-theoretic principles. This section aims to provide the minimal number-theoretic background necessary for describing the RSA cryptosystem by highlighting important definitions and theorems. The main reference for this section is [Ros11].

Definition B.1 (Greatest Common Divisor)**.**
The *greatest common divisor* of two integers a and b, which are not both 0, is the largest integer that divides both a and b. The greatest common divisor of a and b is written as $\gcd(a, b)$.

Example B.1. The positive divisors of 9 are $\{1, 3, 9\}$ and divisors of 6 are $\{1, 2, 3, 6\}$. Since $\max(\{1, 3, 9\} \cap \{1, 2, 3, 6\}) = \max(\{1, 3\}) = 3$, we conclude that $\gcd(9, 6) = 3$.

Pairs of integers that have no common divisors greater than 1 are of interest in number theory.

Definition B.2 (Relatively Prime)**.**
The integers a and b, with $a \neq 0$ and $b \neq 0$, are relatively prime if a and b have greatest common divisor $\gcd(a, b) = 1$.

By this definition, any pairs of prime numbers (p, q) are relatively prime, since $\gcd(p, q) = 1$.

Definition B.3 (Congruent Numbers)**.**
Let m be a positive integer. If a and b are integers, we say that a is *congruent to b modulo m* if $m \mid (a - b)$, where $m \mid (a - b)$ means m divides $(a - b)$. In this case, we write $a \equiv b \pmod{m}$.

Example B.2. $22 \equiv 7 \pmod{5}$, since $22 - 7 = 15$ and $5 \mod 15$.

Definition B.4 (Modular Inverse)**.**
Given an integer a with $\gcd(a, m) = 1$, an integer b such that $a \cdot b \equiv 1 \pmod{m}$ is called an *inverse of a modulo m*. In this case, we denote b as a^{-1}.

The definition of modular inverse requires $\gcd(a, m) = 1$. Therefore, not every integer has a modular inverse.

Example B.3. Since $\gcd(5, 7) = 1$, 5 has an inverse modulo 7. By inspection, we see that $5 \times 3 = 15 \equiv 1 \pmod{7}$. Hence, 3 is an inverse of 5 modulo 7 and $5^{-1} \equiv 3 \pmod{7}$.

Definition B.5 (Euler phi-function)**.**
Let n be a positive integer. The *Euler phi-function* $\phi(n)$ is defined to be the number of positive integers not exceeding n that are relatively prime to n.

Example B.4. The positive divisors of 12 are $\{1, 2, 3, 4, 6, 12\}$. Therefore, the positive non-divisors of 12 are $\{5, 7, 8, 9, 10, 11\}$, among which, $\{5, 7, 11\}$ have no common divisors with 12 other than 1. Additionally, $\gcd(1, 12) = 1$. Therefore, there are 4 numbers that are not exceeding 12 and are relatively prime to 12. We conclude that $\phi(12) = 4$.

Finally, we arrive at Euler's theorem, which is crucial for proving the correctness of the RSA cryptosystem.

Theorem B.1 (Euler's Theorem)**.**
If m is a positive integer and a is an integer with $\gcd(a, m) = 1$, then $a^{\phi(m)} \equiv 1 \pmod{m}$.

Instead of proving this theorem, we provide a sketch of proof. Let the set $R = \{r_1, r_2, \ldots, r_{\phi(m)}\}$ be the set of integers less than m that are relatively prime to m. The set R is known as a *reduced residue system modulo* m. Since $\gcd(r_i, m) = 1$ for each r_i with $1 \leq i \leq \phi(m)$, we have $\gcd(r_1 \cdot r_2 \cdots r_{\phi(m)}, m) = 1$. This implies that $r_1 r_2 \cdots r_{\phi(m)} \equiv 1 \pmod{m}$. By a theorem, the set $\{ar_1, ar_2, \ldots, ar_{\phi(m)}\}$ is also a reduced residue system modulo m, since $\gcd(a, m) = 1$. Thus,

$$ar_1 ar_2 \cdots ar_{\phi(m)} \equiv r_1 r_2 \cdots r_{\phi(m)} \equiv 1. \tag{B.1}$$

On the other hand, $ar_1 ar_2 \cdots ar_{\phi(m)} = a^{\phi(m)} r_1 r_2 \cdots r_{\phi(m)}$. Therefore, it follows that $a^{\phi(m)} \equiv 1 \pmod{m}$.

C Perfcet Secrecy of the One-Time Pad

Theorem C.1. The one-time pad is perfectly secret.

Proof. To show that the one-time pad is perfectly secret, we need to establish that $\Pr[M = m|C = c] = \Pr[M = m]$ for an $c \in \mathcal{C}$ and $m \in \mathcal{M}$ with $\Pr[C = c] > 0$.

By applying the by Bayes' Theorem, we have:

$$\Pr[M = m|C = c] = \frac{\Pr[C = c|M = m] \cdot \Pr[M = m]}{\Pr[C = c]} \tag{C.1}$$

Next, we simplify the expression for $\Pr[C = c|M = m]$. Note that $C = K \oplus M$. Given the message m, we can write $\Pr[C = c|M = m]$ as

$$\Pr[K \oplus m = c|M = m] = \Pr[K = m \oplus c|M = m]. \tag{C.2}$$

Since the key K is a uniform ℓ-bit string independent of the message M, we have $\Pr[K = m \oplus c|M = m] = \frac{1}{2^\ell}$. Therefore, $\Pr[C = c|M = m] = \frac{1}{2^\ell}$.

Next, using the law of total probability, we show that the probability of choosing a ciphertext C is $\frac{1}{2^\ell}$.

$$\Pr[C = c] = \sum_{m \in \mathcal{M}} \Pr[C = c|M = m] \cdot \Pr[M = m] \tag{C.3}$$

$$= \frac{1}{2^\ell} \sum_{m \in \mathcal{M}} \Pr[M = m] \qquad \left(\text{since } \Pr[C = c|M = m] = \frac{1}{2^\ell}\right) \tag{C.4}$$

$$= \frac{1}{2^\ell}, \qquad \left(\text{since } \sum_{m \in \mathcal{M}} \Pr[M = m] = 1\right) \tag{C.5}$$

Thus, returning back to Equation C.1, we obtain:

$$\Pr[M = m|C = c] = \frac{\Pr[C = c|M = m] \cdot \Pr[M = m]}{\Pr[C = c]} \tag{C.6}$$

$$= \frac{2^{-\ell} \cdot \Pr[M = m]}{2^{-\ell}} \tag{C.7}$$

$$= \Pr[M = m], \tag{C.8}$$

which shows $\Pr[M = m|C = c] = \Pr[M = m]$ as desired. We conclude that the one-time pad is perfectly secret. □

D Correctness of Key Update Rules for Clifford Gates

We provide the proof of correctness of the key update rules as shown in table 6 as well as the SWAP gate discussed in Section 6.5. Proving the correctness of the key update rules for 1-qubit gates is relatively straightforward. However, we use different techniques to establish the correctness for the CNOT and SWAP gates. Consequently, this section is divided into three parts: 1-qubit gates, CNOT gate and the SWAP gate.

D.1 1-qubit Clifford Gates

To prove the correctness of the key update rules for 1-qubit Clifford gates, we first establish a few equalities that will facilitate the proof.

(1)

$$\mathsf{X}\mathsf{Z} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{D.1}$$

$$= -\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = -\mathsf{Z}\mathsf{X}. \tag{D.2}$$

(2)

$$\mathsf{H}\mathsf{X} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \tag{D.3}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \times \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \right) = \mathsf{Z}\mathsf{H}. \tag{D.4}$$

From $\mathsf{H}\mathsf{X} = \mathsf{Z}\mathsf{H}$, we obtain the following relationships:

$$\mathsf{X} = \mathsf{H}\mathsf{Z}\mathsf{H}, \tag{D.5}$$

$$\mathsf{H}\mathsf{X}\mathsf{H} = \mathsf{Z}, \tag{D.6}$$

$$\mathsf{X}\mathsf{H} = \mathsf{H}\mathsf{Z}. \tag{D.7}$$

(3)

$$\mathsf{P}\mathsf{X} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ i & 0 \end{bmatrix} \tag{D.8}$$

$$= -i \times \underbrace{\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}}_{=\mathsf{X}\mathsf{Z} \text{ by Equation D.1}} \times \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} = -i\,\mathsf{X}\mathsf{Z}\mathsf{P}. \tag{D.9}$$

(4)

$$\mathsf{P}\,\mathsf{Z} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} \tag{D.10}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} = \mathsf{Z}\,\mathsf{P}\,. \tag{D.11}$$

We summarize the relevant equivalencies under the global phase:

$$\mathsf{X}\,\mathsf{Z} \equiv \mathsf{Z}\,\mathsf{X} \tag{D.12}$$

$$\mathsf{H}\,\mathsf{X} \equiv \mathsf{Z}\,\mathsf{H} \tag{D.13}$$

$$\mathsf{X}\,\mathsf{H} \equiv \mathsf{H}\,\mathsf{Z} \tag{D.14}$$

$$\mathsf{P}\,\mathsf{X} \equiv \mathsf{X}\,\mathsf{Z}\,\mathsf{P} \tag{D.15}$$

$$\mathsf{P}\,\mathsf{Z} \equiv \mathsf{Z}\,\mathsf{P} \tag{D.16}$$

Now, we have all the tools we need to prove the correctness of the key update rules for Clifford gates. For $\mathsf{I}\,\mathsf{X}^{a_i}\,\mathsf{Z}^{b_i} = \mathsf{X}^{a_i}\,\mathsf{Z}^{b_i}\,\mathsf{I}$, we use the fact that the identity matrix commutes with all the matrices. Therefore, we can move on to the remaining key update rules.

$$\mathsf{X}\,\mathsf{X}^{a_i}\,\mathsf{Z}^{b_i} = \mathsf{X}^{a_i}\,\mathsf{X}\,\mathsf{Z}^{b_i} \tag{D.17}$$

$$= \mathsf{X}^{a_i}\,\mathsf{Z}^{b_i}\,\mathsf{X} \qquad \text{(by Equation D.12)} \tag{D.18}$$

By similar reasoning, $\mathsf{Z}\,\mathsf{X}^{a_i}\,\mathsf{Z}^{b_i} = \mathsf{X}^{a_i}\,\mathsf{Z}^{b_i}\,\mathsf{Z}$.

$$\mathsf{H}\,\mathsf{X}^{a_i}\,\mathsf{Z}^{b_i} = \mathsf{Z}^{a_i}\,\mathsf{H}\,\mathsf{Z}^{b_i} \qquad \text{(by Equation D.13)} \tag{D.19}$$

$$= \mathsf{Z}^{a_i}\,\mathsf{X}^{b_i}\,\mathsf{H} \qquad \text{(by Equation D.14)} \tag{D.20}$$

$$= \mathsf{X}^{b_i}\,\mathsf{Z}^{a_i}\,\mathsf{H}\,. \qquad \text{(by Equation D.12)} \tag{D.21}$$

$$\mathsf{P}\,\mathsf{X}^{a_i}\,\mathsf{Z}^{b_i} = \mathsf{X}^{a_i}\,\mathsf{Z}^{a_i}\,\mathsf{P}\,\mathsf{Z}^{b_i} \qquad \text{(by Equation D.15)} \tag{D.22}$$

$$= \mathsf{X}^{a_i}\,\mathsf{Z}^{a_i}\,\mathsf{Z}^{b_i}\,\mathsf{P} \qquad \text{(by Equation D.16)} \tag{D.23}$$

$$= \mathsf{X}^{a_i}\,\mathsf{Z}^{a_i \oplus b_i}\,\mathsf{P} \tag{D.24}$$

D.2 CNOT Gate

To prove the correctness of the key update for the 2-qubit CNOT gate, we take a slightly different approach. First, we will derive a compact representation of $\mathsf{X}^a\,\mathsf{Z}^b$ by considering the matrices for different values of (a, b). These matrices are recorded in Table (8), and we will make observations based on them.

(a,b)	$\mathsf{X}^a\,\mathsf{Z}^b$
$(0,0)$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
$(1,0)$	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
$(0,1)$	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
$(1,1)$	$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$

Table 8: $\mathsf{X}^a\,\mathsf{Z}^b$ matrices depending on the pair (a,b)

The important observation from Table (8) is that the when $a=0$ the diagonal entries are $|1|$; otherwise, they are 0. More specifically when $(a,b)=(0,0)$, we have $\left(\mathsf{X}^a\,\mathsf{Z}^b\right)_{(1,1)} = \left(\mathsf{X}^a\,\mathsf{Z}^b\right)_{(2,2)} = 1$. When $(a,b)=(0,1)$, $\left(\mathsf{X}^a\,\mathsf{Z}^b\right)_{(1,1)} = 1$ and $\left(\mathsf{X}^a\,\mathsf{Z}^b\right)_{(2,2)} = -1$. These diagonal entries can be compactly described using the Kronecker delta function, defined by

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases} \tag{D.25}$$

By that token $\left(\mathsf{X}^a\,\mathsf{Z}^b\right)_{(1,1)} = \delta_{a0}$ and $\left(\mathsf{X}^a\,\mathsf{Z}^b\right)_{(1,1)} = (-1)^b\delta_{a0}$. Using similar reasoning, we find that for any pair (a,b), the off-diagonal entries can be expressed: $\left(\mathsf{X}^a\,\mathsf{Z}^b\right)_{(1,2)} = (-1)^b\delta_{a1}$ and $\left(\mathsf{X}^a\,\mathsf{Z}^b\right)_{(2,1)} = \delta_{a1}$. Consequently, we can describe the general form of $\mathsf{X}^a\,\mathsf{Z}^b$ for any pair (a,b) as:

$$\mathsf{X}^a\,\mathsf{Z}^b = \begin{bmatrix} \delta_{a0} & (-1)^b\delta_{a1} \\ \delta_{a1} & (-1)^b\delta_{a0} \end{bmatrix} \tag{D.26}$$

Let $a_i = a, b_i = b, a_{i+1} = c, b_{i+1}$. Our objective is to show that

$$\mathsf{CNOT}\left(\mathsf{X}^a\,\mathsf{Z}^b \otimes \mathsf{X}^c\,\mathsf{Z}^d\right) = \left(\mathsf{X}^a\,\mathsf{Z}^{b\oplus d} \otimes \mathsf{X}^{a\oplus c}\,\mathsf{Z}^d\right)\mathsf{CNOT}\,. \tag{D.27}$$

We begin by expanding both sides of the Equation D.27 and establishing the equivalency of the entries, starting with the left-hand side.

$$\mathsf{CNOT}\left(\mathsf{X}^a\,\mathsf{Z}^b \otimes \mathsf{X}^c\,\mathsf{Z}^d\right) \tag{D.28}$$

$$= \mathsf{CNOT}\left(\begin{bmatrix}\delta_{a0} & (-1)^b\delta_{a1}\\ \delta_{a1} & (-1)^b\delta_{a0}\end{bmatrix} \otimes \begin{bmatrix}\delta_{c0} & (-1)^d\delta_{c1}\\ \delta_{c1} & (-1)^d\delta_{c0}\end{bmatrix}\right) \tag{D.29}$$

$$= \mathsf{CNOT}\left(\begin{bmatrix}\delta_{a0}\delta_{c0} & (-1)^d\delta_{a0}\delta_{c1} & (-1)^b\delta_{a1}\delta_{c0} & (-1)^{b\oplus d}\delta_{a1}\delta_{c1}\\ \delta_{a0}\delta_{c1} & (-1)^d\delta_{a0}\delta_{c0} & (-1)^b\delta_{a1}\delta_{c1} & (-1)^{b\oplus d}\delta_{a1}\delta_{c0}\\ \delta_{a1}\delta_{c0} & (-1)^d\delta_{a1}\delta_{c1} & (-1)^b\delta_{a0}\delta_{c0} & (-1)^{b\oplus d}\delta_{a0}\delta_{c1}\\ \delta_{a1}\delta_{c1} & (-1)^d\delta_{a1}\delta_{c0} & (-1)^b\delta_{a0}\delta_{c1} & (-1)^{b\oplus d}\delta_{a0}\delta_{c0}\end{bmatrix}\right) \tag{D.30}$$

$$= \begin{bmatrix}\delta_{a0}\delta_{c0} & (-1)^d\delta_{a0}\delta_{c1} & (-1)^b\delta_{a1}\delta_{c0} & (-1)^{b\oplus d}\delta_{a1}\delta_{c1}\\ \delta_{a0}\delta_{c1} & (-1)^d\delta_{a0}\delta_{c0} & (-1)^b\delta_{a1}\delta_{c1} & (-1)^{b\oplus d}\delta_{a1}\delta_{c0}\\ \delta_{a1}\delta_{c1} & (-1)^d\delta_{a1}\delta_{c0} & (-1)^b\delta_{a0}\delta_{c1} & (-1)^{b\oplus d}\delta_{a0}\delta_{c0}\\ \delta_{a1}\delta_{c0} & (-1)^d\delta_{a1}\delta_{c1} & (-1)^b\delta_{a0}\delta_{c0} & (-1)^{b\oplus d}\delta_{a0}\delta_{c1}\end{bmatrix} \tag{D.31}$$

Note that the Equation D.31 is derived by swapping the third and the fourth rows of Equation D.30.

Now, expanding the right hand side of the Equation D.27 yields:

$$\left(\mathsf{X}^a\,\mathsf{Z}^{b\oplus d} \otimes \mathsf{X}^{a\oplus c}\,\mathsf{Z}^d\right)\mathsf{CNOT} \tag{D.32}$$

$$= \left(\begin{bmatrix}\delta_{a0} & (-1)^{b\oplus d}\delta_{a1}\\ \delta_{a1} & (-1)^{b\oplus d}\delta_{a0}\end{bmatrix} \otimes \begin{bmatrix}\delta_{(a\oplus c)0} & (-1)^d\delta_{(a\oplus c)1}\\ \delta_{(a\oplus c)1} & (-1)^d\delta_{(a\oplus c)0}\end{bmatrix}\right)\mathsf{CNOT} \tag{D.33}$$

$$= \left(\begin{bmatrix}\delta_{a0}\delta_{(a\oplus c)0} & (-1)^d\delta_{a0}\delta_{(a\oplus c)1} & (-1)^{b\oplus d}\delta_{a1}\delta_{(a\oplus c)0} & (-1)^b\delta_{a1}\delta_{(a\oplus c)1}\\ \delta_{a0}\delta_{(a\oplus c)1} & (-1)^d\delta_{a0}\delta_{(a\oplus c)0} & (-1)^{b\oplus d}\delta_{a1}\delta_{(a\oplus c)1} & (-1)^b\delta_{a1}\delta_{(a\oplus c)0}\\ \delta_{a1}\delta_{(a\oplus c)0} & (-1)^d\delta_{a1}\delta_{(a\oplus c)1} & (-1)^{b\oplus d}\delta_{a0}\delta_{(a\oplus c)0} & (-1)^b\delta_{a0}\delta_{(a\oplus c)1}\\ \delta_{a1}\delta_{(a\oplus c)1} & (-1)^d\delta_{a1}\delta_{(a\oplus c)0} & (-1)^{b\oplus d}\delta_{a0}\delta_{(a\oplus c)1} & (-1)^b\delta_{a0}\delta_{(a\oplus c)0}\end{bmatrix}\right)\mathsf{CNOT} \tag{D.34}$$

$$= \begin{bmatrix}\delta_{a0}\delta_{(a\oplus c)0} & (-1)^d\delta_{a0}\delta_{(a\oplus c)1} & (-1)^b\delta_{a1}\delta_{(a\oplus c)1} & (-1)^{b\oplus d}\delta_{a1}\delta_{(a\oplus c)0}\\ \delta_{a0}\delta_{(a\oplus c)1} & (-1)^d\delta_{a0}\delta_{(a\oplus c)0} & (-1)^b\delta_{a1}\delta_{(a\oplus c)0} & (-1)^{b\oplus d}\delta_{a1}\delta_{(a\oplus c)1}\\ \delta_{a1}\delta_{(a\oplus c)0} & (-1)^d\delta_{a1}\delta_{(a\oplus c)1} & (-1)^b\delta_{a0}\delta_{(a\oplus c)1} & (-1)^{b\oplus d}\delta_{a0}\delta_{(a\oplus c)0}\\ \delta_{a1}\delta_{(a\oplus c)1} & (-1)^d\delta_{a1}\delta_{(a\oplus c)0} & (-1)^b\delta_{a0}\delta_{(a\oplus c)0} & (-1)^{b\oplus d}\delta_{a0}\delta_{(a\oplus c)1}\end{bmatrix} \tag{D.35}$$

where Equation D.35 is obtained by swapping the third and fourth columns of Equation D.34.

Our next objective is to show that all 16 entries of matrices in Equations D.30 and D.34 are equal. We first observe that all the coefficients are equal. Specifically, the coefficients of the entries in each row alternate between 1, $(-1)^d$, $(-1)^b$ and $(-1)^{b\oplus d}$, in that order. It remains to show that the variables themselves are equivalent.

Abstractly, all the entries of Equation D.30 take the form $\delta_{ad_1}\delta_{cd_2}$, and all the entries of Equation D.34 take the form $\delta_{ad'_1}\delta_{(a\oplus c)d'_2}$, where $d_1, d_2, d'_1, d'_2 \in \{0, 1\}$. By inspection, we observe that $d_1 = d'_1$ for all the entries. Furthermore, for entries $(1,1), (1,2), (2,1), (2,2), (3,3), (3,4), (4,3)$ and $(4,4)$, we note that $d_1 = d'_1 = 0$ and $d_2 = d'_2$. Thus, for those entries we want to show that $\delta_{a0}\delta_{cd_2} = \delta_{a0}\delta_{(a\oplus c)d_2}$.

The expression $\delta_{a0}\delta_{cd_2}$ equals 1 when $a = 0$ and $c = d_2$. Similarly, the expression $\delta_{a0}\delta_{(a\oplus c)d_2}$ equals 1 when $a = 0$ and $(a \oplus c) = d_2$, and with $a = 0$, this implies that $c = d_2$. Therefore, we have established equivalence for all those entries.

For the remaining entries, $(1,3),(1,4),(2,3),(2,4),(3,1),(3,2),(4,1)$ and $(4,2)$, we observe a slightly different pattern where $d_1 = 1$ and $d_2 \oplus 1 = d_2'$. Therefore, we aim to show that $\delta_{a1}\delta_{cd_2} = \delta_{a1}\delta_{(a\oplus c)d_2'}$. The expression $\delta_{a1}\delta_{cd2}$ equals 1 when $a = 1$ and $c = d_2$. On the other hand, $\delta_{a1}\delta_{(a\oplus c)d_2'}$ equals 1 when $a = 1$ and $a \oplus c = d_2'$. Given that $a = 1$ and $d_2' = d_2 \oplus 1$, we have $1 \oplus c = d_2 \oplus 1$, which implies $c = d_2$. Therefore, the remaining entries are also equivalent. This establishes that the correctness of Equation D.27, as expected.

D.3 SWAP Gate

To prove the correctness of the key updates for the SWAP gate, which swaps the keys (a_1, b_1) and (a_2, b_2) to (a_2, b_2) and (a_1, b_1), respectively, we begin by noting the decomposition of SWAP gate in terms of the set $\{\mathsf{CNOT}_{1\to2}, \mathsf{CNOT}_{2\to1}\}$. In particular,

$$\mathsf{SWAP} = \mathsf{CNOT}_{1\to2} \circ \mathsf{CNOT}_{2\to1} \circ \mathsf{CNOT}_{1\to2} \tag{D.36}$$

To see the correctness of the Equation D.36, we begin with the right-hand side of the equation:

$$\mathsf{CNOT}_{1\to2} \circ \mathsf{CNOT}_{2\to1} \circ \mathsf{CNOT}_{1\to2} \tag{D.37}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{D.38}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{D.39}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \mathsf{SWAP}\,. \tag{D.40}$$

Now that we have established the correctness of Equation D.36, we recall that the key update rule for the $\mathsf{CNOT}_{1\to2}$ gate transforms the key pairs $(a_1, b_1), (a_2, b_2)$ into $(a_1, b_1 \oplus b_2), (a_1 \oplus a_2, b_2)$. Due the symmetry, the key update rule for $\mathsf{CNOT}_{2\to1}$ gate similarly transforms $(a_1, b_1), (a_2, b_2)$ into $(a_1 \oplus a_2, b_1), (a_2, b_1 \oplus b_2)$.

Finally, we compose the key update rules of the decomposition $\mathsf{CNOT}_{1\to2} \circ \mathsf{CNOT}_{2\to1} \circ \mathsf{CNOT}_{1\to2}$ and present the results in Table (9), which establishes the correctness of the key update rule of the SWAP gate.

Gate \ Key	$(a_1, b_1), (a_2, b_2)$
$\mathsf{CNOT}_{1\to2}$	$(a_1, b_1 \oplus b_2), (a_1 \oplus a_2, b_2)$
$\mathsf{CNOT}_{2\to1}$	$(a_1 \oplus a_1 \oplus a_2, b_1 \oplus b_2), (a_1 \oplus a_2, b_1 \oplus b_2 \oplus b_2)$ $= (a_2, b_1 \oplus b_2), (a_1 \oplus a_2, b_1)$
$\mathsf{CNOT}_{1\to2}$	$(a_2, b_1 \oplus b_1 \oplus b_2), (a_1 \oplus a_2 \oplus a_2, b_1)$ $= (a_2, b_2), (a_1, b_1)$

Table 9: Key update rule composition for gates forming the SWAP gate

E Source Code

This section includes the source code for the `py-qhe-epr` with FHE, which depends on the `py-fhe` package. A similar version of `py-qhe-epr` without FHE, as well as a tutorial for setting up the environment, can be found in [Gan24].

As noted in Section 6.3, the `py-qhe-epr` package consists of seven modules. Below, we provide the code for each module.

E.1 che_initialization

```
# import BFV scheme for the classical homomorphic encryption
↪  component of the EPR scheme
# for more information refer to:
↪  https://github.com/sarojaerabelli/py-fhe/blob/master/examples/bfv_mult_example.py
from bfv.batch_encoder import BatchEncoder
from bfv.bfv_decryptor import BFVDecryptor
from bfv.bfv_encryptor import BFVEncryptor
from bfv.bfv_evaluator import BFVEvaluator
from bfv.bfv_key_generator import BFVKeyGenerator
from bfv.bfv_parameters import BFVParameters

# Initialize BFV parameters
degree = 8

plain_modulus = 17 # Ciphertext modulus is a prime congruent to 1
↪  (mod 16).
ciph_modulus = 8000000000000
params = BFVParameters(poly_degree=degree,
                       plain_modulus=plain_modulus,
                       ciph_modulus=ciph_modulus)
key_generator = BFVKeyGenerator(params)
public_key = key_generator.public_key
secret_key = key_generator.secret_key
relin_key = key_generator.relin_key
encoder = BatchEncoder(params)
encryptor = BFVEncryptor(params, public_key)
decryptor = BFVDecryptor(params, secret_key)
evaluator = BFVEvaluator(params)
```

E.2 basic_quantum_operations

```
    # import NumPy for linear algebra computations
    import numpy as np
    from numpy.linalg import matrix_power
```

```
    # import random for generating random numbers
    import random

    # mathematical constant
    SQRT_2 = np.sqrt(2)

    # single qubit gates
    I = np.eye(2)
    H = (1/SQRT_2) * np.array([[1,1],[1,-1]])
    Z = np.array([[1,0],[0,-1]])
    X = np.array([[0,1],[1,0]])
    P = np.array([[1,0],[0,1j]])
    T = np.array([[1,0],[0, np.exp(1j * np.pi/4)]])

    # two-qubit gates
    CNOT = np.array([[1,0,0,0],[0,1,0,0],[0,0,0,1],[0,0,1,0]])
    SWAP = np.array([[1,0,0,0],[0,0,1,0],[0,1,0,0],[0,0,0,1]])

    def gen_qubit(alpha, beta):
    """
    Generate a single qubit according to the probability
↪   amplitudes alpha and beta
    and the constraint alpha^2 + beta^2 = 1.

    Args:
        - alpha: a real number between 0 and 1 inclusive
        - beta: a real number between 0 and 1 inclusive

    Returns:
        - the vector alpha |0> + beta |1>
    """
    return np.array([[alpha],[beta]]) # alpha^2 + beta^2 = 1

def zero_state():
    """
    Generate the zero state |0>.

    Returns:
        - the vector 1|0>
    """
    return gen_qubit(1,0)


def one_state():
    """
    Generate the one state |1>.
```

```
    Returns:
        - the vector 1|1>
    """

    return gen_qubit(0,1)


def superposition():
    """
    Generate the plus state |+> = 1/sqrt(2) * (|0> + |1>).

    Returns:
        - the vector 1/sqrt(2) * (|0> + |1>)
    """
    kat_zero = zero_state()
    return np.matmul(H, kat_zero)


def gen_epr():
    """
    Generate the EPR state 1/sqrt(2) * (|00> + |11>).

    Returns:
        - the vector 1/sqrt(2) * (|00> + |11>)
    """
    # generate two quantum registers and initialize them to the
    ↪  zero stae
    qubit_1, qubit_2 = zero_state(), zero_state()

    # apply the H gate to the first qubit
    qubit_1 = np.matmul(H, qubit_1)

    # apply the CNOT gate and obtain the EPR state
    epr = np.matmul(CNOT, np.kron(qubit_1, qubit_2))

    return epr

def pure_to_density(psi):
    """
    Compute the density matrix of a pure state |\psi>.

    Args:
        - psi: the vector describing a pure state

    Returns:
```

```
        - |\psi> <\psi|
    """
    return np.matmul(psi, psi.conj().T)


def computational_basis(n: int):
    """
    Generate all the basis elements of an n-dimension qubit
↪   system in the computational basis
    For example, if n=1, the function will return the set
↪   {|0>,|1>} as a numpy array.

    Args:
        - n: the dimension of the system

    Returns:
        - arrays: all the basis elements of the n-dimensional
↪   qubit system in the computational basis
    """
    # create 2^n zero arrays
    arrays = [np.zeros((2**n, 1)) for i in range(2**n)]

    # substitute 1 for the i^th element of the i^th array
    for i in range(2**n):
        arrays[i][i] = 1

    return arrays

def measurement(rho): # measurement is performed in the
↪   computational basis
    """
    Return a random string of bits upon the measurement of the
↪   density matrix \rho
    according to projections onto the computational basis.

    Args:
        - rho: density matrix of a quantum state

    Returns:
        - classical_bits: a string of bits with the same length
↪   as the dimension of the quantum state \rho
    """
    # compute the dimension of the system
    dimension = int(np.log2(rho.shape[0]))

    # generate all the basis elements
```

```
    computational_bases = computational_basis(dimension)

    # initialize a list with 2^n elements with zero
    # which stores the probability amplitudes that
    # result from the projections onto the computational basis
    probabilities = [0]*(2**dimension)

    for i in range(2**dimension):
        probabilities[i] = np.abs(np.trace(np.matmul(rho,
        ↪  pure_to_density(computational_bases[i]))))

    # randomly make a choice based on the probability amplitudes
    measurement_res = np.random.choice(len(probabilities), p =
    ↪  probabilities)

    classical_bits = bin(measurement_res)[2:].zfill(dimension)
    return classical_bits


def operator_builder(string, L):
    """
    Concatenate a string of "I"s whose positions are specified by
↪   the list L with the string variable.
    For example, operator_builder("01", [0,2]) returns the string
↪   "I0I1".
    This function will be useful for computing the partial trace
↪   of a quantum state.

    Args:
        - string: a binary string
        - L: a list

    Returns:
        - the string "I" at positions specified by the list L
↪   concatenated with the string
    """
    counter = 0

    # total length of the string
    n = len(string) + len(L)

    # initalize a 0 string as a list
    new_string = list("0"*n)

    # set the L[i] location of the new string to "I"
    for i in range(len(L)):
```

```
            new_string[L[i]] = "I"

        # set the remaining positions according to the string
        ↪ variable that was passed by as an argument
        for j in range(n):
            if new_string[j] == "I":
                pass
            else:
                new_string[j] = string[counter]
                counter = counter + 1

        # turn the list into a string variable
        return "".join(new_string)


def partial_trace(rho, L):
    """
    Compute the partial trace of a density matrix rho, while
↪   keeping the registers specified in the list L
    and tracing out registers that are not in L.

    Args:
        - rho: density matrix of a quantum state
        - L: registers to keep

    Returns:
        - rho_density: partial trace of rho
    """
    system_dimension = int(np.log2(rho.shape[0]))
    trace_out_dimension = system_dimension - len(L)
    rho_density = np.zeros((2**len(L), 2**len(L)))

    for i in range(2**trace_out_dimension):
        operator_matrix = np.array(1)
        operator = bin(i)[2:].zfill(trace_out_dimension)
        operator = operator_builder(operator, L)

        for j in operator:
            if j == "0":
                operator_matrix = np.kron(operator_matrix,
                ↪ zero_state())
            if j == "1":
                operator_matrix = np.kron(operator_matrix,
                ↪ one_state())
            if j == "I":
                operator_matrix = np.kron(operator_matrix, I)
```

```
            rho_density = rho_density +
            ↪  np.matmul(np.matmul(operator_matrix.conj().T, rho),
            ↪  operator_matrix)

    return rho_density

def qgate_on_density(quantum_gate, rho):
    """
    Applies a quantum gate on density matrix.

    Args:
        - quantum_gate: a numpy array as the matrix
↪   representation of a quantum gate
        - rho: density matrix of a quantum state

    Returns:
        - quantum gate * rho * tranpose(conjugate(quantum gate))
    """
    return np.matmul(np.matmul(quantum_gate, rho),
    ↪  quantum_gate.conj().T)

def post_measurement_state(rho, L, bits):
    """
    Given a density matrix rho and the results of a measurement
↪   of a sub-system of rho,
    return the post-measurement state.

    Args:
        - rho: density matrix of a quantum state
        - L: indices of sub-system(s) of rho that are measured as
↪   a list
        bits: a binary string indicating the results of the
↪   measured sub-system(s)

    Returns:
        - rho_pms: the density matrix of the post-measurement
↪   state
        - bits: the same as the input of the matrix
    """
    system_dimension = int(np.log2(rho.shape[0]))
    operator = list("I"*system_dimension)
    for i in range(len(L)):
        operator[L[i]] = bits[i]
    operator = ''.join(i for i in operator)
    projection_operator = np.array(1)
```

```
    zero_projection = np.kron(zero_state(),
    ↪  zero_state().conj().T)
    one_projection = np.kron(one_state(), one_state().conj().T)

    for i in operator:
        if i == "0":
            projection_operator = np.kron(projection_operator,
            ↪  zero_projection)

        if i == "1":
            projection_operator = np.kron(projection_operator,
            ↪  one_projection)

        if i == "I":
            projection_operator = np.kron(projection_operator, I)

    rho_pms = (projection_operator @ rho @
    ↪  projection_operator.conj().T) /
    ↪  (np.trace(projection_operator.conj().T @
    ↪  projection_operator @ rho))

    return (rho_pms, bits)
```

E.3 quantum_one_time_pad

```
    %run basic_quantum_operations.ipynb

    def qotp_enc(psi, a, b):
    """
    Encrypt the pure quantum state |psi> with a list of bits a
↪   and b using quantum one-time pad

    Args:
        psi: a vector representation of the quantum state |psi>
        a: a list of bits needed for the X-Pauli gates of the
↪   quantum one-time pad
        b: a list of bits needed for the Z-Pauli gates of the
↪   quantum one-time pad

    Returns:
        psi_enc: vector representation of the encrypted state
    """
    xz_tensor = np.matmul(matrix_power(X, a[0]),  matrix_power(Z,
    ↪  b[0]))
    n = len(a)
    for i in range(1,n):
```

```
        xz_tensor = np.kron(xz_tensor,
        ↪  np.matmul(np.linalg.matrix_power(X, a[i]),
                                                ↪  np.linalg.matrix_power(Z,
                                                ↪  b[i])))

    psi_enc = np.matmul(xz_tensor, psi)
    return psi_enc


def qotp_dec_density(enc_rho, a,b):
    """
    Decrypt the density matrix rho with a list of bits a and b
↪   using quantum one-time pad

    Args:
        enc_rho: a density matrix representation of the encrypted
↪   quantum state rho
        a: a list of bits needed for the X-Pauli gates of the
↪   quantum one-time pad
        b: a list of bits needed for the Z-Pauli gates of the
↪   quantum one-time pad

    Returns:
        dec_rho: density matrix of the decrypted state
    """
    xz_tensor = np.matmul(np.linalg.matrix_power(Z, b[0]),
    ↪  np.linalg.matrix_power(X, a[0]))
    n = len(a)
    for i in range(1,n):
        xz_tensor = np.kron(xz_tensor,
        ↪  np.matmul(np.linalg.matrix_power(Z, b[i]),
                                                ↪  np.linalg.matrix_power(X,
                                                ↪  a[i])))

    dec_rho = qgate_on_density(xz_tensor, enc_rho)
    return dec_rho
```

E.4 epr_encryption

```
    %run quantum_one_time_pad.ipynb
    %run che_initialization.ipynb

    def pad_zeros(L, n):
    """
```

```
    Pad the remaining elements of each member of a list with 0
↪   until the length of each list becomes n.
    This function is used for encrypting a single bit using BFV
↪   scheme where
    the degeree of a polynomial must be a power of two.

    For example, if L = [0,1] and n = 4, then
    pad_zeros(L, n) returns [[0,0,0,0],[1,0,0,0]]

    Args:
        - L: list of bits
        - n: the length of list after padding 0 elements
    Returns:
        - a concatenation of lst with a list of zeros so that the
↪   output is a list of n elements
    """
    return [[i] + [0] * (n - 1) for i in L]

    def epr_enc(psi,a,b):
    """
    Encrypt a pure quantum state |psi> using a list of classical
↪   bits a and b according to the epr scheme.
    by Broadbent and Jeffery https://arxiv.org/abs/1412.8766

    Args:
        - psi: a vector representation of the pure state |psi>
        - a: list of (randomly generated) classical bits
        - b: list of (randomly generated) classical bits

    Returns:
        - psi_enc: encrypted pure state by the quantum one-time
↪   pad
        - a_enc: a list of ciphertexts resulted from the BFV
↪   classical homomorphic encryption scheme of list `a`
        - b_enc: a list of ciphertexts resulted from the BFV
↪   classical homomorphic encryption scheme of list `b`
    """
    padded_a = pad_zeros(a, degree)
    padded_b = pad_zeros(b, degree)

    encoded_a = [encoder.encode(i) for i in padded_a]
    encoded_b = [encoder.encode(i) for i in padded_b]

    a_enc = [encryptor.encrypt(i) for i in encoded_a]
    b_enc = [encryptor.encrypt(i) for i in encoded_b]
```

```
    psi_enc = qotp_enc(psi, a, b)
    return psi_enc, a_enc, b_enc
```

E.5 epr_evaluation

```
    # import copy for copying lists
    import copy

    # import regular expression package for string manipulation
    import re

    # import SymPy package for symoblic computation
    from sympy import symbols, Add, Mul

    %run quantum_one_time_pad.ipynb
    %run che_initialization.ipynb
    %run epr_encryption.ipynb

    def tcl_layers(circuit):
    """
    Seperate T + Clifford gates of each layer of a quantum
↪   circuit.

    Args:
        - circuit: a quantum circuit as a list of strings. each
↪   member of the list
        represents a layer in the quantum circuit.

    Returns:
        - layered_circuit: a nested list where each nested list
↪   seperates individual gates.
    """
    qgates = ["CNOT", "Z", "I", "X", "P", "H", "T"]
    layered_circuit = []

    length_current_layer = 0
    for layer in range(len(circuit)):
        length_current_layer = len(circuit[layer])
        counter = 0
        current_gates = []
        for gates in range(length_current_layer):
            if counter >= length_current_layer:
                break
            if circuit[layer][counter] != "C":
                current_gates.append(circuit[layer][counter])
```

```
                counter += 1
            else:
                current_gates.append("CNOT")
                counter += 4
        layered_circuit.append(current_gates)

    return layered_circuit

    def clifford_key_updates(cg, a, b):
    """
    Perform key updates homomorphically on ciphertexts depending
↪   on based on a Clifford gate.

    Args:
        - cg: Clifford gate as a string
↪   ("I","X","Z","H","P","CNOT")
        - a: ciphertext
        - b: ciphertext

    Returns:
        tuple
            If the Clifford gate is a single qubit gate:
                - a tuple of updated ciphertexts
            If the Clifford gate is the CNOT gate:
                - a tuple of lists, each containing the updated
↪   ciphertexts for the current wire and
                its neighboring wire for a and b, respectively.
    """
    if cg == 'X' or cg == 'Z' or cg == 'I':
        return (a,b)

    if cg == 'H':
        return(b,a)

    if cg == 'P':
        return(a, evaluator.add(a, b))


    if cg == 'CNOT':
        return ([a[0], evaluator.add(a[0],a[1])]
        ↪  ,[evaluator.add(b[0],b[1]), b[1]])


    def cl_eval(circuits, enc_psi, a_tilde, b_tilde):
    """
```

```
    Perform clifford circuit on the encrypted pure state
↪   homomorphically
    and update the classical ciphertexts using classical
↪   homomorphic encryption scheme.
    For more information refer to the Clifford scheme by
↪   Broadbent and Jeffery: https://arxiv.org/abs/1412.8766

    Args:
        - circuits: description of the Clifford circuit as a list
↪   of strings
        - enc_psi: a vector representation of the encrypted pure
↪   state |psi>_{enc}
        - a_tilde: a list of classical ciphertexts which contain
↪   the information about the X-Pauli gate exponents
        - b_tilde: a list of classical ciphertexts which contain
↪   the information about the Z-Pauli gate exponents

    Returns:
        - rho_hom_eval: density matrix of the encrypted evaluated
↪   quantum state
        - a_tilde_update: a list of updated classical ciphertexts
        which contain the information about the X-Pauli gate
↪   exponents
        - b_tilde_update: a list of updated classical ciphertexts
        which contain the information about the Z-Pauli gate
↪   exponents
        - circuit_dictionary: a dictonary object where keys
↪   indicate the ordering of qubits, starting from 1 as a string.
        The values associated with each key is a list which
↪   contains the following information, in the same order:
            1) acting wire
            2) a Boolean True/False value indicating whether a T
↪   gate has been applied during evaluation or not.
            This value gets updated during the evaluation
↪   process.
            3) the total number of T gates that are going to be
↪   applied on the qubit as an integer.
            4) the number of T gates applied at the end of the
↪   execution of each layer. This value gets updated
            during the evaluation process.
    """
    # making copies of ciphertexts
    a_tilde_copy, b_tilde_copy = a_tilde, b_tilde
    a_tilde_update, b_tilde_update = a_tilde, b_tilde

    # density matrix representation of the encrypted state
```

```
rho_hom_eval = pure_to_density(enc_psi)

# dimension of the quantum state
n = len(a_tilde)

for i in tcl_layers(circuits):
    counter = 0

    # current circuit iteratively build up the quantum gate
    ↪  in each layer of the ciruit
    current_circuit = np.array([1])
    for j in range(len(i)):
        if i[j] == 'I':
            current_circuit = np.kron(current_circuit, I)
            a_tilde_update[counter], b_tilde_update[counter]
            ↪  = clifford_key_updates(
                i[j], a_tilde_copy[counter],
                ↪  b_tilde_copy[counter]
            )
            counter = counter + 1

        if i[j] == 'X':
            current_circuit = np.kron(current_circuit, X)
            a_tilde_update[counter], b_tilde_update[counter]
            ↪  = clifford_key_updates(
                i[j], a_tilde_copy[counter],
                ↪  b_tilde_copy[counter]
            )
            counter = counter + 1

        if i[j] == 'Z':
            current_circuit = np.kron(current_circuit, Z)
            a_tilde_update[counter], b_tilde_update[counter]
            ↪  = clifford_key_updates(
                i[j], a_tilde_copy[counter],
                ↪  b_tilde_copy[counter]
            )
            counter = counter + 1

        if i[j] == 'H':
            current_circuit = np.kron(current_circuit, H)
            a_tilde_update[counter], b_tilde_update[counter]=
            ↪  clifford_key_updates(
                i[j], a_tilde_copy[counter],
                ↪  b_tilde_copy[counter]
            )
```

```
                counter = counter + 1

            if i[j] == 'P':
                current_circuit = np.kron(current_circuit, P)
                a_tilde_update[counter], b_tilde_update[counter]
                ↪  = clifford_key_updates(
                    i[j], a_tilde_copy[counter],
                    ↪  b_tilde_copy[counter]
                )
                counter = counter + 1

            if i[j] == 'CNOT':
                current_circuit = np.kron(current_circuit, CNOT)
                a_tilde_update[counter:counter+2],
                ↪  b_tilde_update[counter:counter+2] =
                ↪  clifford_key_updates(
                    i[j], a_tilde_copy[counter:counter+2],
                    ↪  b_tilde_copy[counter:counter+2]
                )
                counter = counter + 2
        rho_hom_eval =
        ↪  qgate_on_density(current_circuit,rho_hom_eval)

    circuit_dictionary = wire_dictionary(circuits)
    return (rho_hom_eval, a_tilde_update, b_tilde_update,
    ↪  circuit_dictionary, None, None, None, None)


    def no_qubits(circuit):
    """
    Compute the number of qubits based on the circuit.

    Args:
        - circuit: description of the quantum circuit as a list
↪   of strigs. Each element of the list represents one layer
        of the circuit.

    Returns:
        - number_of_qubits: an integer corresponding to the
↪   number of qubits on which the circuit is acting upon.
    """
    number_of_qubits = 0
    first_layer = tcl_layers([circuit[0]])[0]
    for i in first_layer:
        if i == "CNOT":
            number_of_qubits = number_of_qubits + 2
```

```
        else:
            number_of_qubits = number_of_qubits + 1
    return(number_of_qubits)

    def flatten_circuit(circuits):
    """
    Extract all the gates applied to each qubit in the same
↪   ordering of the qubits.

    Args:
        - circuits: description of the quantum circuit as a list
↪   of strigs. Each element of the list represents one layer
        of the circuit.

    Returns:
        - flattened_circuit: a nested list, where each sub-list
↪   contains the gates applied to each qubit
        and each layer. If the CNOT gate is present, then CNOT(1)
↪   indicates the control qubit
        and CNOT(2) indicates the target qubit.
    """
    number_of_qubits = no_qubits(circuits)
    individual_gates = tcl_layers(circuits)
    number_of_layers = len(individual_gates)
    flattened_circuit = [[] for _ in range(number_of_qubits)]
    for i in range(number_of_layers):
        counter = 0
        for j in range(number_of_qubits):
            if counter + 1 > number_of_qubits:
                break

            if individual_gates[i][j] != "CNOT":

                ↪  flattened_circuit[counter].append(individual_gates[i][j])
                counter = counter + 1
            else:
                flattened_circuit[counter].append("CNOT(1)")
                flattened_circuit[counter+1].append("CNOT(2)")
                counter = counter + 2

    return flattened_circuit

    def number_of_tgates(circuit):
    """
    Count the number of T gates in a circuit.
```

```
    Args:
        - circuits: description of the quantum circuit as a list
↪   of strigs. Each element of the
        list represents one layer of the circuit.

    Returns:
        - t_gate_count: total number of T gates in a circuit as
↪   an integer
    """
    L = tcl_layers(circuit)
    t_gate_count = 0
    for i in L:
        for j in i:
            if j == "T":
                t_gate_count = t_gate_count + 1
    return t_gate_count

    def wire_dictionary(circuits):
    """
    Construct a dictionary which contains information about the
↪   order of wires that act upon each qubit,
    whether a T gate has been applied to a qubit during the
↪   evaluation of the EPR scheme or not, how many T gates
    will be acted upon each qubit and how many T gates have been
↪   applied during each iteration of the EPR evaluation
    process.
    This dictionary will provide a systematic way of keeping
↪   track of wires acting upon each qubit.

    Args:
        - circuits: description of the quantum circuit as a list
↪   of strigs. Each element of the list represents one layer
        of the circuit.

    Returns:
        - circuit_dictionary: a dictonary object where keys
↪   indicate the ordering of qubits, starting from 1 as a string.
        The values associated with each key is a list which
↪   contains the following information, in the same order:
            1) acting wire
            2) a Boolean True/False value indicating whether a T
↪   gate has been applied during evaluation or not.
            This value gets updated during the evaluation
↪   process.
            3) the total number of T gates that are going to be
↪   applied on the qubit as an integer.
```

```
        4) the number of T gates applied at the end of the
↪ execution of each layer. This value gets updated
        during the evaluation process.
    """
    flat_circuit = flatten_circuit(circuits)
    number_of_qubits = len(flat_circuit)
    circuit_dictionary = {}
    for i in range(1, number_of_qubits+1):
        key = str(i)
        value = [i, False, number_of_tgates(flat_circuit[i-1]),
        ↪  0]
        circuit_dictionary[key] = value
    return circuit_dictionary

    def no_wires(circuits):
    """
    Count the total of number of wires required to run a circuit
↪ using the EPR evaluation scheme.

    Args:
        - circuits: description of the quantum circuit as a list
↪ of strigs. Each element of the list represents one layer
        of the circuit.

    Returns:
        - number_of_wires: total number of wires, which is
↪ calculated by
        number of qubits in the input message + 2 * number of
↪ T gates
    """
    number_of_qubits = no_qubits(circuits)
    t_gate_count = number_of_tgates(circuits)
    number_of_wires = number_of_qubits + 2 * t_gate_count
    return number_of_wires

    def tgate_index(dictionary):
    """
    Construct a list consisting of the orderings of qubits on
↪ which a T gate is being applied.

    Args:
        - dictionary: a dictonary object where keys indicate the
↪ ordering of qubits, starting from 1 as a string.
        The values associated with each key is a list which
↪ contains the following information, in the same order:
        1) acting wire
```

```
            2) a Boolean True/False value indicating whether a T
↪   gate has been applied during evaluation or not.
            This value gets updated during the evaluation
↪   process.
            3) the total number of T gates that are going to be
↪   applied on the qubit as an integer.
            4) the number of T gates applied at the end of the
↪   execution of each layer. This value gets updated
            during the evaluation process.

    Returns:
        - t_index_list: a list which records the orderings of
↪   qubits on which a T gate is being applied.
    """
    t_index_list = []
    for key, value in dictionary.items():
        if dictionary[key][2] > 0:
            t_index_list.append(dictionary[key][0]-1)
    return t_index_list

    def iterated_tensor_prod(M, n): # Compute M^{\otimes n}
    """
    Compute the matrix M^{\otimes n} with the convention that if
↪   n = 0, then the output is the integer 1.

    Args:
        - M: A matrix
        - n: a non-negative integer

    Returns:
        - iterated_tensored_m: M^{\otimes n}
    """
    if n == 0:
        return 1
    iterated_tensored_m = M
    for i in range(n-1):
        iterated_tensored_m = np.kron(iterated_tensored_m, M)
    return iterated_tensored_m

    def number_of_t_gates_applied(dictionary, qubit_order):
    """
    Count the total number of T gates applied to a given qubit
↪   during the evaluation procedure of the EPR scheme.

    Args:
```

```
        - dictionary: a dictonary object where keys indicate the
↪   ordering of qubits, starting from 1 as a string.
        The values associated with each key is a list which
↪   contains the following information, in the same order:
            1) acting wire
            2) a Boolean True/False value indicating whether a T
↪   gate has been applied during evaluation or not.
            This value gets updated during the evaluation
↪   process.
            3) the total number of T gates that are going to be
↪   applied on the qubit as an integer.
            4) the number of T gates applied at the end of the
↪   execution of each layer. This value gets updated
            during the evaluation process.
        qubit_order: the order of a qubit in a n-qubit system, an
↪   integer between 1 to n.

    Returns:
        - number_of_t_gates: an integer counting the number of T
↪   gates applied during
    """
    number_of_t_gates = 0
    for key in dictionary:
        if int(key)-1 < qubit_order:
            number_of_t_gates += dictionary[key][2]  # Access the
            ↪  third element of the list
    return number_of_t_gates

    def controlled_not_constructor(control, target,
    ↪  number_of_qubits):
    """
    This function applies the controlled-not operation on the
↪   target £t£ which is being controlled by £c£,
    where £t < c£ or £c > t£ and the operation is being applied
↪   on a £n£-qubit system.
    The operator matrix is defined as follows:

    if £t < c£:

    ££CX_{c, t, n} = \left[ I^{\otimes(c-1)} \otimes | 0 \rangle
↪   \langle 0 | \otimes I^{\otimes(n-c)} \right] +
    \left[ I^{\otimes(t-1)} \otimes X \otimes I^{\otimes(c-t-1)}
↪   \otimes | 1 \rangle \langle 1 | \otimes I^{\otimes (n-c)}
↪   \right]
    ££
```

```
    if £t > c£:

    ££
    CX_{c, t, n} = \left[ I^{\otimes(c-1)} \otimes | 0 \rangle
↪   \langle 0 | \otimes I^{\otimes(n-c)} \right]
    + \left[ I^{\otimes(c-1)} \otimes | 1 \rangle \langle 1 |
↪   \otimes I^{\otimes(t-c-1)} \otimes X \otimes I^{\otimes (n-t)}
↪   \right]
    ££

    £I^{\otimes 0}£ is defined to be the real number £1£.

    This function is used when applying the `T` gate during the
↪   EPR scheme.

    Args:
        - control: the order of the control wire
        - target: the order of the target wire
        - number_of_qubits: number of qubits in the quantum
↪   system

    Returns:
        - controlled_not: the unitary operator which applies the
↪   X gate on
        wire `target` if the `control` wire is 1.
    """
    do_nothing_operator = np.kron(np.eye(2 ** (control - 1)),
    ↪  pure_to_density(zero_state()))
    do_nothing_operator = np.kron(do_nothing_operator, np.eye(2
    ↪  ** (number_of_qubits - control)))
    if target < control:
        flip_target_operator = np.eye(2 ** (target - 1))
        flip_target_operator = np.kron(flip_target_operator, X)
        flip_target_operator = np.kron(flip_target_operator,
        ↪  np.eye(2 ** (control - target - 1)))
        flip_target_operator = np.kron(flip_target_operator,
        ↪  pure_to_density(one_state()))
        flip_target_operator = np.kron(flip_target_operator,
        ↪  np.eye(2 ** (number_of_qubits - control)))

    else:
        flip_target_operator = np.eye(2 ** (control - 1))
        flip_target_operator = np.kron(flip_target_operator,
        ↪  pure_to_density(one_state()))
        flip_target_operator = np.kron(flip_target_operator,
        ↪  np.eye(2 ** (target - control - 1)))
```

```
            flip_target_operator = np.kron(flip_target_operator, X)
            flip_target_operator = np.kron(flip_target_operator,
            ↪  np.eye(2 ** (number_of_qubits - target)))

        controlled_not = do_nothing_operator + flip_target_operator
        return controlled_not

        def encrypt_one_bit(a):
        """
        Encrypt a bit `a` using the BFV classical fully homomorphic
    ↪   encryption scheme.

        Args:
            - a: a list containing a bit to be encrypted

        Returns:
            - a_enc: a list containing the ciphertext resulted from
    ↪   the encryption of the bit
        """
        padded_a = pad_zeros(a, degree)
        encoded_a = [encoder.encode(i) for i in padded_a]
        a_enc = [encryptor.encrypt(i) for i in encoded_a]

        return a_enc

        def t_gate_sequence(circuit):
        """
        Construct a nested list where each nested list contains the
    ↪   information about the
        order of the qubit on which a T gate is applied and the
    ↪   corresponding order of the layer
        as it appears in the circuit sequentially.

        Args:
            - circuit: description of the quantum circuit as a list
    ↪   of strigs. Each element of the list represents one layer
            of the circuit.

        Returns:
            - t_gate_sequence_over_time: a nested list where each
    ↪   element is a list with the order of the qubit and the
            order of the layer in which it appears in the circuit.
        """
        individual_gates = tcl_layers(circuit)
        t_gate_count = number_of_tgates(circuit)
        t_gate_sequence_over_time = [[]]*t_gate_count
```

```
    t_gate_count_over_time = 0
    for gate_order in range(len(individual_gates)):
        qubit_order = 0
        for quasi_qubit_order in
        ↪  range(len(individual_gates[gate_order])):
            if individual_gates[gate_order][quasi_qubit_order] ==
            ↪  "T":
                t_gate_sequence_over_time[t_gate_count_over_time]
                ↪  = [qubit_order, gate_order]
                t_gate_count_over_time = t_gate_count_over_time +
                ↪  1

            if individual_gates[gate_order][quasi_qubit_order] in
            ↪  ("T","H","P","X","Z","I"):
                qubit_order = qubit_order + 1

            if individual_gates[gate_order][quasi_qubit_order] ==
            ↪  "CNOT":
                qubit_order = qubit_order + 2

    return t_gate_sequence_over_time

    def sub_zero_ab(polynomial):
    """
    Substitute 0 for variables `a` and `b` in a symbolic
↪   polynomial which may consists `a` or `b`.

    Args:
        - polynomial: a symbolic polynomial which may consist `a`
↪   or `b`

    Returns:
        - eval_polynomial: the evaluated `polynomial` where 0 has
↪   been substituted for variables `a` and `b`
    """
    string_polynomial = str(polynomial)
    eval_polynomial = polynomial
    if "a" in string_polynomial:
        for i in list(polynomial.free_symbols):
            if "a" in str(i):
                eval_polynomial = eval_polynomial.subs(i,0)

    if "b" in string_polynomial:
        for i in list(polynomial.free_symbols):
            if "b" in str(i):
                eval_polynomial = eval_polynomial.subs(i,0)
```

```
    return eval_polynomial

    def get_subscript_superscript(monomial):
    """
    Extract d1 and d2 from a monomial of the form a_{d1}^{d2}.
↪   This information will be used
    to determine the gate order and the qubit order in the
↪   evaluation of a polynomial.

    Args:
        - monomial: a symbolic expression of the form
↪   "c_{d1}^{d2}" or "k_{d1}^{d2}".

    Returns:
        - subscript: the integer that is the subscript of the
↪   monomial.
        - superscript: the integer that is the superscript of the
↪   monomial.
    """
    # Define a regular expression pattern to match integers
    ↪  preceded by "_" and followed by "^"
    pattern = re.compile(r'_(\d+)\^(\d+)')
    matches = pattern.findall(monomial)

    # Extract subscript and superscript
    subscript = int(matches[0][0])
    superscript = int(matches[0][1])

    return (subscript, superscript)


    def evaluate_expression(polynomial, k_measurements,
    ↪  c_measurements):
    """
    Evaluate a polynomial with variables in terms of `k` and `c`
↪   whose values are given
    in the lists `k_measurements` and `c_measurements`.

    Args:
        - polynomial: a symbolic polynomial consisting of
↪   variables `a`, `b`, `c` and `k`
        - k_measurements: a list of measured `k` values
        - c_measurements: a list of measured `c` values

    Returns:
```

```
        - evaluated_polynomial: the evaluated value of the
↪   polynomial after substituting the values for `k` and `c`
        modulo 2
    """
    polynomial_no_ab = sub_zero_ab(polynomial)
    evaluated_polynomial = polynomial_no_ab
    for i in list(polynomial_no_ab.free_symbols):
        if "c" in str(i):
            c_gate_order, c_qubit_order =
            ↪  get_subscript_superscript(str(i))
            evaluated_polynomial = evaluated_polynomial.subs(i,
            ↪  c_measurements[c_qubit_order][c_gate_order])

        if "k" in str(i):
            k_gate_order, k_qubit_order =
            ↪  get_subscript_superscript(str(i))
            evaluated_polynomial = evaluated_polynomial.subs(i,
            ↪  k_measurements[k_qubit_order][k_gate_order])

    evaluated_polynomial = int(evaluated_polynomial) % 2
    return evaluated_polynomial

    def phase_gate_polynomial(circuits):
    """
    Construct the symbolic expression for evaluating the phase
↪   gate exponent before applying
    a T gate, as well as the the symbolic expression of the key
↪   updates at the end of the
    circuit evaluation.

    The symbolic construction is defined in the following way
↪   depending on the input gate:
    (Table generated using the `tabulate` package)


↪   ---------------------------------------------------------------------------
    | Gate    | key                       | updated key
↪   |

↪   ---------------------------------------------------------------------------
    | I, X, Z | (a,b)                     | (a,b)
↪   |

↪   ---------------------------------------------------------------------------
    | H       | (a,b)                     | (b,a)
↪   |
```

```
    ----------------------------------------------------------------------------
    | P       | (a,b)                    | (a, a + b)
    |

    ----------------------------------------------------------------------------
    | CNOT    | ((a_1, b_1),(a_2, b_2)) | ((a_1, b_1 + b_2),(a_1
    + a_2), b_2) |

    ----------------------------------------------------------------------------
    | T       | (a,b)                    | (a+c, a+(a*c)+b+k)
    |

    ----------------------------------------------------------------------------

    Args:
        - circuits: description of the quantum circuit as a list
    of strigs. Each element of the list represents one layer
        of the circuit.

    Returns:
        - phase_gate_polynomials: a list containing the symbolic
    polynomial before the applicaton
        of a T gate and all the key updates at the end of the
    circuit evaluation.
    """
    individual_gates = tcl_layers(circuits)
    number_of_qubits = no_qubits(circuits)
    key_update_list =
    ↪  [[symbols(f'a_{1}^{i}'),symbols(f'b_{1}^{i}')] for i in
    ↪  range(number_of_qubits)]
    circuit_dictionary = wire_dictionary(circuits)
    t_index_list = tgate_index(circuit_dictionary)
    flat_circuit = flatten_circuit(circuits)
    phase_gate_polynomials = [[] for i in
    ↪  range(number_of_qubits)]

    for i in t_index_list:
        phase_gate_polynomials[i] = [key_update_list[i][0]]


    for gate_order in range(len(individual_gates)):

        # we create a seperate variable to keep track of the
        ↪  order of the qubit
        # to handle the case when a CNOT is applied after which
```

```
# the order of qubit will increase by 2.
qubit_order = 0
for quasi_qubit_order in range(number_of_qubits):
    if qubit_order < number_of_qubits:
        current_gate =
        ↪  individual_gates[gate_order][quasi_qubit_order]

        if current_gate in ("I","X","Z"):
            key_update_list[qubit_order] =
            ↪  [key_update_list[qubit_order][0],
            ↪  key_update_list[qubit_order][1]]
            if gate_order+1 < len(individual_gates) and
            ↪  flat_circuit[qubit_order][gate_order+1]
            ↪  == "T" and
            ↪  circuit_dictionary[str(qubit_order+1)][3]
            ↪  > 0:

                ↪  phase_gate_polynomials[qubit_order].append(
                key_update_list[qubit_order][0])

            elif gate_order+1 < len(individual_gates) and
            ↪  flat_circuit[qubit_order][gate_order+1]
            ↪  == "T" and
            ↪  circuit_dictionary[str(qubit_order+1)][3]
            ↪  == 0:
                phase_gate_polynomials[qubit_order] =
                ↪  [key_update_list[qubit_order][0]]


            qubit_order += 1

        if current_gate == "H":
            key_update_list[qubit_order] =
            ↪  [key_update_list[qubit_order][1],
            ↪  key_update_list[qubit_order][0]]

            if gate_order+1 < len(individual_gates) and
            ↪  flat_circuit[qubit_order][gate_order+1]
            ↪  == "T" and
            ↪  circuit_dictionary[str(qubit_order+1)][3]
            ↪  > 0:

                ↪  phase_gate_polynomials[qubit_order].append(
                key_update_list[qubit_order][0])
```

```
            elif gate_order+1 < len(individual_gates) and
            ↪  flat_circuit[qubit_order][gate_order+1]
            ↪  == "T" and
            ↪  circuit_dictionary[str(qubit_order+1)][3]
            ↪  == 0:
                phase_gate_polynomials[qubit_order] =
                ↪  [key_update_list[qubit_order][0]]

            qubit_order += 1


        if current_gate == "P":
            key_update_list[qubit_order] =
            ↪  [key_update_list[qubit_order][0],
                                                Add(

                                                ↪  key_update_list[qubit_order][0],

                                                  ↪  key_update_list

                                                  ↪  [qubit_order][1])]

            if gate_order+1 < len(individual_gates) and
            ↪  flat_circuit[qubit_order][gate_order+1]
            ↪  == "T" and
            ↪  circuit_dictionary[str(qubit_order+1)][3]
            ↪  > 0:

                ↪  phase_gate_polynomials[qubit_order].append(
                key_update_list[qubit_order][0])

            elif gate_order+1 < len(individual_gates) and
            ↪  flat_circuit[qubit_order][gate_order+1]
            ↪  == "T" and
            ↪  circuit_dictionary[str(qubit_order+1)][3]
            ↪  == 0:
                phase_gate_polynomials[qubit_order] =
                ↪  [key_update_list[qubit_order][0]]


            qubit_order += 1


        if current_gate == "CNOT":
            key_update_list[qubit_order] =
            ↪  [key_update_list[qubit_order][0],
```

```
                                        Add(
                                        ↪  key_update_list[qubit_order][1],
                                            ↪  key_update_list
                                            ↪  [qubit_order+1][1]
                                           )
                                        ]
key_update_list[qubit_order+1] =
↪  [Add(key_update_list[qubit_order+1][0],
                                                ↪  key_update_list[qubit_order]
                                              ↪  key_update_list
                                              ↪  [qubit_order+1][1]]


if gate_order + 1 < len(individual_gates) and
↪  flat_circuit[qubit_order][gate_order+1]
↪  == "T" and
↪  circuit_dictionary[str(qubit_order+1)][3]
↪  > 0:

    ↪  phase_gate_polynomials[qubit_order].append(key_update_list
    [qubit_order][0])

if gate_order+1 < len(individual_gates) and
↪  flat_circuit[qubit_order][gate_order+1]
↪  == "T" and
↪  circuit_dictionary[str(qubit_order+1)][3]
↪  == 0:
    phase_gate_polynomials[qubit_order] =
    ↪  [key_update_list[qubit_order][0]]

if gate_order + 1 < len(individual_gates) and
↪  flat_circuit[qubit_order+1][gate_order+1]
↪  == "T" and
↪  circuit_dictionary[str(qubit_order+2)][3]
↪  > 0:

    ↪  phase_gate_polynomials[qubit_order+1].append(
    key_update_list[qubit_order+1][0])
```

```
                if gate_order+1 < len(individual_gates) and
                ↪  flat_circuit[qubit_order+1][gate_order+1]
                ↪  == "T" and
                ↪  circuit_dictionary[str(qubit_order+2)][3]
                ↪  == 0:
                    phase_gate_polynomials[qubit_order+1] =
                    ↪  [key_update_list[qubit_order+1][0]]

                if gate_order + 1 == len(individual_gates):

                    ↪  phase_gate_polynomials[qubit_order].append(
                        [key_update_list[qubit_order][0],
                        ↪  key_update_list[qubit_order][1]]
                    )

                    ↪  phase_gate_polynomials[qubit_order+1].append(
                        [key_update_list[qubit_order+1][0],
                        ↪  key_update_list[qubit_order+1][1]]
                    )


                qubit_order += 2

            if current_gate == "T":
                no_t_gates_ith_wire =
                ↪  circuit_dictionary[str(qubit_order+1)][3]
                c_symbol =
                ↪  symbols(f'c_{no_t_gates_ith_wire}^{qubit_order}')
                k_symbol =
                ↪  symbols(f'k_{no_t_gates_ith_wire}^{qubit_order}')

                key_update_list[qubit_order] =
                ↪  [key_update_list[qubit_order][0],
                                                Add(Add(Add(

                                                ↪  key_update_list[qubit_order][0],
                                                    Mul(

                                                    ↪  key_update_list

                                                    ↪  [qubit_order][0],c_symbol)),

                                                    ↪  k_symbol),
                                                    ↪  key_update_list

                                                    ↪  [qubit_order][1])]
```

```
                    if gate_order == 0:
                        phase_gate_polynomials[qubit_order] =
                        ↪  [key_update_list[qubit_order][0]]
                        circuit_dictionary[str(qubit_order+1)][3]
                        ↪  = no_t_gates_ith_wire+1

                    if gate_order > 0 and gate_order+1 <
                    ↪  len(individual_gates) and
                    ↪  flat_circuit[qubit_order][gate_order+1]
                    ↪  == "T" and
                    ↪  circuit_dictionary[str(qubit_order+1)][3]
                    ↪  == 0:
                        phase_gate_polynomials[qubit_order] =
                        ↪  [key_update_list[qubit_order][0]]

                        circuit_dictionary[str(qubit_order+1)][3]
                        ↪  = no_t_gates_ith_wire+1

                    if gate_order+1 < len(individual_gates) and
                    ↪  flat_circuit[qubit_order][gate_order+1]
                    ↪  == "T" and
                    ↪  circuit_dictionary[str(qubit_order+1)][3]
                    ↪  > 0:

                        ↪  phase_gate_polynomials[qubit_order].append(
                        key_update_list[qubit_order][0])
                        circuit_dictionary[str(qubit_order+1)][3]
                        ↪  = no_t_gates_ith_wire+1

                    if gate_order+1 < len(individual_gates) and
                    ↪  flat_circuit[qubit_order][gate_order+1]
                    ↪  != "T":
                        circuit_dictionary[str(qubit_order+1)][3]
                        ↪  = no_t_gates_ith_wire+1
                    qubit_order += 1

                if gate_order+1 == len(individual_gates) and
                ↪  current_gate != "CNOT":
                    phase_gate_polynomials[qubit_order-1].append(
                        [key_update_list[qubit_order-1][0],
                        ↪  key_update_list[qubit_order-1][1]]
                    )

    return phase_gate_polynomials
```

```
    def epr_eval(circuits, enc_psi, a_tilde, b_tilde):
    """
    Perform a circuit decomposed into Clifford + T gates on the
↪   encrypted quantum state
    homomorphically and update the keys. For every T gate, store
↪   the updated value of "a_tilde" before the application
    of the T gate. Upon applying the T gate, make a measurement,
↪   encrypt the measured bit using the
    public key component of the classical homomorphic encryption
↪   scheme and store the ciphertext in a list, which will
    be returned at the end of the module.
    For more information, refer to the EPR scheme by Broadbent
↪   and Jeffery: https://arxiv.org/abs/1412.8766

    Args:
        - circuits: description of the quantum circuit as a list
↪   of strigs. Each element of the list represents one layer
        of the circuit.
        - enc_psi: a vector representation of the encrypted pure
↪   state |psi>_{enc}
        - a_tilde: a list of classical ciphertexts which contain
↪   the information about the X-Pauli gate exponents
        - b_tilde: a list of classical ciphertexts which contain
↪   the information about the Z-Pauli gate exponents

    Returns:
        - rho_hom_eval: density matrix of the encrypted evaluated
↪   quantum state
        - a_tilde_update: a list of updated classical ciphertexts
        which contain the information about the X-Pauli gate
↪   exponents
        - b_tilde_update: a list of updated classical ciphertexts
        which contain the information about the Z-Pauli gate
↪   exponents
        - circuit_dictionary: a dictonary object where keys
↪   indicate the ordering of qubits, starting from 1 as a string.
        The values associated with each key is a list which
↪   contains the following information, in the same order:
            1) acting wire
            2) a Boolean True/False value indicating whether a T
↪   gate has been applied during evaluation or not.
            This value gets updated during the evaluation
↪   process.
            3) the total number of T gates that are going to be
↪   applied on the qubit as an integer.
```

```
            4) the number of T gates applied at the end of the
↪   execution of each layer. This value gets updated
            during the evaluation process.

        If there are any T gates in the circuit:
            - t_gate_sequence_over_time: a nested list where each
↪   element is a list with the order of the qubit and the
        order of the layer in which it appears in the circuit.
            - c_encrypted: a list of ciphertexts resulting from
↪   the measurement during the evaluation of the EPR scheme
            - p_exponents: a list of ciphertexts recording the
↪   updated "a_tilde" value before the application of a
            T gate.
    """
    # Define variables for bookkeeping
    flat_circuit = flatten_circuit(circuits)
    acting_wire_dictionary = wire_dictionary(circuits)
    number_of_qubits = no_qubits(circuits)
    t_gate_count = number_of_tgates(circuits)
    number_of_wires = no_wires(circuits)
    individual_gates = tcl_layers(circuits)
    number_of_layers = len(individual_gates)
    t_index_list = tgate_index(acting_wire_dictionary)

    # initialize list
    to_be_encrypted = [0] * (number_of_layers + 1)
    c_encrypted = [[] for i in range(number_of_qubits)] #
    ↪   encrypted to-be-measured values of c will be stored in
    ↪   this list


    # if there are no T gates in the circuit, then apply the
    ↪   Clifford scheme evaluation procedure
    if t_gate_count == 0:
        return cl_eval(circuits, enc_psi, a_tilde, b_tilde)


    elif t_gate_count > 0:
        p_exponents = [[] for i in range(number_of_qubits)] #
        ↪   phase exponents will be stored in this list which
        ↪   then will be used for decryption

        # initialize the first phase gate exponent according to
        ↪   the a_tilde values for any
        # qubit on which a T gate is going to be applied
        for i in t_index_list:
```

```
            p_exponents[i] = [a_tilde[i]]

        # append EPR pairs to the system
        enc_psi_eval = np.kron(enc_psi,
        ↪  iterated_tensor_prod(gen_epr(), t_gate_count))

        # define lists for storing ciphertexts and key updates
        a_tilde_intermediate_update =
        ↪  [copy.deepcopy(to_be_encrypted) for i in
        ↪  range(number_of_qubits)]
        b_tilde_intermediate_update =
        ↪  [copy.deepcopy(to_be_encrypted) for i in
        ↪  range(number_of_qubits)]

        for i in range(number_of_qubits):
            a_tilde_intermediate_update[i][0] = a_tilde[i]
            b_tilde_intermediate_update[i][0] = b_tilde[i]

    # density matrix of the encrypted state |psi>_{enc}
    rho_hom_eval = pure_to_density(enc_psi_eval)

    # apply SWAP gates
    for qubit_order in range(number_of_qubits):
        if acting_wire_dictionary[str(qubit_order+1)][2] == 0:
            continue
        else:
            # number of wires before the first T gate on the i^ht
            ↪  qubit
            n_w_b_t_f_t = number_of_qubits +
            ↪  number_of_t_gates_applied(acting_wire_dictionary,
            ↪  qubit_order) * 2

            # number of T gates on the i^th qubit
            n_t_q_n =
            ↪  acting_wire_dictionary[str(qubit_order+1)][2]

            starting_wire = n_w_b_t_f_t + 1
            current_t_wire = starting_wire
            for actingTwire in range(n_t_q_n * 2):
                current_t_wire = starting_wire + actingTwire
                for remaining_qubits in
                ↪  range(number_of_qubits-acting_wire_dictionary[str(qubit_order+1)][0]):
                    current_circuit = iterated_tensor_prod(I,
                    ↪  current_t_wire-2)
                    current_circuit = np.kron(current_circuit,
                    ↪  SWAP)
```

```
                current_circuit = np.kron(current_circuit,
                                          ↪  iterated_tensor_prod(I,number_of_wires
                                          ↪ -
                                          ↪  current_t_wire))
                rho_hom_eval =
                ↪  qgate_on_density(current_circuit,rho_hom_eval)
                current_t_wire = current_t_wire - 1

    # evaluation procedure begins
    for gate_order in range(len(individual_gates)):

        # we create a seperate variable to keep track of the
        ↪  order of the qubit
        # to handle the case when a CNOT is applied after which
        # the order of qubit will increase by 2.
        qubit_order = 0

        for quasi_qubit_order in
        ↪  range(len(individual_gates[gate_order])):
            if acting_wire_dictionary[str(qubit_order+1)][3] ==
            ↪  0:
                acting_wire =
                ↪  acting_wire_dictionary[str(qubit_order+1)][0]
                ↪  + (

                    ↪  2*number_of_t_gates_applied(acting_wire_dictionary,
                    ↪  qubit_order)
                )
            else:
                acting_wire =
                ↪  acting_wire_dictionary[str(qubit_order+1)][0]

            current_gate =
            ↪  individual_gates[gate_order][quasi_qubit_order]

            # Clifford Gate
            if current_gate != "T":
                if current_gate in ["I", "X", "Z"]:

                    ↪  a_tilde_intermediate_update[qubit_order][gate_order+1],

                    ↪  b_tilde_intermediate_update[qubit_order][gate_order+1]
                    ↪  =
                    clifford_key_updates("I",
```

```
                ↪  a_tilde_intermediate_update[qubit_order][gate_order],
                                        ↪  b_tilde_intermediate_update
                                        ↪  [qubit_order][gate_order])
                if current_gate == "I":
                    current_circuit = iterated_tensor_prod(I,
                    ↪  number_of_wires)


                elif current_gate == "X":
                    current_circuit = iterated_tensor_prod(
                    I, acting_wire-1)
                    current_circuit = np.kron(current_circuit,
                    ↪  X)
                    current_circuit =
                    ↪  np.kron(current_circuit,
                                                ↪  iterated_tensor_prod(
                                                ↪  I,number_of_wires
                                                ↪  -
                                                ↪  acting_wire))

                else:
                    current_circuit = iterated_tensor_prod(
                    I, acting_wire-1)
                    current_circuit = np.kron(current_circuit,
                    ↪  Z)
                    current_circuit =
                    ↪  np.kron(current_circuit,
                                               ↪  iterated_tensor_prod(
                                               I,number_of_wires
                                               ↪  -
                                               ↪  acting_wire))

                qubit_order += 1

            elif current_gate == "P":
                current_circuit = iterated_tensor_prod(I,
                ↪  acting_wire-1)
                current_circuit = np.kron(current_circuit, P)
                current_circuit = np.kron(current_circuit,
```

```
                                        ↪  iterated_tensor_prod(I,number_of_wires
                                        ↪  - acting_wire))


            ↪  a_tilde_intermediate_update[qubit_order][gate_order+1],
            ↪  b_tilde_intermediate_update[qubit_order][gate_order+1]
            ↪  = clifford_key_updates("P",
            ↪  a_tilde_intermediate_update[qubit_order][gate_order],

                                    ↪  b_tilde_intermediate_update

                                    ↪  [qubit_order][gate_order])


            qubit_order += 1


        elif current_gate == 'H':
            current_circuit = iterated_tensor_prod(I,
            ↪  acting_wire-1)
            current_circuit = np.kron(current_circuit, H)
            current_circuit = np.kron(current_circuit,
            ↪  iterated_tensor_prod(I,number_of_wires -
            ↪  acting_wire))



            ↪  a_tilde_intermediate_update[qubit_order][gate_order+1],
            ↪  b_tilde_intermediate_update[qubit_order][gate_order+1]
            ↪  = clifford_key_updates("H",
            ↪  a_tilde_intermediate_update[qubit_order][gate_order],

                                    ↪  b_tilde_intermediate_update

                                    ↪  [qubit_order][gate_order])


            qubit_order += 1

        # CNOT
        else:
            control_wire = acting_wire
            if
            ↪  acting_wire_dictionary[str(qubit_order+2)][3]
            ↪  == 0:
```

```
                    target_wire =
                    ↪  acting_wire_dictionary[str(qubit_order+2)][0]
                    ↪  + (

                        ↪  2*number_of_t_gates_applied(acting_wire_dictionary,
                        ↪  qubit_order+1)
                    )
                else:
                    target_wire =
                    ↪  acting_wire_dictionary[str(qubit_order+2)][0]

                current_circuit =
                ↪  controlled_not_constructor(control_wire,
                ↪  target_wire, number_of_wires)


                temp1, temp2 = clifford_key_updates(
                    "CNOT",
                    ↪  [a_tilde_intermediate_update[qubit_order][gate_order],
                    ↪  a_tilde_intermediate_update[qubit_order+1][gate_order]],

                    ↪  [b_tilde_intermediate_update[qubit_order][gate_order],
                    ↪  b_tilde_intermediate_update[qubit_order+1][gate_order]]
                )


                ↪  a_tilde_intermediate_update[qubit_order][gate_order+1]
                ↪  = temp1[0]

                ↪  a_tilde_intermediate_update[qubit_order+1][gate_order+1]
                ↪  = temp1[1]

                ↪  b_tilde_intermediate_update[qubit_order][gate_order+1]
                ↪  = temp2[0]

                ↪  b_tilde_intermediate_update[qubit_order+1][gate_order+1]
                ↪  = temp2[1]

                qubit_order += 2

            rho_hom_eval =
            ↪  qgate_on_density(current_circuit,rho_hom_eval)

        elif current_gate == "T":
            # The following two 'if' statements store the
            ↪  `a_tilde_intermediate_update` ciphertext
```

```
# before applying the T gate:
# - if a T gate has not been yet applied to this
↪  qubit, then create a fresh list
# - if a T gate has been applied at this point of
↪  the evaluation then append to the list
if acting_wire_dictionary[str(qubit_order+1)][3]
↪  == 0:
    p_exponents[qubit_order] =
    ↪  [a_tilde_intermediate_update[qubit_order][gate_order]]

if acting_wire_dictionary[str(qubit_order+1)][3]
↪  > 0:
    p_exponents[qubit_order].append(

    ↪  a_tilde_intermediate_update[qubit_order][gate_order])

# keeping track of the control wire and target
↪  wire
if acting_wire_dictionary[str(qubit_order+1)][1]
↪  == True:
    target_wire = acting_wire
    control_wire = acting_wire + 2

if acting_wire_dictionary[str(qubit_order+1)][1]
↪  == False:
    acting_wire_dictionary[str(qubit_order+1)][1]
    ↪  = True
    no_tgates_before_ith_qubit =
    ↪  number_of_t_gates_applied(acting_wire_dictionary,
    ↪  qubit_order)
    target_wire = acting_wire
    control_wire = (qubit_order) +
    ↪  2*no_tgates_before_ith_qubit + 2


# apply the T gate
t_gate_operator = np.kron(iterated_tensor_prod(I,
↪  target_wire-1), T)
t_on_rho_hom_eval = qgate_on_density(np.kron(
t_gate_operator,iterated_tensor_prod(I,
↪  number_of_wires - target_wire)),
↪  rho_hom_eval)

# apply the CNOT gate
```

```
cnot_on_t_rho_hom_eval =
↪  qgate_on_density(controlled_not_constructor(control_wire,
↪  target_wire, number_of_wires),
↪  t_on_rho_hom_eval)

# make measurement and obtain the
↪  post-measurement state and the classical bit
↪  c
pms, c =
↪  post_measurement_state(cnot_on_t_rho_hom_eval,
↪  [target_wire-1],

                          ↪  measurement(partial_trace(
                          cnot_on_t_rho_hom_eval,
                          ↪  [target_wire-1])))

# encrypt bit c using the public key
c_tilde = encrypt_one_bit([int(c)])

# append the ciphertext to a list storing the
↪  encryption of measured 'c' values
c_encrypted[qubit_order].append(c_tilde[0])

# update `a_tilde` value homomorphically
a_tilde_add_c =
↪  evaluator.add(a_tilde_intermediate_update[qubit_order][gate_order],
↪  c_tilde[0])

# update `b_tilde` value homomorphically
a_tilde_times_c_tilde =
↪  evaluator.multiply(a_tilde_intermediate_update[qubit_order][gate_order],
↪  c_tilde[0], relin_key)
a_tilde_times_c_tilde_add_a_tilde =
↪  evaluator.add(a_tilde_times_c_tilde,
↪  a_tilde_intermediate_update[qubit_order][gate_order])
a_tilde_times_c_tilde_add_a_tilde_add_b_tilde =
↪  evaluator.add(

    ↪  b_tilde_intermediate_update[qubit_order][gate_order],
    ↪  a_tilde_times_c_tilde_add_a_tilde)

# update the lists

↪  a_tilde_intermediate_update[qubit_order][gate_order+1]
↪  = a_tilde_add_c
```

```
                b_tilde_intermediate_update[qubit_order][gate_order+1]
                =
                a_tilde_times_c_tilde_add_a_tilde_add_b_tilde

                # perform the necessary updates
                acting_wire_dictionary[str(qubit_order+1)][0] =
                control_wire
                acting_wire_dictionary[str(qubit_order+1)][3] =
                acting_wire_dictionary[str(qubit_order+1)][3]
                + 1
                rho_hom_eval = pms
                qubit_order += 1

    # retain the last key updates
    a_tilde_update = [a_tilde_intermediate_update[i][-1] for i in
    range(number_of_qubits)]
    b_tilde_update = [b_tilde_intermediate_update[i][-1] for i in
    range(number_of_qubits)]

    # fresh copy of the circuit dictionary required for
    decryption
    circuit_dictionary = wire_dictionary(circuits)

    t_gate_sequence_over_time = t_gate_sequence(circuits)
    key_polynomials = phase_gate_polynomial(circuits)
    return (rho_hom_eval, a_tilde_update, b_tilde_update,
    circuit_dictionary, key_polynomials,
    t_gate_sequence_over_time, c_encrypted, p_exponents)
```

E.6 epr_decryption

```
    def cl_dec(rho_hom_eval, a_tilde_update, b_tilde_update):
    """
    Perform decryption of the encrypted evaluated cipherstate
    according to the Clifford scheme and
    return the decrypted evaluated state.
    For more information refer to the Clifford scheme by
    Broadbent and Jeffery: https://arxiv.org/abs/1412.8766

    Args:
        - rho_hom_eval: density matrix of the encrypted evaluated
    quantum state
        - a_tilde_update: a list of updated classical ciphertexts
        which contain the information about the X-Pauli gate
    exponents
```

```
        - b_tilde_update: a list of updated classical ciphertexts
        which contain the information about the Z-Pauli gate
↪   exponents

    Returns:
        - rho_dec_eval: evaluated decrypted density matrix of the
↪   input message
    """
    # decrypt the ciphertexts
    a_decrypted = [decryptor.decrypt(i) for i in a_tilde_update]
    b_decrypted = [decryptor.decrypt(i) for i in b_tilde_update]

    a_decoded = [encoder.decode(i) for i in a_decrypted]
    b_decoded = [encoder.decode(i) for i in b_decrypted]

    # extract the first bit from each list
    a_dec = [i[0] for i in a_decoded]
    b_dec = [i[0] for i in b_decoded]

    rho_dec_eval = qotp_dec_density(rho_hom_eval, a_dec, b_dec)
    return rho_dec_eval

    def number_of_tgates_dictionary(circuit_dictionary):
    """
    Count the total number of T gates in a circuit from the
↪   circuit dictionary.

    Args:
        - circuit_dictionary: a dictonary object where keys
↪   indicate the ordering of qubits,
        starting from 1 as a string. The values associated with
↪   each key is a list which contains
        the following information, in the same order:
            1) acting wire
            2) a Boolean True/False value indicating whether a T
↪   gate has been applied during evaluation or not.
            This value gets updated during the evaluation
↪   process.
            3) the total number of T gates that are going to be
↪   applied on the qubit as an integer.
            4) the number of T gates applied at the end of the
↪   execution of each layer. This value gets updated
            during the evaluation process.

    Returns:
```

```
        - number_of_t_gates: the total number of T gates in a
↪   circuit as an integer
    """
    number_of_t_gates = 0
    for key in circuit_dictionary:
        number_of_t_gates += circuit_dictionary[key][2]  # Access
        ↪  the third element of the list
    return number_of_t_gates

    def epr_dec(rho_hom_eval, a_tilde_update, b_tilde_update,
    ↪  circuit_dictionary,
    ↪  key_polynomials=None,t_gate_sequence_over_time=None,
    ↪  c_encrypted=None, p_exponents=None):
    """
    Perform decryption of the encrypted evaluated cipherstate
↪   according to the EPR scheme.
    If the evaluated circuit does not consist of any T gates,
↪   then an application of the
    quantum one-time pad is sufficient for decryption.
    If there is at least one T gate in the circuit, then phase
↪   gates, conditional
    on whether a X Paulis were applied in the encryption of the
↪   quantum message, is required.
    A symbolic computation is required to assess whether a
↪   correction is needed for the application of
    the phase gate. After running a circuit consisting of a
↪   potential Phase gate and a Hadamard gate,
    a measurement is performed to obtain the unknown `k` value,
↪   which then will be stored in a list.
    At the end, one last check is required before the performing
↪   the quantum one-time pad on
    `a_tilde_update` and `b_tilde_update`. The evaluated
↪   decrypted density matrix of the input message is returned
    at the end.
    For more information, refer to the EPR scheme by Broadbent
↪   and Jeffery: https://arxiv.org/abs/1412.8766.


    Args:
        - rho_hom_eval: density matrix of the encrypted evaluated
↪   quantum state
        - a_tilde_update: a list of updated classical ciphertexts
        which contain the information about the X-Pauli gate
↪   exponents
        - b_tilde_update: a list of updated classical ciphertexts
```

```
        which contain the information about the Z-Pauli gate
↪   exponents
        - circuit_dictionary: a dictonary object where keys
↪   indicate the ordering of qubits, starting from 1 as a string.
        The values associated with each key is a list which
↪   contains the following information, in the same order:
            1) acting wire
            2) a Boolean True/False value indicating whether a T
↪   gate has been applied during evaluation or not.
            This value gets updated during the evaluation
↪   process.
            3) the total number of T gates that are going to be
↪   applied on the qubit as an integer.
            4) the number of T gates applied at the end of the
↪   execution of each layer. This value gets updated
            during the evaluation process.

        If there are any T gates in the circuit:
            - key_polynomials: a list containing the symbolic
↪   polynomial before the applicaton
        of a T gate and all the key updates at the end of the
↪   circuit evaluation.
            - t_gate_sequence_over_time: a nested list where each
↪   element is a list with the order of the qubit and the
        order of the layer in which it appears in the circuit.
            - c_encrypted: a list of ciphertexts resulting from
↪   the measurement during the evaluation of the EPR scheme
            - p_exponents: a list of ciphertexts recording the
↪   updated "a_tilde" value before the application of a
            T gate.

    Returns:
        - rho_dec_eval: evaluated decrypted density matrix of the
↪   input message
    """
    # making a copy of the `circuit_dictionary`
    acting_wire_dictionary = circuit_dictionary

    t_gate_count =
    ↪  number_of_tgates_dictionary(acting_wire_dictionary)
    if t_gate_count == 0:
        return cl_dec(rho_hom_eval, a_tilde_update,
        ↪  b_tilde_update)

    # Define variables for bookkeeping
    t_index_list = tgate_index(acting_wire_dictionary)
```

```
number_of_qubits =
↪  int(list(acting_wire_dictionary.keys())[-1])
number_of_wires = int(np.log2(rho_hom_eval.shape[0]))
c_dec = [[] for i in range(number_of_qubits)]
k_measurements = [[] for i in range(number_of_qubits)]
phase_dec = [[] for i in range(number_of_qubits)]


# decrypt the ciphertexts
a_dec = [encoder.decode(decryptor.decrypt(i))[0]%2 for i in
↪  a_tilde_update]
b_dec = [encoder.decode(decryptor.decrypt(i))[0]%2 for i in
↪  b_tilde_update]
for qubit_order in range(len(t_index_list)):
    c_dec[t_index_list[qubit_order]] =
    ↪  [encoder.decode(decryptor.decrypt(j))[0] for j in
    ↪  c_encrypted[t_index_list[qubit_order]]]
    phase_dec[t_index_list[qubit_order]] =
    ↪  [encoder.decode(decryptor.decrypt(j))[0] % 2 for j in
    ↪  p_exponents[t_index_list[qubit_order]]]

rho_dec_eval = rho_hom_eval


# decryption begins here
for i in range(len(t_gate_sequence_over_time)):
    qubit_order = t_gate_sequence_over_time[i][0]

    if acting_wire_dictionary[str(qubit_order+1)][3] == 0:
        acting_wire_dictionary[str(qubit_order+1)][1] = True
        acting_wire_dictionary[str(qubit_order+1)][3] =
        ↪  acting_wire_dictionary[str(qubit_order+1)][3] + 1
        no_t_gates_before_ith_qubit =
        ↪  number_of_t_gates_applied(acting_wire_dictionary,
        ↪  qubit_order)
        acting_wire = qubit_order +
        ↪  2*no_t_gates_before_ith_qubit + 3
        temp_list =
        ↪  acting_wire_dictionary[str(qubit_order+1)]
        temp_list.append(acting_wire)
        acting_wire_dictionary[str(qubit_order+1)] =
        ↪  temp_list

    else:
        acting_wire =
        ↪  acting_wire_dictionary[str(qubit_order+1)][4] + 2
```

```
    acting_wire_dictionary[str(qubit_order+1)][3] =
    ↪  acting_wire_dictionary[str(qubit_order+1)][3] + 1
    acting_wire_dictionary[str(qubit_order+1)][4] =
    ↪  acting_wire

t_gate_count_on_ith_qubit =
↪  acting_wire_dictionary[str(qubit_order+1)][3] - 1
current_polynomial =
↪  key_polynomials[qubit_order][t_gate_count_on_ith_qubit]

phase_gate_exponent_correction =
↪  evaluate_expression(current_polynomial,
↪  k_measurements, c_dec)
p_exponent_value =
↪  phase_dec[qubit_order][t_gate_count_on_ith_qubit] ^
↪  phase_gate_exponent_correction

p_operator =
↪  np.kron(iterated_tensor_prod(I,acting_wire-1),
                    matrix_power(P, p_exponent_value))
p_operator = np.kron(p_operator,
↪  iterated_tensor_prod(I,number_of_wires-acting_wire))
phase_on_rho_dec_eval = qgate_on_density(p_operator,
↪  rho_dec_eval)


h_operator =
↪  np.kron(iterated_tensor_prod(I,acting_wire-1), H)
h_operator = np.kron(h_operator,
↪  iterated_tensor_prod(I,number_of_wires-acting_wire))
h_on_phase_rho_dec_eval = qgate_on_density(h_operator,
↪  phase_on_rho_dec_eval)


pms, k = post_measurement_state(h_on_phase_rho_dec_eval,
                                [acting_wire - 1],  # -1
                                ↪  to account for
                                # acting_wire starts from
                                ↪  1
                                # python indexing starts
                                ↪  from 0
                                measurement(partial_trace(
                                h_on_phase_rho_dec_eval,
                                ↪  [acting_wire - 1])))
```

```
        k_measurements[qubit_order].append(int(k))
        rho_dec_eval = pms

    # applying necessary correction to the last layer of the
    ↪  circuit
    for qubit_order in range(number_of_qubits):
        a_exponent_poly = key_polynomials[qubit_order][-1][0]
        b_exponent_poly = key_polynomials[qubit_order][-1][1]
        a_exponent_correction =
        ↪  evaluate_expression(a_exponent_poly, k_measurements,
        ↪  c_dec)
        b_exponent_correction =
        ↪  evaluate_expression(b_exponent_poly, k_measurements,
        ↪  c_dec)

        a_dec[qubit_order] = a_dec[qubit_order] ^
        ↪  a_exponent_correction
        b_dec[qubit_order] = b_dec[qubit_order] ^
        ↪  b_exponent_correction

    # extracting the order of wires that are not traced out after
    ↪  partial trace
    acting_wires = []
    for i in range(number_of_qubits):
        if len(acting_wire_dictionary[str(i+1)]) == 5:

            ↪  acting_wires.append(acting_wire_dictionary[str(i+1)][4]-2)
        if len(acting_wire_dictionary[str(i+1)]) == 4:
            acting_wires.append(i + 2 *
            ↪  number_of_t_gates_applied(acting_wire_dictionary,
            ↪  i))

    rho_dec_eval = qotp_dec_density(partial_trace(rho_dec_eval,
    ↪  acting_wires),
                                    a_dec,
                                    b_dec)

    return rho_dec_eval
```

E.7 epr_scheme

```
%run quantum_one_time_pad.ipynb
%run che_initialization.ipynb
%run epr_encryption.ipynb
%run epr_evaluation.ipynb
%run epr_decryption.ipynb
```

```
    def epr_quantum_homomorphic_scheme(psi, a, b, circuits):
    """
    Homomorphically apply the quantum circuit provided by
↪   `circuits` on the quantum pure state
    |psi>, which is encrypted with bits `a` and `b` using the
↪   quantum one-time pad according to the EPR scheme.
    For more information refer to the Clifford scheme by
↪   Broadbent and Jeffery: https://arxiv.org/abs/1412.8766

    Args:
        - psi: a vector representation of the pure state |psi>
        - a: list of (randomly generated) classical bits
        - b: list of (randomly generated) classical bits
        circuits: description of the quantum circuit as a list of
↪   strigs. Each element of the list represents one layer
        of the circuit.

    Returns:
        - rho_dec_eval: evaluated decrypted density matrix of the
↪   input message
    """
    psi_enc, a_enc, b_enc = epr_enc(psi,a,b)
    rho_hom_eval, a_tilde_update, b_tilde_update,
    ↪   circuit_dictionary, key_polynomials,
    ↪   t_gate_sequence_over_time,  c_encrypted, p_exponents =
    ↪   epr_eval(circuits,psi_enc, a_enc, b_enc)
    rho_dec_eval = epr_dec(rho_hom_eval, a_tilde_update,
    ↪   b_tilde_update, circuit_dictionary, key_polynomials,
    ↪   t_gate_sequence_over_time, c_encrypted, p_exponents)
    return rho_dec_eval
```

References

[Aar06] S. Aaronson. Quantum computing since Democritus: Lecture 19, 2006. Accessed 8 July 2024.
https://www.scottaaronson.com/democritus/lec19.html

[AAUC19] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM computing surveys*, vol. 51, no. 4, pp. 1–35, 2019.

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[ABC+15] F. Armknecht *et al.* A guide to fully homomorphic encryption. Cryptology ePrint Archive, Paper 2015/1192, 2015.
https://eprint.iacr.org/2015/1192

[ADSS17] G. Alagic, Y. Dulek, C. Schaffner, and F. Speelman. Quantum fully homomorphic encryption with verification. In *Advances in Cryptology — ASIACRYPT 2017*, vol. 1, pp. 438–467, 2017.
doi:10.1007/978-3-319-70694-8_16

[AF15] M. Anderson and T. Feil. *A first course in abstract algebra : rings, groups, and fields*. CRC Press, 2015.

[AKS04] M. Agrawal, N. Kayal, and N. Saxena. Primes is in P. *Annals of Mathematics*, vol. 160, no. 2, pp. 781–793, 2004.

[ATL12] ATLAS Collaboration. Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc. *Physics Letters B*, vol. 716, p. 1–29, 2012.
doi:10.1016/j.physletb.2012.08.020

[BB84] C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. In *International Conference on Computers, Systems and Signal Processing*, pp. 175–179, 1984.

[BBT05] G. Brassard, A. Broadbent, and A. Tapp. Quantum pseudo-telepathy. *Foundations of Physics*, vol. 35, no. 11, pp. 1877–1907, 2005.
doi:10.1007/s10701-005-7353-4

[BDCG+20] S. Ben-David, A. M. Childs, A. Gilyén, W. Kretschmer, S. Podder, and D. Wang. Symmetries, graph properties, and quantum speedups. In *2020 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2020)*, pp. 649–660, 2020.
doi:10.1109/FOCS46700.2020.00066

[BGHD+23] K. Barooti *et al.* Public-key encryption with quantum keys. In *Theory of Cryptography*, pp. 198–227, 2023.
doi:10.1007/978-3-031-48624-1_8

[Bil95] P. Billingsley. *Probability and Measure*. John Wiley & Sons, 3rd edition, 1995.

[BIS+22] V. Bergholm *et al.* Pennylane: Automatic differentiation of hybrid quantum-classical computations. 2022.
arXiv:1811.04968 [quant-ph]

[BJ14] A. Broadbent and S. Jeffery. Quantum homomorphic encryption for circuits of low T-gate complexity. 2014.
arXiv:1412.8766 [quant-ph]

[BJ15] A. Broadbent and S. Jeffery. Quantum homomorphic encryption for circuits of low T-gate complexity. In *Advances in Cryptology — CRYPTO 2015*, vol. 2, pp. 609–629, 2015.
doi:10.1007/978-3-662-48000-7_30

[BR03] P. O. Boykin and V. Roychowdhury. Optimal encryption of quantum bits. *Physical Review A*, vol. 67, no. 4, p. 042317, 2003.
doi:10.1103/PhysRevA.67.042317

[Bra12] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology — CRYPTO 2012*, vol. 1, pp. 868–886, 2012.
doi:10.1007/978-3-642-32009-5_50

[Bro19] A. Broadbent. Award, 2019. Accessed 7 September 2024.
https://mysite.science.uottawa.ca/abroadbe/award.php

[BS12] S. Burris and H. P. Sankappanavar. *A course in Universal Algebra*. Springer-Verlag, 2012.
https://www.math.uwaterloo.ca/~snburris/htdocs/UALG/univ-algebra2012.pdf

[BS23] D. Boneh and V. Shoup. A graduate course in applied cryptography, 2023.
https://toc.cryptobook.us/book.pdf

[BV11] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pp. 97–106. IEEE, 2011.

[Chi05] A. M. Childs. Secure assisted quantum computation. *Quantum Information & Computation*, vol. 5, no. 6, pp. 456–466, 2005.
doi:10.26421/QIC5.6-4

[CHSH69] J. F. Clauser, M. A. Horne., A. Shimony, and R. A. Holt. Proposed experiment to test local hidden-variable theories. *Physical Review Letters*, vol. 23, no. 15, pp. 880–884, 1969.
doi:10.1103/PhysRevLett.23.880

[Cir24] Cirq Developers. Cirq, 2024.
doi:https://zenodo.org/records/11398048

[Deu85] D. Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. Series A, Mathematical and physical sciences*, vol. 400, no. 1818, pp. 97–117, 1985.

[Dev12] J. L. Devore. *Probability and statistics for engineering and the sciences*. Brooks/Cole, Cengage Learning, 8th ed. edition, 2012.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.
doi:10.1109/TIT.1976.1055638

[DM99] L. Debnath and P. Mikusiński. *Introduction to Hilbert Spaces with Applications*. Academic Press, 1999.

[DMGD23] T. V. T. Doan, M.-L. Messai, G. Gavin, and J. Darmont. A survey on implementations of homomorphic encryption schemes. *The Journal of supercomputing*, vol. 79, no. 13, pp. 15098–15139, 2023.

[DSS16] Y. Dulek, C. Schaffner, and F. Speelman. Quantum homomorphic encryption for polynomial-sized circuits. In *Advances in Cryptology — CRYPTO 2016*, pp. 3–32, 2016.
doi:10.1007/978-3-662-53015-3_1

[EB64] F. Englert and R. Brout. Broken symmetry and the mass of gauge vector mesons. *Physical Review Letters*, vol. 13, pp. 321–323, 1964.

[EPR35] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Physical Review Letters*, vol. 47, no. 10, pp. 777–780, 1935.
doi:10.1103/physrev.47.777

[Era20a] S. Erabelli. py-fhe. GitHub, 2020.
https://github.com/sarojaerabelli/py-fhe

[Era20b] S. Erabelli. pyfhe - a Python library for fully homomorphic encryption. Master's thesis, Massachusetts Institute of Technology, 2020.
https://dspace.mit.edu/bitstream/handle/1721.1/129204/1227275316-MIT.pdf

[Fay24] J. Faye. Copenhagen Interpretation of Quantum Mechanics. In E. N. Zalta and U. Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2024 edition, 2024.

[Fey82] R. P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, vol. 21, no. 6, pp. 467–488, 1982. doi:10.1007/BF02650179

[FMD24] P. Fernández and M. A. Martin-Delgado. Implementing the grover algorithm in homomorphic encryption schemes. 2024. arXiv:2403.04922 [quant-ph]

[FV12] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. http://eprint.iacr.org/2012/144

[Gan24] S. Ganjian. py-qhe-epr. GitHub, 2024. https://github.com/sohrabganjian/py-qhe-epr

[GE21] C. Gidney and M. Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, vol. 5, p. 433, 2021. doi:10.22331/q-2021-04-15-433

[Gen09] C. Gentry. *Fully Homomorphic Encryption Using Ideal Lattices*. PhD thesis, Stanford University, 2009. https://crypto.stanford.edu/craig/craig-thesis.pdf

[GHK64] G. S. Guralnik, C. R. Hagen, and T. W. B. Kibble. Global conservation laws and massless particles. *Physical Review Letters*, vol. 13, pp. 585–587, 1964.

[Gol08] O. Goldreich. *Computational complexity : a conceptual perspective*. Cambridge University Press, 2008.

[Got98a] D. Gottesman. The Heisenberg representation of quantum computers. In *22nd International Colloquium on Group Theoretical Methods in Physics—GROUP 22*, pp. 32–43, 1998.

[Got98b] D. Gottesman. Theory of fault-tolerant quantum computation. *Physical Review A*, vol. 57, pp. 127–137, 1998. doi:10.1103/PhysRevA.57.127

[GPB24] S. Ganjian, C. Paddock, and A. Broadbent. Demonstrating quantum homomorphic encryption through simulation. 2024. arXiv preprint. To appear in QCE24. arXiv:2406.16247 [quant-ph]

[Gro96] L. K. Grover. A fast quantum mechanical algorithm for database search. In *STOC '96: Proceedings of the twenty-eighth ACM symposium on Theory of computing*, pp. 212–219, 1996. doi:10.1145/237814.237866

[GS22] D. Grier and L. Schaeffer. The classification of Clifford gates over qubits. *Quantum*, vol. 6, p. 734, 2022. doi:10.22331/q-2022-06-13-734

[GSW13] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology — CRYPTO 2013*, vol. 1, pp. 75–92, 2013. doi:10.1007/978-3-642-40041-4_5

[Gö31] K. Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für Mathematik und Physik*, vol. 38, pp. 173–198, 1931. doi:10.1007/BF01700692

[Hig64] P. W. Higgs. Broken symmetries and the masses of gauge bosons. *Physical Review Letters*, vol. 13, pp. 508–509, 1964.

[HMvdW+20] C. R. Harris *et al.* Array programming with NumPy. *Nature*, vol. 585, no. 7825, pp. 357–362, 2020. doi:10.1038/s41586-020-2649-2

[HPS98] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium—ANTS 1998*, pp. 267–288, 1998. doi:10.1007/BFb0054868

[Ion24] IonQ. The value of classical quantum simulators, 2024. https://ionq.com/resources/the-value-of-classical-quantum-simulators

[IP07] Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *Theory of Cryptography*, pp. 575–594. Springer Berlin Heidelberg, 2007.

[Jac85] N. Jacobson. *Basic Algebra*. W.H. Freeman, 2nd edition, 1985.

[JATK+24] A. Javadi-Abhari *et al.* Quantum computing with Qiskit. 2024. arXiv:2405.08810 [quant-ph]

[JCJ14] T. H. Johnson, S. R. Clark, and D. Jaksch. What is a quantum simulator? *European Physical Journal Quantum Technology*, vol. 1, pp. 1–, 2014.

[KL21] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. CRC Press, 3rd edition, 2021.

[KY23] T. Khattar and N. Yosri. A comment on "factoring integers with sublinear resources on a superconducting quantum processor". 2023.
arXiv:2307.09651 [quant-ph]

[Lan86] S. Lang. *Introduction to Linear Algebra*. Springer, 2nd edition, 1986.

[LaR18] R. LaRose. Lecture notes: Quantum states and partial trace, 2018. Accessed 5 June 2024.
https://www.ryanlarose.com/uploads/1/1/5/8/115879647/quic06-states-trace.pdf

[Lay14] S. R. Lay. *Analysis: With an Introduction to Proof*. Pearson, 5th edition, 2014.

[LD24] O. Lombardi and D. Dieks. Modal Interpretations of Quantum Mechanics. In *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2024 edition, 2024.

[Lia15] M. Liang. Quantum fully homomorphic encryption scheme based on universal quantum circuit. *Quantum Information Processing*, vol. 14, no. 8, p. 2749–2759, 2015.
doi:10.1007/s11128-015-1034-9

[Lia20] M. Liang. Teleportation-based quantum homomorphic encryption scheme with quasi-compactness and perfect security. *Quantum Information Processing*, vol. 19, no. 1, p. 28, 2020.
doi:10.1007/s11128-019-2529-6

[Mar11] A. Maruoka. *Concise Guide to Computation Theory*. Springer London, 2011.

[Mil76] G. L. Miller. Riemann's hypothesis and tests for primality. *jcss*, vol. 13, no. 3, pp. 300–317, 1976.
doi:https://doi.org/10.1016/S0022-0000(76)80043-8

[ML22] G. Ma and H. Li. Quantum fully homomorphic encryption by integrating Pauli one-time pad with quaternions. *Quantum*, vol. 6, p. 866, 2022.
doi:10.22331/q-2022-12-01-866

[MSM+22] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. P. Fitzek, and N. Aaraj. Survey on fully homomorphic encryption, theory, and applications. *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1572–1609, 2022.

[MSP+17] A. Meurer *et al.* SymPy: symbolic computing in Python. *PeerJ Computer Science*, vol. 3, p. e103, 2017.
doi:10.7717/peerj-cs.103

[Nat16] National Institute of Standards and Technology. NIST announces the release of DRAFT NISTIR 8105, report on post-quantum cryptography for public comment, 2016. Accessed 2 September 2024.
https://csrc.nist.gov/News/2016/NIST-Announce-the-Release-of-DRAFT-NISTIR-8105

[Nat24] National Institute of Standards and Technology. NIST releases first 3 finalized post-quantum encryption standards, 2024. Accessed 2 September 2024.
https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards

[NC10] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 10th anniversary edition, 2010.
doi:10.1017/CBO9780511976667

[Nob24] Nobel Prize Outreach AB. Wolfgang pauli – biographical. *NobelPrize.org*, 2024. Accessed 7 September 2024.
https://www.nobelprize.org/prizes/physics/1945/pauli/biographical/

[Noe21] E. Noether. Idealtheorie in Ringbereichen. *Mathematische annalen*, vol. 83, no. 1-2, pp. 24–66, 1921.

[OR01] J. J. O'Connor and E. F. Robertson. Charles hermite. *MacTutor History of Mathematics, University of St Andrews, Scotland*, 2001. Accessed 7 September 2024.
https://mathshistory.st-andrews.ac.uk/Biographies/Hermite/

[OR03] J. J. O'Connor and E. F. Robertson. Jacques salomon hadamard. *MacTutor History of Mathematics, University of St Andrews, Scotland*, 2003. Accessed 7 September 2024.
https://mathshistory.st-andrews.ac.uk/Biographies/Hadamard/

[OR15] J. J. O'Connor and E. F. Robertson. William kingdon clifford. *MacTutor History of Mathematics, University of St Andrews, Scotland*, 2015. Accessed 7 September 2024.
https://mathshistory.st-andrews.ac.uk/Biographies/Clifford/

[Oxf24a] Oxford English Dictionary. -graphy, 2024.
https://www.oed.com/dictionary/graphy_combform

[Oxf24b] Oxford English Dictionary. crypto-, 2024.
https://www.oed.com/dictionary/crypto_combform

[Oxf24c] Oxford English Dictionary. cryptography, n., 2024.
https://www.oed.com/dictionary/cryptography_n

[Oxf24d] Oxford English Dictionary. homomorphism, n., 2024.
https://www.oed.com/dictionary/homomorphism_n

[Oxf24e] Oxford English Dictionary. quantum, 2024.
https://www.oed.com/dictionary/quantum_n

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology — EUROCRYPT '99*, pp. 223–238. Springer Berlin Heidelberg, 1999.

[Pre18] J. Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, vol. 2, p. 79, 2018.
doi:10.22331/q-2018-08-06-79

[Pyt24] Python Software Foundation. Python, 2024.
https://www.python.org/

[Rab80] M. O. Rabin. Probabilistic algorithm for testing primality. *jnt*, vol. 12, no. 1, pp. 128–138, 1980.
doi:https://doi.org/10.1016/0022-314X(80)90084-0

[RAD78] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pp. 169–177. Academic Press, 1978.

[Reg05] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC '05: Proceedings of the thirty-seventh ACM symposium on Theory of computing*, pp. 84–93, 2005.
doi:10.1145/1060590.1060603

[RFG12] P. P. Rohde, J. F. Fitzsimons, and A. Gilchrist. Quantum walks with encrypted data. *Physical Review Letters*, vol. 109, p. 150501, 2012.
doi:10.1103/PhysRevLett.109.150501

[Ros11] K. H. Rosen. *Elementary number theory and its applications*. Addison-Wesley, 6th ed. edition, 2011.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
doi:10.1145/359340.359342

[Sch19] W. Scherer. *Mathematics of Quantum Computing An Introduction*. Springer International Publishing, 2019.

[Sho94] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science (FOCS 1994)*, pp. 124–134, 1994.
doi:10.1109/SFCS.1994.365700

[Sho99] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, vol. 41, no. 2, pp. 303–332, 1999.
doi:10.1137/S0036144598347011

[SIK22] SIKE Team. Sike foreword and postscript, 2022. Accessed 2 September 2024.
https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/round-4/submissions/sike-team-note-insecure.pdf

[SP19] D. R. Stinson and M. B. Paterson. *Cryptography : theory and practice*. CRC Press, Taylor & Francis Group, 4th edition, 2019.

[TFBF+20] W. K. Tham *et al.* Experimental demonstration of quantum fully homomorphic encryption with application in a two-party secure protocol. *Physical Review X*, vol. 10, no. 1, p. 011038, 2020.
doi:10.1103/PhysRevX.10.011038

[The24a] The Editors of Encyclopaedia Britannica. Charles hermite. *Encyclopedia Britannica*, 2024. Accessed 7 September 2024.
https://www.britannica.com/biography/Charles-Hermite

[The24b] The Editors of Encyclopaedia Britannica. Jacques-salmon hadamard. *Encyclopedia Britannica*, 2024. Accessed 7 September 2024.
https://www.britannica.com/biography/Jacques-Salomon-Hadamard

[The24c] The Editors of Encyclopaedia Britannica. William kingdon clifford. *Encyclopedia Britannica*, 2024. Accessed 7 September 2024.
https://www.britannica.com/biography/William-Kingdon-Clifford

[The24d] The Editors of Encyclopaedia Britannica. Wolfgang pauli. *Encyclopedia Britannica*, 2024. Accessed 7 September 2024.
https://www.britannica.com/biography/Wolfgang-Pauli

[The24e] The Planetary Society. Pale blue dot, 2024. Accessed 31 August 2024.
https://www.planetary.org/worlds/pale-blue-dot

[Tur37] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, vol. s2-42, no. 1, pp. 230–265, 1937.
doi:10.1112/plms/s2-42.1.230

[U.S21] U.S. Department of Defense. Venona: An overview, 2021. Accessed 1 July 2024.
https://media.defense.gov/2021/Jun/29/2002752012/-1/-1/0/VENONA_AN_OVERVIEW.PDF

[Vai21] L. Vaidman. Many-worlds interpretation of quantum mechanics. In *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition, 2021.

[vRWC13] G. van Rossum, B. Warsaw, and A. Coghlan. Pep 8 – style guide for Python code, 2013.
https://peps.python.org/pep-0008/

[Wat18] J. Watrous. *The Theory of Quantum Information*. Cambridge University Press, 2018.
doi:10.1017/9781316848142

[WN18] S. Wehner and N. Ng. edx quantum cryptography: Week 0, 2018.
https://learning.edx.org/course/course-v1:CaltechDelftX+QuCryptox+3T2018/home

[Wol21] R. Wolf. *Quantum Key Distribution: An introduction with exercises*. Springer, 2021.

[Yac21] R. Yackel. What is homomorphic encryption, and why isn't it mainstream?, 2021. Accessed 13 July 2024.
https://www.keyfactor.com/blog/what-is-homomorphic-encryption/

[YSE23] K. Young, M. Scese, and A. Ebnenasir. Simulating quantum computations on classical machines: A survey. 2023.
arXiv:2311.16505 [quant-ph]

[YTW+22] B. Yan *et al.* Factoring integers with sublinear resources on a superconducting quantum processor. 2022.
arXiv:2212.12372 [quant-ph]

[YUS22] M. Yarter, G. Uehara, and A. Spanias. Implementation and analysis of quantum homomorphic encryption. In *2022 13th International Conference on Information, Intelligence, Systems & Applications (IISA)*, pp. 1–5, 2022.
doi:10.1109/IISA56318.2022.9904399

[ZLC00] X. Zhou, D. W. Leung, and I. L. Chuang. Methodology for quantum logic gate construction. *Physical Review A*, vol. 62, no. 5, p. 052316, 2000.
doi:10.1103/PhysRevA.62.052316