Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias Físico Matemáticas

A Quantum Algorithm for Proof of Work

Tesis presentada al

Colegio de Física

como requisito parcial para la obtención del grado de

Licenciado en Física

por

Fabiola Cañete Leyva

Asesorada por

Dra. Areli Montes Pérez y Dr. Salvador E. Venegas Andraca

Puebla Pue.
Agosto de 2025

Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias Físico Matemáticas

A Quantum Algorithm for Proof of Work

Tesis presentada al

Colegio de Física

como requisito parcial para la obtención del grado de

Licenciado en Física

por

Fabiola Cañete Leyva

Asesorada por

Dra. Areli Montes Pérez y Dr. Salvador E. Venegas Andraca

Puebla Pue.
Agosto de 2025

Título: A Quantum Algorithm for Proof of Work
Estudiante: Fabiola Cañete Leyva

COMITÉ

Dr. Carlos Ignacio Robledo Sánchez
Presidente

Dra. Bárbara E. Sánchez Rinza
Secretario

Dra. Iraís Bautista Guzmán
Vocal

Dr. Carlos A. Pérez Delgado
Vocal

Dra. Areli Montes Pérez y Dr. Salvador E. Venegas Andraca
Asesor

Acknowledgements

A Dios, a mi familia y a mis profesores —en especial al Dr. Salvador E. Venegas-Andraca y al Dr. Carlos A. Pérez-Delgado— por su guía, inspiración y las conversaciones que dieron forma al camino de esta tesis.

Contents

List of Figures

List of Tables

Abstract

Quantum computers provide efficiency advantages over their classical counterparts, for various computational problems. One such problem is "mining" cryptocurrencies. Mining is the process, in Proof-of-Work (PoW) cryptocurrencies such as Bitcoin, by which individuals are rewarded for solving computationally hard problems. It has been stated that quantum miners could have a computational advantage over classical ones. In this work, we develop an explicit quantum algorithm for performing proof-of-work.

Keywords: *Quantum Computing, cryptocurrency, Bitcoin mining, cryptographic hash function, Grover's algorithm*

Introduction

Quantum computing is a scientific and technological paradigm that seeks to take advantage of the physical phenomena that manifest at scales where the laws of quantum mechanics describe nature, in order to process information.

The concept was first attributed to the scientist and 1965 Nobel Prize winner, Richard Feynman, who, in the year 1981, spoke at a conference on the subject "Simulating physics with computers". During his participation, he talked about the idea of simulating nature through the use of quantum computers instead of the classical ones which, as it was known since then, exhibit difficulties when simulating quantum systems. From that moment on, research in this field has been growing, and the results that prevail bring us closer and closer to a new era in which quantum computing permeates almost every aspect of life.

There are two main approaches to quantum computing. The gate-based model that operates over circuits — the one this dissertation is based on—, and the analog approach based on the adiabatic theorem, which states that if small changes occur slowly to a quantum state initialized at the ground state of a system it is highly likely that the system will remain in a ground state.

Unlike traditional computational algorithms, quantum algorithms implement the use of non-classical gates that operate on the fundamental quantum units of information denominated *qubits* (quantum bits). By exploiting physical properties such as superposition and entanglement, it has been proved that, for some problems, quantum algorithms can be significantly faster than their classical analogs [1].

In 1994, Shor's algorithm became one of the earliest examples of a quantum algorithm that could represent real-world consequences by posing a threat to long-established RSA algorithm.

The RSA algorithm is an encryption technique that uses a public-key to encrypt messages and a private-key to decrypt them. It is widely used to protect data and transfer information securely because it relies on the difficulty of factoring a large integer. With a sufficiently large quantum computer, Shor's algorithm could break RSA more efficiently than with the classical method [2].

Another notable example of a quantum algorithm that has an advantage over its classical equivalent is Grover's algorithm. The algorithm was developed in 1996 and it represents the solution to the unstructured search problem. Grover's algorithm, known also as the *quantum search algorithm*, provides a quadratic speed-up.

It is for the above reasons that the study and design of quantum algorithms has grown significantly over the last few decades.

Since the realization that quantum computers will be able to solve problems for which no efficient classical algorithm is known so far, there have been some concerns about the consequences

of such a power of performance [3]. We can choose, however, to focus on the improvements that quantum technology can provide over well-established classical algorithms. This is what we call a "quantum advantage".

The motivation for this work is aligned with this idea.

Quantum computing has serious potential for applications in diverse disciplines [4], [5]. One particular area of interest is the potential for quantum advantage in the "mining" cryptocurrencies such as Bitcoin [6]. Bitcoin is a cryptocurrency based on Blockchain technology, which emerged in the year 2008, when Satoshi Nakamoto released a paper entitled "Bitcoin: A Peer-to-Peer Electronic Cash System".

Bitcoin mining is based on a consensus protocol that requires miners to solve a computationally hard problem. Quantum computing may, in this sense, have a significant role to play in Bitcoin mining, as the speed at which quantum computers could solve this problem could improve the efficiency of the process.

Chapter 1

Blockchain Fundamentals

In this chapter, we will summarize the importance of Blockchain technology and describe the basic concepts on which it is built. To do so, we will introduce Bitcoin, arguably the most popular application of Blockchain technology.

In section 1.2, we will elaborate on the key concepts that make up the structure of a Blockchain.

And, in section 1.3, we will address the fundamentals of cryptographic hash functions, a powerful tool of modern cryptography with an essential role in Blockchain's information storage.

1.1. Blockchain & Bitcoin

Although used interchangeably, Blockchain is just one particular example of Distributed Ledger Technology (DLT).

A DLT is a decentralized database managed by various members. This way, the information it contains is shared between several participants, which makes it less prone to hacking attacks.

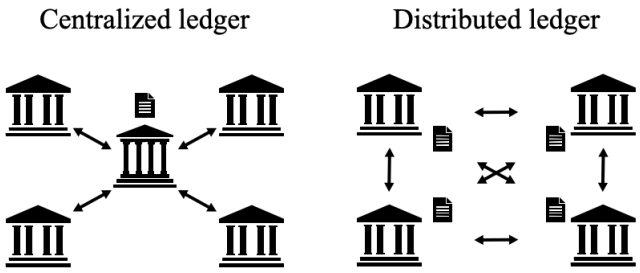

Figure 1.1: Unlike a centralized ledger, a distributed ledger makes the information available to all users.

In a DLT, any member of the network can contribute to the ledger by means of consensus protocols that often involve the use of cryptography. Once the information is shared, it becomes part of an immutable ledger that keeps a record of all the contributions made by the other

participants.

Blockchain is a DLT that organizes information into a series of blocks, which are validated by a network. Any participant of the network can enter a transaction to the ledger and all the transactions that occur in a given time period are added to the same block, which is linked to the previous one through the use of a cryptographic hash function. In section 1.3 we will explain in more detail the workings of a cryptographic hash function.

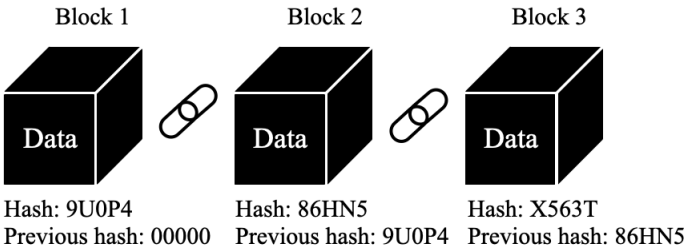

Figure 1.2: Basic structure of Blockchain.

Blockchain technology has a wide range of applications [7]. However, the most prominent has to do with the management of digital currencies. Indeed, Bitcoin emerged as a financial application of Blockchain technology and, as such, became the first peer-to-peer payment network.

The creation of Bitcoin was primarily motivated by the double-spending problem.

The double-spending problem describes a characteristic that is inherent to digital money. Unlike physical money, digital money can be in two places at once since a data file can be duplicated and thus spent simultaneously in several transactions.

One solution to this problem is to relay on trusted third parties that can prevent fraudulent activities. However, these financial institutions do not solve the problem completely and also generate costs. In addition, they usually limit the number and type of transactions a user can perform.

The idea of eliminating the need to rely on a bank for user-to-user transactions is, in fact, the core concept behind Bitcoin.

Bitcoin's solution to the double-spending problem is based on cryptographic proof rather than trust, and involves using a decentralized time-stamped ledger, which can be accessed by any member of a peer-to-peer network. The security of the information inserted into the ledger can then be maintained if the majority of users controlling the network are honest.

In the case of Bitcoin, the data stored in the ledger contains information about financial transactions. A Bitcoin transaction can be briefly described as follows. Suppose Alice wants to send a transaction to Bob. Then, she will announce a new transaction and the information within will be broadcast to all nodes in the network. Later, each node will collect this and other new transactions and propose a new block through a process called *mining*.

During mining, a node —miner— will select a set of broadcast transactions and verify that they are valid. Then, they will select the hash value of the most recent block header and perform a consensus mechanism known as *proof-of-work*. Once the miner has encountered a solution to a given computational problem, they will issue a new candidate block. If the candidate block is

verified, the miner will receive a reward and the block will be added to the chain. Only at this point, the transaction from Alice to Bob will be completed.

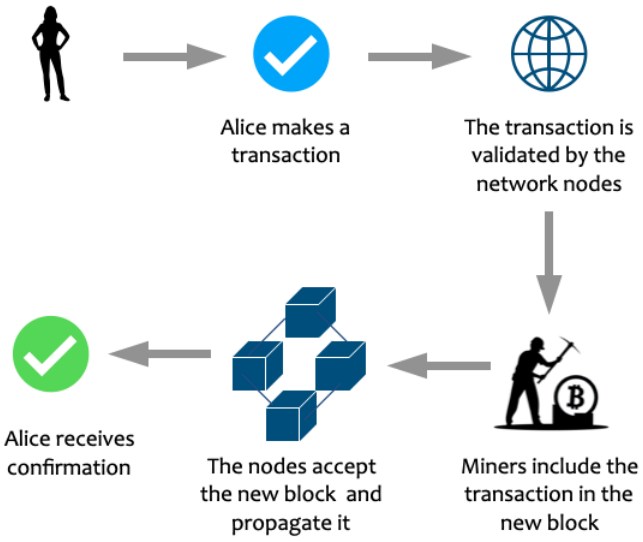

Figure 1.3: A Bitcoin transaction.

1.2. The structure of a Blockchain

Blockchain technology was formally introduced in 2008 when Satoshi Nakamoto published the paper "Bitcoin: A Peer-to-Peer Electronic Cash System" [8]. However, the foundations on which it was developed go back decades. Notably, Merkle Tree technology, developed in the late 1970s, was later employed in the early 1990s as part of a proposal for the time-stamping of digital documents.

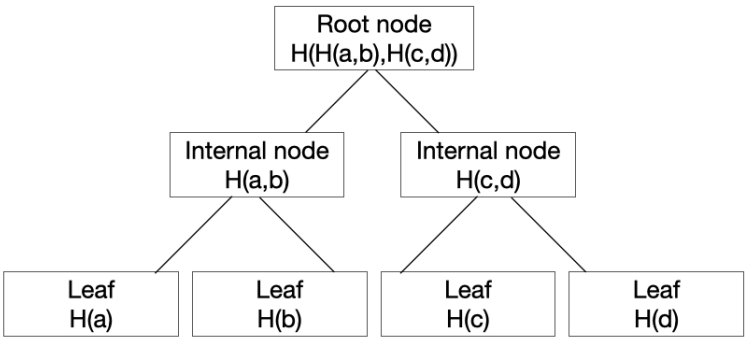

Figure 1.4: Merkle Tree structure with H as an underlying cryptographic function.

It was in 1979 when Ralph C. Merkle patented a concept that allows easy verification of any type of data being transferred between users. As its name suggests, the structure of a *Merkle tree* is similar to that of a tree in that it has leaves and nodes. Each leaf contains a data fragment

that has been hashed using an underlying cryptographic hash function H, and each internal node contains the hashed value of two leaves. The process can be iterated until what is called the *root node* is obtained.

The integrity of the information relies on the fact that H is effectively a one-way function. As a result, the authenticity of a specific data fragment can be verified through a *Merkle proof*.

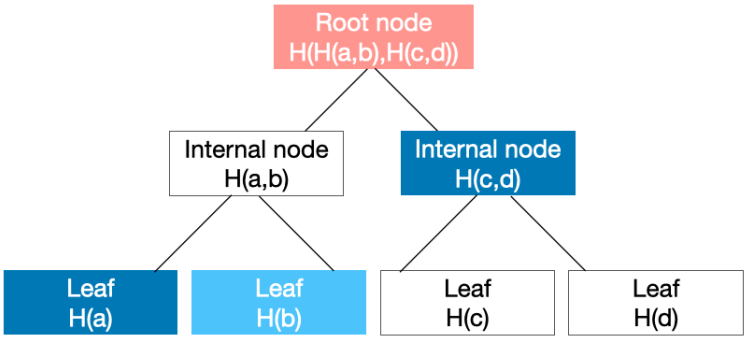

Figure 1.5: A Merkle proof is a specific subset of all the hashes in a Merkle tree that allows the root node of the tree to be recomputed if a particular piece of data is part of the tree.

Blockchain structure inherits this feature. Each block in the chain has its own header: a section of information that identifies it as unique and contains, among other data, the root node of all transactions in the block.

The block header also contains a nonce, used for the Proof of Work algorithm, and a timestamp that indicates when the block was created.

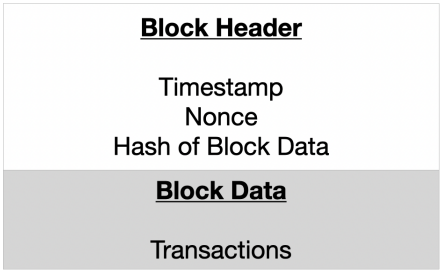

Figure 1.6: The structure of a block.

For more details, the concept of a time-stamping system that prevents the modification of digital documents and forgery of timestamps was introduced in the 1991 article by Stuart Haber and W. Scott Stornetta, titled “How to Time-Stamp a Digital Document" [9].

1.3. Cryptographic Hash Functions

Hash functions are one of the most important elements in modern cryptography due to the role they play in various digital security protocols, such as those destined for password confirmation, information integrity checking, and digital signature generation and verification.

In Blockchain technology, cryptographic hash functions are used to hash all the blocks in the chain.

The main objectives of cryptographic hash functions are to maintain privacy in a network, to ensure the integrity of the messages exchanged between different users, and to have a system that allows authenticating the digital identity of each of them so that the transactions that may occur between participants are valid and reliable.

Due to the different nature of the information, the idea is to implement a function capable of encoding messages of arbitrary length and producing outputs of a fixed size to suit the needs of a given application.

Thus, since the 1970s, people have designed this type of function, better known as a cryptographic hash function.

1.3.1. Definition and Properties

Hash Functions

A function is a relation that associates each element of a set, called the *domain*, with an element in another set, known as the *codomain*. Simply put, a function can be thought of as a machine that given an input element, produces an output element. In some cases, the domain and codomain may be the same set. If the function returns each input as its own output, it is referred to as the identity function.

It is important to note that, by definition, a function cannot assign two or more elements of the codomain to a single element of the domain.

Thus, a hash function is defined as a function H that, given an input message or string x of arbitrary size, produces an output y of fixed size, called *hash value* or *digest*.

$$H : \{0,1\}^* \rightarrow \{0,1\}^k$$

Where k is a natural number.

Hash functions can be used to organize efficiently a database. Instead of performing a linear search over a large number of data, hashing an element of a list allows to reduce the complexity of the algorithm in the average case.

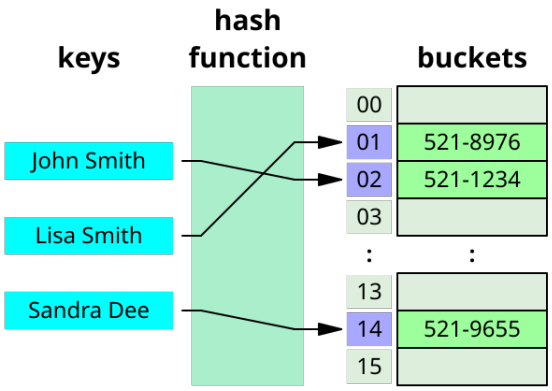

Figure 1.7: A hash table is a data structure that associates each of the keys in a list with an index value generated by a hash function.

However, hash functions are mostly used for cryptographic purposes, as they allow to easily check whether a message has passed through a noisy channel or has been compromised.

Cryptographic Hash Functions

In addition to this definition, a secure cryptographic hash function must have at least the following three properties: preimage resistance, second preimage resistance, and collision resistance.

1. **Preimage Resistance**
   This security property consists of the following: given a string y that belongs to the codomain of a cryptographic hash function H, it must be computationally inefficient to try to find some element x in the domain, such that $H(x) = y$.

   What this property tells us is that, ideally, we need a function that given x, a string as input, its hash value is easy to compute, i.e., computationally efficient. And at the same time, we require the reverse procedure, i.e., given a hash value, finding its preimage, to be a computationally difficult task to implement.

   a) **Computational Complexity**
   What do we mean by the term “computationally inefficient"?

   In computer science, this concept describes the amount of computational time it takes to execute an algorithm. The time period is approximated by counting the number of elementary operations contained in the algorithm in the worst case, and for this purpose a certain amount of time is assigned to each elementary operation.

   Thus, an algorithm is usually classified as computationally inefficient if its time complexity cannot be described by a polynomial term.

   b) **One-way Functions**
   It is worth talking about unidirectional functions, which are characterized by being easy to compute but difficult to invert. Again, here easy or difficult are terms that have to do with the computational complexity of the problems.

The existence of this type of function is an open conjecture, which means that it is not yet a proven fact that these functions exist, however, neither has proof been found to disprove their existence.

Among the candidate functions to belong to this class is the multiplication between prime numbers. According to the fundamental theorem of arithmetic, any positive integer greater than one is a prime number or a unique product of prime numbers. As we know, it is relatively simple to carry out the multiplication algorithm.

It is tedious but feasible to calculate the usual product between two prime numbers with several significant figures, however, performing the reverse process, i.e. finding the prime factors of a very large number in a reasonable time is difficult even for today's computers because there is so far no efficient algorithm capable of giving us the result without avoiding trying all possible combinations.

2. **Second Preimage Resistance**
Another important property necessary for a hash function to be secure is that given a string x it must be very difficult to find an $x' \neq x$ such that $H(x) = H(x'))$.

This is not trivial since the hash function accepts in principle any string but has a finite domain of cardinality 2^n, where n is an integer that is determined when constructing the function. Therefore, the function cannot be injective. This fact illustrates Dirichlet's pigeonhole principle.

Imagine that there is a number a of pigeons and a number b of pigeon boxes, the fact that $a > b$ implies that if the flock of pigeons flies into it, there will be at least one pigeon box with more than one pigeon in it.

In short, the codomain of a hash function is finite: the number of elements it possesses is 2^n, so it is to be expected that at least two strings have the same hash value. What we say then is that we require the hash function to have resistance to the second preimage.

3. **Collision Resistance**
The third property we will discuss also has to do with two elements associated with the same hash value, these elements are called collisions.

A collision-resistant hash function must ensure that it is difficult to try to find collisions, i.e., to find x' and x', with x' such that $H(x) = H(x')$. This is related to the birthday paradox.

Suppose we want to know how many people we have to gather so that the probability of finding two of them with the same birthday is at least 50%. Will twenty-three people be enough? Let's see. In a room with twenty-three people, the probability of encountering at least one collision is:

$$P(collision) = 1 - P(no\,collision) \tag{1.1}$$

We can treat the non-collision probability as an extraction problem without replacement.

The probability that one person has a birthday on one of the 365 days of the year is equal to 1, then, the probability that the second person has a different birthday is equal to 364/365. Continuing this way, we can take into account up to twenty-three people and observe that,

$$\begin{aligned} P(collision) &= 1 - \left(1 - \frac{1}{365}\right)\left(1 - \frac{2-1}{365}\right)...\left(1 - \frac{23-1}{365}\right) \\ &= 0.507 \\ &\approx 50\% \end{aligned} \tag{1.2}$$

The problem is called a paradox simply because the result is not very intuitive: it is generally expected that many more people are needed to obtain the desired result.

4. **Additional Properties**
   There are additional properties worth mentioning.
   - The uncorrelation property ensures that each bit in the input string affects the computed hash value noticeably, i.e., even if we had two messages that differed by only one bit, their hash values would be practically different.
   - The near-collision resistance property ensures that it is difficult to find two strings that are similar to some degree.
   - And in addition, partial preimage resistance indicates that it should be difficult, given a hash value, to recover any part of the message.

1.3.2. Merkle-Dåmgard's Construction

Hash functions can be broadly classified into two categories: dedicated functions and block cipher-based functions.

Dedicated functions are functions that are built from scratch exclusively to compute hash values. They are algorithms that do not rely on other cryptographic elements already in existence for other purposes. Unlike these, those belonging to the second category are based on existing elements, such as block ciphers.

The Merkle-Dåmgard structure is one of the most commonly used structures for constructing dedicated hash functions.

Hash functions process messages of arbitrary size and return fixed-length strings. Therefore, it is necessary for the function to have a system that expands or abbreviates messages that do not have the function's output length.

Dåmgard's construction is based on a segmentation process and the iterative execution of a compression function. For H to be a hash function following the Merkle-Dåmgard structure, it must have the following two elements:

1. A padding scheme that takes the message m and extends its length to a multiple of the block size.

2. A compression function that processes each of the blocks taking into account an initial value *(IV)*.

The iteration process takes a block and the hash value of the immediately preceding block, so that at each step of the iteration the information from the previous block is taken into account for the hash value of the next block.

The padding scheme can also include the length of the message in the last block. This feature is known as the Merkle-Dåmgard strengthening.

Another important contribution is the Merkle-Dåmgard's Theorem which states that any compression function that is collision-resistant can be used to construct a collision-resistant hash function.

1.3.3. Hash Function, an example: SHA-1

To better understand how hash functions work we will develop an example using the structure of the SHA-1 function.

The SHA-1 function belongs to a family of cryptographic functions called Secure Hash Algorithm. These functions were designed by the National Security Agency of the United States with the purpose of having a standard among the functions used in the federal security protocols. Currently, there are four functions within this family: SHA-0, SHA-1, SHA-2, and SHA-3.

The SHA-1 function was discontinued at the beginning of the previous decade due to the fact that from 2005 onwards attacks requiring a number of evaluations lower than 280 were reported. Since then, the U.S. government recommends using SHA-2 or SHA-3 functions instead of SHA-1.

The SHA-1 function produces values of 160 bits in length. Suppose we want to obtain the hash value of the message '*abc*'.

In binary code, each letter consists of 8 bits, so the message we want to hash looks like this:

01100001 01100010 01100011

The first step is to initialize five random variables, also called buffers. The values used in the case of the SHA-1 function are:

$H0 = 0x67452301$
$H1 = 0xEFCDAB89$
$H2 = 0x98BADCFE$
$H3 = 0x10325476$
$H4 = 0xC3D2E1F0$

Let us proceed to the bit-padding scheme. The message has 24 bits, we will add a 1 and enough zeros to have a string of 448 bits. The last 64 bits will be used to indicate the length of the message which in this case is the number 24 in binary code. Thus, we will get a block of 512 bits, which is the block size of the function.

The next step is to segment each block into sixteen strings or words of 32 bits each, $W(i)$, $0 \leq i \leq 15$. In this instance, since the message is short, we will only have one block.

It follows to expand the sixteen words until we have eighty words of 32 bits each, $W(i)$, $0 \leq i \leq 79$. For this purpose, we need to apply certain operations to the words obtained in each of the iterations of a cycle that begins at word $W(0)$ and concludes at word $W(79)$.

In the iterations $16 \leq i \leq 79$, the operations applied are:

$$W(i) = S^1(W(i-3) \oplus W(i-8) \oplus W(i-14) \oplus W(i-16)) \tag{1.3}$$

Where the symbol $\oplus$ denotes the binary logic operation XOR:

a	b	a $\oplus$ b
0	0	0
0	1	1
1	0	1
1	1	0

Table 1.1: XOR truth table.

And where the symbol S^n denotes the *left shift* of n bits to the left, i.e., the S^1 operator will shift one bit, the most significant bit (at the left end) to the position of the least significant bit:

$$\underline{1}0010101101110 \rightarrow 00101011011101\underline{1}$$

For example, let's see how to obtain the seventeenth word.

$$W(16) = S^1(W(16-3) \oplus W(16-8) \oplus W(16-14) \oplus W(16-16)) \tag{1.4}$$

$$\rightarrow W(16) = S^1(W(13) \oplus W(8) \oplus W(2) \oplus W(0)) \tag{1.5}$$

W(13)	00000000 00000000 00000000 00000000
W(8)	00000000 00000000 00000000 00000000
W(2)	00000000 00000000 00000000 00000000
W(0)	01100001 01100010 01100011 10000000
$\oplus$	
W(16)	01100001 01100010 01100011 10000000

Table 1.2: We obtain the 17th word by XORing the words W(0), W(2), W(8) and W(13).

$$W(16) = S^1(01100001011000100110001110000000) \tag{1.6}$$

$$\rightarrow W(16) = 11000010110001001100011100000000 \tag{1.7}$$

SHA-1 algorithm requires also eighty constant words:

$K_i = 0x5A827999$ for $\quad 0 \leq i \leq 19$
$K_i = 0x6ED9EBA1$ for $\quad 20 \leq i \leq 39$
$K_i = 0x8F1BBCDC$ for $\quad 40 \leq i \leq 59$
$K_i = 0xCA62C1D6$ for $\quad 60 \leq i \leq 79$

For the main cycle, we need to take those eighty words and at each iteration calculate the $TEMP$ variable as shown below:

$$TEMP = S^5(A) \oplus f(i; B, C, D) \oplus E \oplus W(i) \oplus K(i) \tag{1.8}$$

Where the variables A, B, C, D, and E initially store the values $H0, H1, H2, H3$, and $H4$ respectively.

And where:

$f(i; B, C, D) = (B \wedge C) \vee ((\neg B) \wedge D) \quad for \quad 0 \leq i \leq 19$
$f(i; B, C, D) = B \oplus C \oplus D \quad for \quad 20 \leq i \leq 39$
$f(i; B, C, D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) \quad for \quad 40 \leq i \leq 59$
$f(i; B, C, D) = B \oplus C \oplus D \quad for \quad 60 \leq i \leq 79$

For each iteration the variables A, B, C, D, and E are reassigned the variables:

$E = D$
$D = C$
$C = S^{30}(B)$
$B = A$
$A = TEMP$

Finally, after the last iteration, the variables $H0, H1, H2, H3$, and $H4$ get updated:

$H_0 = H_0 \oplus A$
$H_1 = H_1 \oplus B$
$H_2 = H_2 \oplus C$
$H_3 = H_3 \oplus D$
$H_4 = H_4 \oplus E$

And the digest results from appending the values of each of the previous variables:

$digest = H_0 \parallel H_1 \parallel H_2 \parallel H_3 \parallel H_4 = A9993E364706816ABA3E25717850C26C9CD0D89D$

1.3.4. Hash Function Attacks

Attacking a hash function means violating one or more of its security properties. We might think that a cryptographic component such as a hash function can be broken by means of some decryption method, however, while it is true that a hash function hides messages, it is not by means of encryption that it does so. The main difference between encryption and hashing is that the former is reversible through a key scheme (either symmetric or asymmetric), while the latter should not be, in principle.

Attacks on hash functions fall mainly into two categories: generic attacks and cryptanalytic attacks.

Generic or brute-force attacks are attacks that can be applied to any type of hash function that has the Merkle-Dåmgard's construction as its basis. Thus, this class of attacks does not focus on any specific internal structure of the hash function. Examples of such attacks are preimage attacks, second preimage attacks and collision attacks.

1. **Preimage attack**
   In this attack, we consider an adversary that, given a hash value of length n, will try to find a string corresponding to that value under the hash function, evaluating as many messages as he can until he finds it. The effort required to perform this kind of attack is 2^n.

2. **Second preimage attack**
   In this attack, we consider an adversary that, given a message m and its hash value, will try to find a string corresponding to the same value under the hash function, evaluating as many messages $m' \neq m$ as he can until he finds it. The effort required to perform this type of attack is 2^n.

3. **Collision attack**
   In this attack, an adversary will try to find to find two messages that produce the same hash value, i.e. a hash collision.

On the other hand, cryptanalytic attacks directly involve the compression function, this as a consequence of the Merkle-Dåmgard's theorem that establishes a relationship between the properties of the compression function and the hash function that is constructed from it.

Chapter 2

Quantum Computing Fundamentals

In this chapter we will introduce the elementary concepts of quantum computing. We will start by talking about how to encode information in qubits and the operations we can perform with them. Next, we will discuss the most significant algorithms to date. And finally, we will mention some of the most widely used physical implementations.

2.1. Qubits

Just as the bit is the fundamental unit of information in classical computing, in quantum computing there is an analogous concept called the quantum bit or *qubit*.

Mathematically, a qubit can be taken as an element of a two-dimensional complex vector space that can be in infinite states. Two possible states are $|0\rangle$ and $|1\rangle$ as it is in the classical case. The rest of the states can be expressed as a linear combination of these two. Most commonly, the notation used to represent the states of a qubit is the Dirac notation.

We choose to use the computational basis, which is, of course, an orthonormal set.

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.1}$$

Unlike the classical bit, we cannot probe into the qubit to determine its state. All we can say is that when we measure the qubit we will get the states $|0\rangle$ or $|1\rangle$ with probabilities $|\alpha|^2$ and $|\beta|^2$ respectively. This feature illustrates one of the postulates of quantum mechanics called the *quantum collapse*.

Although one could think that it is difficult to predict the behavior of quantum systems, quantum mechanics allows the manipulation of the state of a qubit in such a way that the measured result depends on the properties of the state. Ultimately this is the hallmark of quantum computing.

As mentioned above, a qubit can exist in a continuum of states between $|0\rangle$ and $|1\rangle$ until it is observed. Since $|\alpha|^2$ and $|\beta|^2$ represent the measurement probabilities, their sum must equal 1.

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.2}$$

And furthermore, we can effectively write:

$$|\Psi\rangle = \cos\frac{\theta}{2} + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \tag{2.3}$$

Where θ and ϕ are real numbers.

Graphically, the state of a qubit can be visualized as a point on the three-dimensional unit sphere (Figure 2.1) called the Bloch sphere. This representation is very useful, especially when visualizing the operations that can be applied to a qubit.

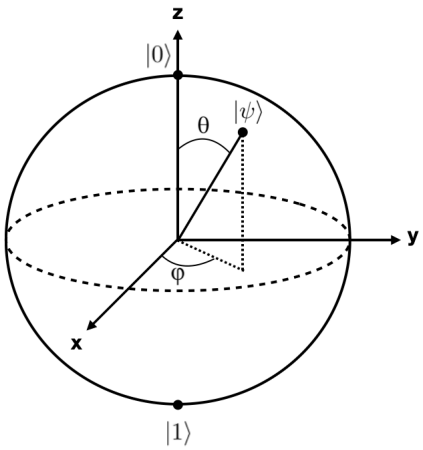

Figure 2.1: Bloch sphere: a graphic representation of a qubit.

2.2. Single Qubit Gates

A qubit can also be represented as a two-row column vector.

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix}$$

Similarly to classical computation, in quantum computing, it is possible to define gates. Single qubit gates are represented by 2x2 unitary matrices and their effect on a state can be visualized as a rotation and/or reflection in the Bloch sphere. The action of a gate on a qubit is represented by the usual matrix multiplication.

Consider the quantum NOT gate, whose output will be:

$$X\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = X\begin{bmatrix}\beta\\ \alpha\end{bmatrix} \tag{2.4}$$

We can represent the quantum NOT gate as follows.

$$X \equiv \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$$

Thus, the quantum NOT gate can be thought of as a rotation about the x-axis by π radians.

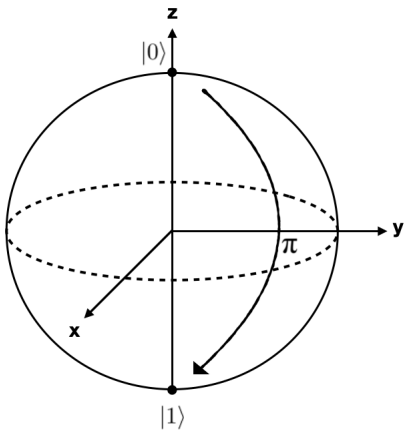

Figure 2.2: The NOT gate produces a rotation of π radians about the x-axis.

Two important single qubit gates are also the Z gate, and the Hadamard gate, which turns state $|0\rangle$ into state $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and state $|1\rangle$ into state $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ thus creating a superposition state.

$$Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

2.3. Multi-Qubit Gates

In classical computation there exists a set of logic gates that perform the fundamental binary operations from which any digital circuit is composed. Most of these logic gates act on a pair of bits and output a single bit of information.

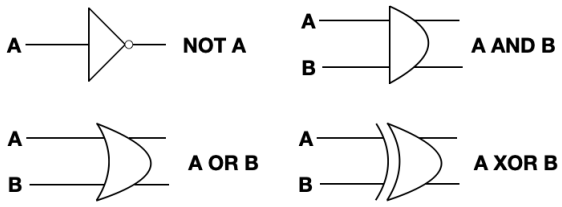

Figure 2.3: Aside from the NOT gate, classical two-bit gates.

Although not entirely analogous to classical computing, quantum computing allows the action of a single-qubit gate to be extended to a multi-qubit system. Additionally, by including other mechanisms like controlled operations, we can also define multi-qubit gates, all while considering fundamental limitations like the no-cloning theorem.

It is important to note that all quantum gates are represented by unitary operators and are therefore reversible, in contrast to classical logic gates, which are generally irreversible.

Let's begin with the simplest case of a two-qubit system. As discussed above, describing the state of a qubit requires two complex numbers. In the same manner, describing the state of two qubits requires four complex amplitudes.

$$|\Psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \tag{2.5}$$

Same as before, the probabilities of measuring states $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$ are,

$$P(|00\rangle) = |\alpha_{00}|^2, \quad P(|01\rangle) = |\alpha_{01}|^2, \quad P(|10\rangle) = |\alpha_{10}|^2, \quad P(|11\rangle) = |\alpha_{11}|^2 \tag{2.6}$$

With the normalization condition as follows,

$$\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2} = 1 \tag{2.7}$$

Algebraically, we can represent the two-qubit state using the tensor product,

$$|\psi_0\rangle = \begin{bmatrix}\alpha_0\\ \alpha_1\end{bmatrix}, \quad |\psi_1\rangle = \begin{bmatrix}\beta_0\\ \beta_1\end{bmatrix}$$

$$|\psi_1\psi_0\rangle = |\psi_1\rangle \otimes |\psi_1\rangle = \begin{bmatrix}\beta_0\alpha_0\\ \beta_0\alpha_1\\ \beta_1\alpha_0\\ \beta_1\alpha_1\end{bmatrix} \tag{2.8}$$

However, not all two-qubit states can be written as a tensor product of individual qubit states. Entangled states, such as the Bell states, are examples where the full state of the system cannot be expressed as the tensor product of single-qubit states.

Similar as before, we can represent simultaneous gates,

$$X|\psi_1\rangle \otimes Z|\psi_0\rangle = (X \otimes Z)|\psi_1\psi_0\rangle \tag{2.9}$$

$$X \otimes Z = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \otimes \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} = \begin{bmatrix}0 & 0 & 1 & 0\\ 0 & 0 & 0 & -1\\ 1 & 0 & 0 & 0\\ 0 & -1 & 0 & 0\end{bmatrix}$$

The generalization to n qubits is straightforward. The basis states of the system will be of the form

$$|\Psi\rangle = |x_1 x_2 ... x_n\rangle \tag{2.10}$$

and the collective quantum state will require 2^n complex amplitudes.

One of the most important multi-qubit gates is the CNOT gate. The CNOT gate is a 2-qubit quantum gate that performs a NOT gate in a *target* qubit depending on the state of a *control* qubit.

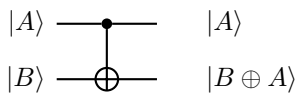

Figure 2.4: Circuit representation of a CNOT gate applied to target qubit B.

$$|00\rangle \rightarrow |00\rangle, |01\rangle \rightarrow |01\rangle, |10\rangle \rightarrow |11\rangle, |11\rangle \rightarrow |10\rangle$$

It can be proved [10] that any multiple-qubit gate can be built from CNOT and single-qubit gates. This concept is known as the universality of the CNOT gate.

2.4. Circuits

We may represent the processing of information, as in classical computing, with circuits.

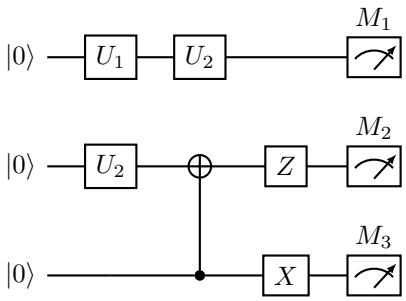

Figure 2.5: A simple quantum circuit

Circuits are read from left to right. The lines represent wires carrying the information through time and the gates represent the operations performed to manipulate the quantum information.

One important operation to note is the measurement, labeled by an M symbol. The measurement operation describes the quantum collapse of a quantum state into a classical state. Up until now we have approached things from the computational basis but let's remember that there are other possible choices for the basis.

Another key feature of quantum computing is entanglement. Below we show a circuit whose output states are known as the Bell states.

In a two-qubit system, we begin by applying a Hadamard gate to the first qubit, followed by a CNOT gate. What makes this system special is that the measurement outcomes (Fig. 2.4) in the

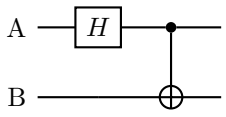

Figure 2.6: Quantum circuit that creates Bell states.

computational basis are perfectly correlated, which allows protocols such as quantum teleportation.

$\|\Phi^+\rangle$	$\frac{\|00\rangle+\|11\rangle}{\sqrt{2}}$
$\|\Phi^-\rangle$	$\frac{\|00\rangle-\|11\rangle}{\sqrt{2}}$
$\|\Psi^+\rangle$	$\frac{\|01\rangle+\|10\rangle}{\sqrt{2}}$
$\|\Psi^-\rangle$	$\frac{\|01\rangle-\|10\rangle}{\sqrt{2}}$

Table 2.1: Bell states.

2.5. Quantum Algorithms

Quantum algorithms differ greatly from conventional algorithms since there is no analogue of quantum superposition or entanglement in the classical scenario. Moreover, some quantum algorithms exhibit the potential to solve certain problems in a more feasible way as opposed to classical algorithms. In many cases, even if a problem has clear solution classically, the resources to obtain it scale exponentially, rendering the solution not viable.

There exist two main classes of algorithms that under the right settings provide an advantage over classical algorithms: the algorithms that are based on the Quantum Fourier Transform, and the ones based on Grover's algorithm. In this section we're going to discuss briefly the basics of both.

2.5.1. Quantum Fourier Transform

It's no surprise that Fourier transforms play a significant role in the quantum algorithmic landscape. After all, quantum states are described by complex amplitudes and phases, which encode information about the state and can be manipulated to perform specific operations. By applying a linear transformation like the one below we can obtain the discrete Fourier transform of a computational basis state vector $|\alpha_j\rangle$,

$$|\alpha_j\rangle = \frac{1}{2^n}\sum_{k=0}^{2^n-1} e^{\frac{2\pi i j k}{2^n}} |\alpha_k\rangle \tag{2.11}$$

where $0 \leq j \leq 2^n - 1$.

The previous result can be generalized for a superposition of basis state vectors $|\beta\rangle$,

$$|\beta\rangle = \sum_{j=0}^{2^n-1} a_j |\alpha_j\rangle = \frac{1}{2^n}\sum_{k=0}^{2^n-1}\left[\sum_{j=0}^{2^n-1} e^{\frac{2\pi i j k}{2^n}} a_j\right]|\alpha_k\rangle = \sum_{k=0}^{2^n-1} b_k |\alpha_k\rangle \tag{2.12}$$

This transformation can be verified as unitary and implemented using a quantum circuit more efficiently than classical algorithms, such as the Fast Fourier Transform. Although the measurement outcome of the transformation alone may not yield a directly valuable result, it plays a crucial role in phase estimation—a procedure that, when combined with other routines, allows to find solutions to problems such as the factoring problem and the order-finding problem.

For a 2^n-dimensional basis, equation (2.12) can be re-expressed using the binary basis $\{|00...00\rangle$, $|00...01\rangle$, $|00...10\rangle$, ..., $|11...11\rangle\} = \{|j_1...j_n\rangle\}$, where each basis element corresponds to a tensor product of n qubits,

$$|j_1...j_n\rangle = \frac{1}{\sqrt{2^n}}(|0\rangle + e^{(2\pi i)(\frac{j_n}{2})}|1\rangle) \otimes (|0\rangle + e^{(2\pi i)(\frac{j_n}{2}+\frac{j_{n-1}}{4})}|1\rangle) \otimes ... \otimes (|0\rangle + e^{(2\pi i)(\frac{j_n}{2}+\frac{j_{n-1}}{4}+...+\frac{j_1}{2^n})}|1\rangle) \tag{2.13}$$

The circuit that realizes this representation consists of applying in a controlled manner (ie. having target qubits) the following transformation

$$R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^k}} \end{bmatrix} \tag{2.14}$$

to each of the n qubits that can be either initialized in $|0\rangle$ or $|1\rangle$, depending on the state to transform.

For a 3-qubit system the circuit implementation of the algorithm looks like,

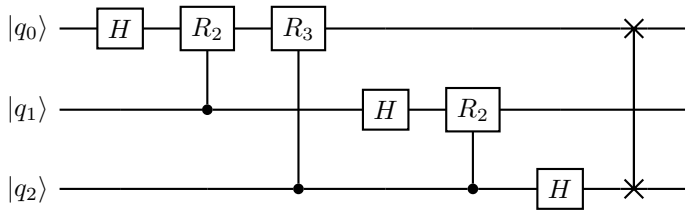

Figure 2.7: QFT circuit for a 3-qubit state, with final SWAP gates to correct qubit order.

2.5.2. Grover's Algorithm

Search algorithms provide a solution to one of the most fundamental problems in computer science. In an unstructured search problem, there is a (possibly disordered) set $X = x_0, x_1, ..., x_{N-1}$ containing N elements. The objective, given a function $f : X \to 0, 1$, is to find the marked element(s) x' such that $f(x') = 1$. Traditionally, the problem requires N/M evaluations of the function f, but there is a quantum algorithm known as Grover's algorithm that solves the same problem in $O(\sqrt{N/M})$ queries [11].

For convenience we assume that $N = 2^n$, requiring n qubits to cover the entire search space, where the data can be labeled as an n-bit string in $\{0, 1\}^n$. Our search problem can have M solutions: $1 \leq M \leq N$.

Grover's algorithm consists of three stages prior to measurement: the creation of a superposition of states, the application of the oracle operator, and the use of the amplitude amplification operator.

1. Superposition of states

Grover's algorithm starts by initializing all n qubits to the $|0\rangle$ state and then applying Hadamard gates to create a uniform superposition over the entire search space. This process makes use of a feature called quantum parallelism, which allows the simultaneous manipulation of all the possible inputs.

At this point the state can be expressed in the following way,

$$|\psi\rangle = \sum_{x\in\{0,1\}^n} \frac{1}{\sqrt{N}} |x\rangle \tag{2.15}$$

And the circuit implementation can be visualized as,

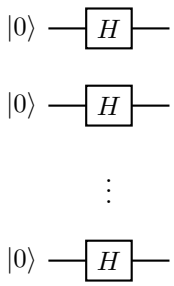

Figure 2.8: Circuit implementation of a $H^{\otimes n}$ gate.

Measuring the system at this stage would produce any basis state with equal probability since all amplitudes in the superposition are identical [1],

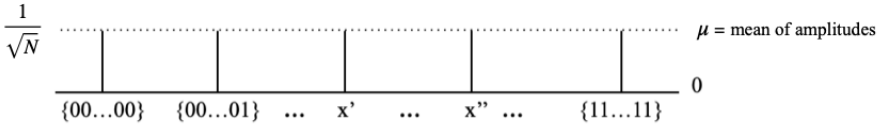

Figure 2.9: Uniform superposition of states.

2. The oracle

As mentioned previously, the goal is to implement a reversible operation that recognizes the elements for which the state represents a solution to the problem. Mathematically framed, this is achieved through a function $f : X \to 0, 1$ for which $f(x) = 1$ if and only if x represents a solution, and zero otherwise.

[1] Diagram by the author, closely following a diagram from [13]

In order to implement this action in a quantum circuit we need a unitary operator capable of identifying the correct states. This operator is known as the oracle. Its essential function can be represented as,

$$|x\rangle |a\rangle \xrightarrow{\text{oracle}} |x\rangle |a \oplus f(x)\rangle \tag{2.16}$$

For this formulation, in the computational basis, the oracle qubit $|a\rangle$ flips its state only when x corresponds to a valid solution of the problem. To illustrate this mechanism let's analyze the case where $|a\rangle = |0\rangle$ and x represents a solution,

$$|x\rangle |0\rangle \xrightarrow{\text{oracle}} |x\rangle |0 \oplus f(x) = 1\rangle = |x\rangle |1\rangle \tag{2.17}$$

In the same manner the other three cases can be analyzed.

For our search algorithm though, it is convenient to initialize the oracle qubit in the state,

$$|a\rangle = \frac{1}{\sqrt{2}} |0\rangle - |1\rangle \tag{2.18}$$

This way, we produce the so-called *phase kickback*, which corresponds to the conditional flipping of the phase of the state (ie. only when x is a solution) effectively marking the solutions to the search problem

$$|x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \xrightarrow{\text{oracle}} (-1)^{f(x)} |x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{2.19}$$

and allowing us to disregard the oracle qubit in the subsequent description of the system since its effect is now encoded as a phase on the search space qubits.

At this point the state can be expressed as a superposition of marked states $|x'\rangle$ and unmarked states,

$$O |\psi\rangle = -\frac{1}{\sqrt{N}} \sum_{x'} |x'\rangle + \frac{1}{\sqrt{N}} \sum_{x \neq x'} |x\rangle \tag{2.20}$$

where O stands for unitary oracle operator.

More generally,

$$O |\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x} (-1)^{f(x)} |x\rangle \tag{2.21}$$

for every computational basis state $|x\rangle$.

Although measuring the system at this point would still result in an outcome similar to the previous one, the state is ready to demonstrate a powerful feature of quantum computing: phase interference (Fig. 2.10[2]), which leads us to the third step: amplification.

[2]Diagram by the author, closely following a diagram from [13]

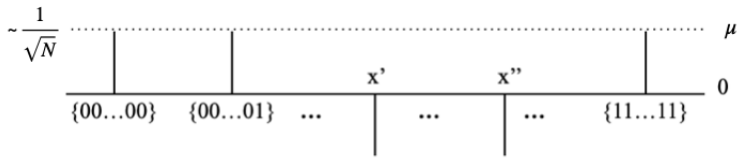

Figure 2.10: After the action of the oracle there is an amplitude flip of the states that represent solutions. The oracle is a circuit that needs to be designed for each particular instance of the problem.

Example of a quantum oracle

To exemplify the implementation of an oracle acting on a quantum state let's consider the uniform superposition of all the basis states of a 2-qubit system,

$$|\psi\rangle = \frac{1}{2}\left(|00\rangle + |01\rangle + |10\rangle + |11\rangle\right) \tag{2.22}$$

and suppose we want to mark the state $|11\rangle$.

We need to design an oracle O such that,

$$O\,|x\rangle = (-1)^{f(x)}\,|x\rangle \tag{2.23}$$

where $f(x) = 1$ if $x = 11$, and 0 otherwise.

This is:

$$O\,|\psi\rangle = \frac{1}{2}\left(|00\rangle + |01\rangle + |10\rangle - |11\rangle\right) \tag{2.24}$$

Which can be achieved through the controlled application of the unitary operator represented by

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.25}$$

in the following way,

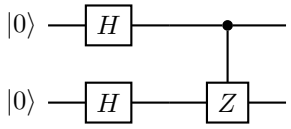

Figure 2.11: Circuit that prepares a uniform superposition over 2 qubits and applies an oracle that flips the phase of $|11\rangle$ using a controlled Z gate.

3. Amplification

The amplification operator, as its name suggests, increases the amplitude of the desired state(s) through an iterative process that rotates the quantum state within the plane spanned by the marked (solution) and non-marked (non-solution) subspaces.

To understand how this works, let's notice that applying a Hadamard gate H to a single qubit transforms the state as,

$$H|x\rangle = \frac{1}{\sqrt{2}}\sum_{y=0}^{1}(-1)^{x\cdot y}|y\rangle \tag{2.26}$$

where $x \cdot y$ is defined as $x \cdot y = xy \mod 2$, ie. the bitwise mod 2 dot product of x and y.

Back to our setting, we start by applying a $H^{\otimes n}$ gate to the state $O|\psi\rangle$, where the oracle O has flipped the phase of the solution state(s). This gives:

$$H^{\otimes n}O|\psi\rangle = \sum_{y\in\{0,1\}^n}\left(\frac{1}{N}\sum_{x\in\{0,1\}^n}(-1)^{f(x)}(-1)^{x\cdot y}|y\rangle\right) \longrightarrow |\psi'\rangle = \sum_{z\in\{0,1\}^n}\alpha_z|z\rangle \tag{2.27}$$

Here, the inner product $x \cdot y$ is defined for bitstrings $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$ as $x \cdot y := \bigoplus_{i=1}^{n} x_i y_i$, ie. the sum mod 2 of the bitwise products.

The resulting state $|\psi'\rangle$ exhibits an interference pattern: amplitudes for some states increase, while others decrease. In particular, amplitudes associated with solution states begin to grow due to constructive interference.

To amplify the solution amplitudes further, we apply a phase shift of -1 to all computational basis states except $|0\rangle$. This step can be written as:

$$\left|\psi^{(1)}\right\rangle = \alpha_0|0\rangle \quad - \sum_{z\in\{0,1\}^n\setminus\{0\}}\alpha_z|z\rangle \tag{2.28}$$

This operation reflects the state about the $|0\rangle$ state and is crucial for increasing the relative amplitude of the solution.

Lastly, we apply a Hadamard gate $H^{\otimes n}$ once more to return to the computational basis.

These three previous steps—Hadamard gate, selective phase shift, and Hadamard gate—together form what is known as the Grover diffusion operator.

The Grover diffusion operator D can be written as,

$$D = H^{\otimes n}(2|0^n\rangle\langle 0^n| - I_n)H^{\otimes n} \quad \longrightarrow \quad D = |\psi\rangle\langle\psi| - I_n \tag{2.29}$$

and its role is to flip the amplitudes of the state around their average, which can be interpreted as a "reflection around the mean" (Fig. 2.12[3]).

Finally, the composition of the oracle operator followed by the diffusion operator defines a single Grover iteration:

$$G = D \cdot O \tag{2.26}$$

which can also be interpreted geometrically as a rotation in the plane spanned by the solution and non-solution subspaces.

[3]Diagram by the author, closely following a diagram from [13]

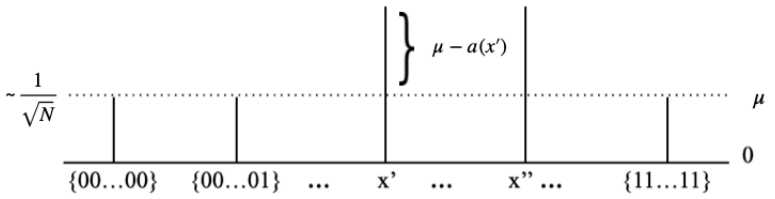

Figure 2.12: After the action of the diffusion operator, the amplitudes a of the desired states x, x' increase.

This is, for a problem with M solutions represented by the states $|x'\rangle$ we can express the initial uniform superposition state $|\psi\rangle$ before the oracle as,

$$|\psi\rangle = \frac{M}{\sqrt{N}} \sum_{x'} |x'\rangle + \frac{N-M}{\sqrt{N}} \sum_{x \neq x'} |x\rangle = \sqrt{\frac{M}{N}} |\beta\rangle + \sqrt{\frac{N-M}{N}} |\alpha\rangle \tag{2.30}$$

for normalized states $|\alpha\rangle$ and $|\beta\rangle$.

Letting,

$$\sqrt{\frac{N-M}{N}} = cos\frac{\theta}{2} \tag{2.31}$$

We have that a single Grover iteration produces the state,

$$G|\psi\rangle = D \cdot O\left(sin\frac{\theta}{2}|\beta\rangle + cos\frac{\theta}{2}|\alpha\rangle\right) = D\left(-sin\frac{\theta}{2}|\beta\rangle + cos\frac{\theta}{2}|\alpha\rangle\right) = \left(sin\frac{3\theta}{2}|\beta\rangle + cos\frac{3\theta}{2}|\alpha\rangle\right) \tag{2.32}$$

Thus, in general,

$$G^t|\psi\rangle = \left|\psi^{(t)}\right\rangle = sin\left(\frac{(2t+1)\theta}{2}\right)|\beta\rangle + cos\left(\frac{(2t+1)\theta}{2}\right)|\alpha\rangle \tag{2.33}$$

It can be proved [12] that after $O(\sqrt{N/M})$ applications of a Grover iteration, the amplitude of the state(s) that encodes the solution(s) to our problem increases sufficiently so that at the end we have a high probability ($\geq 1/2$) of measuring one of our desired states.

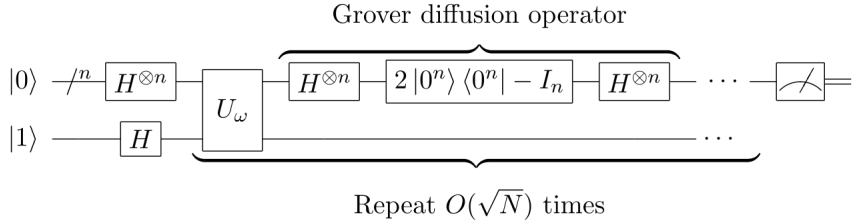

Figure 2.13: Quantum circuit for Grover's algorithm, for $M = 1$.

2.6. Physical Implementation

The physical implementation of a system of qubits is a crucial aspect of realizing practical quantum computers since the underlying platform determines the qubit's performance in terms

of coherence, gate fidelity, and scalability. Several settings have been explored in order to create qubits, each of them emphasizing different aspects of the quantum mechanical nature of the respective systems.

From superconducting circuits and trapped ions to semiconductor spin qubits and photonic systems, these platforms offer specific advantages but also face certain challenges when trying to build robust, large-scale quantum processors. This section explores very briefly the most common physical systems used to implement qubits.

In this context, an effective approach to evaluating the progress and challenges of a quantum computing platform is provided by DiVincenzo's criteria [14] which can be summarized as follows,

1. Scalability, which refers to the capability of having a large set of well-characterized qubits working together.

2. Qubit initialization, which would allow to prepare states repeatedly in an effective manner.

3. Long decoherence times, which would allow to perform operations on qubits before they lose their quantum properties.

4. Universal set of quantum gates, necessary for performing any operation on the qubits.

5. Measurement, which would allow to extract the results of a quantum circuit.

2.6.1. Superconducting Qubits

In building a quantum processor the goal is to have a two-level system capable of performing basic quantum gates. Superconducting qubits are based on superconducting circuits to realize this functionality. The underlying principle lies in considering an LC circuit.

The Hamiltonian of an LC circuit can be modeled as a quantum harmonic oscillator. After quantization we find ourselves with uniformly separated states. These uniform energy levels, however, are not ideal for qubit operations which require a well-defined two-level system.

The solution to this predicament is to introduce anharmonicity to the quantum electric circuit by replacing the linear inductor with a non-linear element like a Josephson junction.

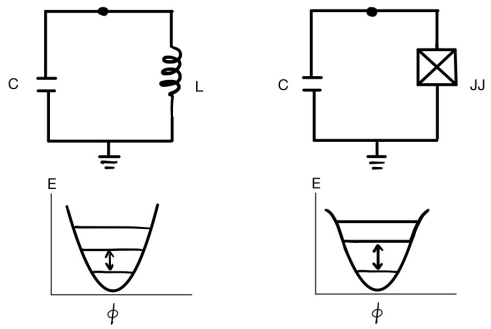

Figure 2.14: On the right, a linear LC oscillator with equidistant energy levels. On the left, a non-linear circuit created by introducing a Josephson junction (JJ).

There exist three main types of superconducting qubits: charge qubits, phase qubits, and flux qubits. They are classified according to the ratio of the Josephson energy (E_J) to the charge energy (E_C), which determines the degree of freedom that dominates the qubit's behavior.

In this setting the implementation of single-qubit gates is achieved by applying microwave pulses tuned to specific frequencies in order to induce transitions between the $|0\rangle$ and $|1\rangle$ states.

There exist different methods to enable interactions between qubits, including capacitive and inductive coupling between neighboring qubits. However, one of the most effective approaches is to use tunable couplers since they offer more flexibility and control.

As for the measurement, it is often performed by inducing Rabi oscillations and observing the system's response via a scattered electromagnetic signal.

The primary advantage of this platform lies in the ease with which each circuit component can be designed and fabricated. However, while the strong interaction with electromagnetic radiation is a key strength of superconducting qubits, it also makes the system particularly susceptible to noise.

2.6.2. Trapped Ions

In this case, as the name suggests, charged particles (ions) serve as the basic carriers of information. The creation of qubits in this setting involves an ion source and ion traps which employ laser cooling techniques in order to maintain the ions nearly stationary.

The most widely used method for trapping ions is the linear Paul trap, which consists of four electrodes that produce oscillating electric potentials, creating a time-averaged pseudo-potential in which a chain of ions can get uniformly arranged and act as a quantized harmonic oscillator with translational and vibrational modes.

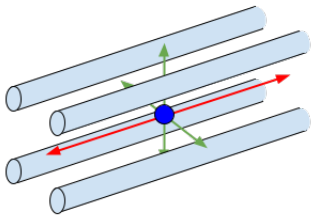

Figure 2.15: Paul trap schematics [24].

One of the most used ions is the $^{40}Ca^{+}$. The $^{40}Ca^{+}$ ion possesses relatively stable ground and excited states. The fact that these states can be manipulated by electromagnetic fields, without the need of the system interacting with other external physical elements that introduce noise, makes this platform suitable for long decoherence times and quantum control.

State manipulation is performed using laser pulses tuned to specific amplitudes, frequencies and durations. Depending on the parameters of the laser pulses, various operations can be performed on the qubit states. Additionally, the excitation of the system's vibrational modes

enables the implementation of multi-qubit gates.

For the readout process, one can take advantage of the excited states only available when the qubit is in a particular state. In this way the repeated measurement of a fluorescence signal can reveal the information about the quantum state of the system.

The main advantages of this platform are long coherence times, the ability to isolate the system from noisy environments and the high fidelity levels that can be achieved for the basic quantum operations.

The primary disadvantage of this platform lies in addressing the first DiVincenzo criterion: scalability. As the length of the ion chain increases, the spacing between vibrational states becomes narrower and so the precision to manipulate states becomes more difficult to achieve.

2.6.3. Photonic Qubits

This platform uses photons as the physical system for qubits, allowing for the choice between two sets: discrete or continuous-valued variables. In the former case, the polarization of a photon gives two well-defined basis states naturally, namely horizontal and vertical polarizations.

A major advantage of this platform is that the field of optics is quite developed. For instance, implementing quantum gates involves manipulating experimental equipment — such as beam splitters and phase shifters — which is already well characterized and provides high accuracy and reliability.

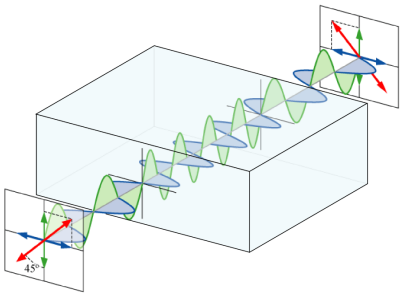

Figure 2.16: A waveplate induces a phase shift between the two perpendicular polarization components of a light wave [23].

A waveplate, for example, can modify the polarization of a photon, enabling the implementation of single-qubit gates. For entanglement one can rely on linear optical instruments, such as beam-splitters, to create Bell states.

As for the measurement stage, the process is relatively straightforward, as it primarily involves the use of photodetectors.

Another key advantage of this platform is the ease of transmitting photons over large distances with low decoherence. This feature allows, in principle, the development of quantum networks that can eventually become part of a larger scheme: a quantum internet.

In this case, as with trapped ions, the primary disadvantage comes from considering the scalability criterion, with photon loss due to absorption or scattering phenomena being the main

limiting factor.

Although the physical platform does not affect directly the theoretical formulation of a quantum algorithm, it can influence its implementation, especially in terms of gate depth, coherence times, and the scalability. For example, superconducting qubit systems are currently among the most promising candidates for near-term applications of quantum search, including quantum-enhanced cryptographic protocols such as Blockchain consensus mechanisms.

Chapter 3

A Quantum Algorithm for Proof-of-Work

As mentioned before, one of the main purposes of Blockchain technology is to be decentralized, that is, to be independent of any third party while also avoiding any member of the network having control over the entire system.

But then the following question arises, how can participants prevent malicious transactions like false operations and double spending occurrences? The answer provided by Nakamoto comes in the form of a consensus mechanism.

In this chapter, we are going to explain the workings of the Proof-of-Work mechanism.

Proof-of-work (PoW) is a consensus mechanism by which participants of a network can validate the information aggregated to the blockchain. The mechanism is devised to ensure that no participant takes control of the network.

PoW requires the miner to find a hash whose value is under a given target. The task involves spending massive resources (computational power) to solve a computational "puzzle" that requires no skill but only brute force. Thus, the idea of using a quantum algorithm to perform PoW is only natural.

It has been stated that quantum miners could have a computational advantage over classical ones [6]. By the end of this chapter, we will have shown how to implement Grover's algorithm as a sub-routine for mining in Proof-of-Work.

3.1. Proof-of-Work Mechanism

When the use of the Internet escalated, so did spam messages and other forms of service exploitation such as denial-of-service attacks, which consist in flooding a service with expendable requests so as to prevent its availability to legitimate users.

The solution to this problem was to charge the solicitors of a service for carrying out their request. The cost would have to be moderate for regular users but unprofitable for spammers.

Instead of money, one proposal was to ask the solicitors to make use of their computational power and present this as a proof of expended resources —work. The verification of their

"payment", though, would have to be easily realized by the service providers.

Requesting the solution of a mathematical problem seemed to be an appropriate cost since there exist some problems for which the solution is hard to find but easy to verify. This is the basic concept of Proof-of-Work.

Proof-of-Work is a cryptographic mechanism by which a participant can prove to the other participants in a network that they have expended a certain amount of computational resources.

In the Bitcoin setting though, the proof-of-work mechanism makes sure that the Blockchain system is hard to manipulate. If a malicious user wanted to tamper the data in a particular block, he would have to tamper with all the previous blocks in the chain to do so. And that action would cost them a non realizable amount of computational power.

Hence, PoW serves as a consensus mechanism in a non-trust environment such as that of Bitcoin and other cryptocurrencies.

3.2. Mining in PoW

Mining is the process, in Proof-of-Work cryptocurrencies such as Bitcoin, by which individuals add new blocks to the chain. The incentive comes as a monetary reward for the miner who first succeeds in solving a computationally hard problem.

The problem resides in finding a number —also known as nonce— such that the hash value of the header of the new block is less than a certain value —also known as the target hash.

The most widely used PoW consensus is based on SHA-256, whose block size is 512 bits.

The header of a block contains the following information:

Data	Size (Bytes)	Description
Version	4	Version number
Previous hash	32	Hash of the previous block header
Merkle root node	32	Root node of the transactions of new block
Time	4	Timestamp of new block
Bits	4	Encoded target
Nonce	4	Number to update

Table 3.1: A block's header data.

During mining, a miner will concatenate together all of the above information and since most of the data is already determined, the task of a miner consists in trying different numbers repeatedly until the adequate nonce produces a hash value equal or below the target. The target is a 256-bit number available to all the network.

The classical algorithm for mining can be summarized as follows:

Classical Mining Algorithm

Input: Hash value of most recent block's header: $\texttt{h}_{\texttt{prev}}$
Input: Block data: $\texttt{version}, \texttt{merkle\_root}, \texttt{timestamp}, \texttt{bits}$
Input: Target value: $\texttt{target}$
Input: Initialize $\texttt{nonce} = 0$
Output: New candidate block's header and valid nonce

Steps:

1. Concatenate: $\texttt{block\_header\_fixed} = \texttt{version} \,\|\, \texttt{merkle\_root} \,\|\, \texttt{h}_{\texttt{prev}} \,\|\, \texttt{timestamp} \,\|\, \texttt{bits}$
2. **while** $\texttt{nonce} < 2^{32}$:
   Build new block's header by appending: $\texttt{block\_header} = \texttt{block\_header\_fixed} \,\|\, \texttt{nonce}$
   Compute $\texttt{h}_{\texttt{new}} = \text{SHA256}(\text{SHA256}(\texttt{block\_header}))$
   if $\texttt{h}_{\texttt{new}} <= \texttt{target}$:
   return $(\texttt{nonce}, \texttt{block\_header})$
   Broadcast new candidate block
   else:
   $\texttt{nonce} = \texttt{nonce} + 1$
3. If candidate block is verified, fetch reward

For a given set of transactions, a candidate block can have 2^{32} different nonces. This means that the search space contains 4,294,967,296 instances.

The target is adjusted once every 2016 mined blocks —once every two weeks, approximately— so that a block is added to the chain every 10 minutes on average. The difficulty of the problem depends on the target (Diff = 2^{256}/target). In this case, the lower the target, the harder it is to mine a new block.

3.3. Quantum Algorithm for PoW

PoW requires the miner to find a message digest which value is under a given target. Previously, Bard et al. [6] provided an analysis for a quantum Bitcoin miner that uses Grover's algorithm to solve this problem. However, an explicit algorithm was not given.

In this work we develop an algorithm that uses Grover's search as a sub-routine and that performs PoW for Bitcoin (or any other PoW-based cryptocurrency).

Quantum Search Algorithm for Classical PoW

Input: Hash value of most recent block's header: $\texttt{h}_{\texttt{prev}}$
Input: Block data: $\texttt{version}, \texttt{merkle\_root}, \texttt{timestamp}, \texttt{bits}$
Input: Target value: $\texttt{target}$
Output: New candidate block's header and valid nonce

Steps:

1. Concatenate: $\texttt{block\_header\_fixed} = \texttt{version} \,\|\, \texttt{merkle\_root} \,\|\, \texttt{h}_{\texttt{prev}} \,\|\, \texttt{timestamp} \,\|\, \texttt{bits}$ and initialize the state $|\texttt{block\_header\_fixed}\rangle$
2. Initialize a 32-qubit quantum register to $|0\rangle$ and perform a uniform superposition on all of them through a $\text{H}^{\otimes 32}$ gate in order to create the state $|\texttt{nonce}\rangle$
3. For the oracle qubit, initialize one quantum register to $|0\rangle$ and apply quantum gate X followed by a Hadamard gate H creating the state $|-\rangle$

4. Perform Grover's algorithm:
   for i in range $(0, k)$: (k being the optimal number of Grover iterations)
     Perform the oracle
     Perform Grover's diffusion operator
   Measure final state $|\texttt{nonce}\rangle \rightarrow \texttt{nonce}$
   Build new block's header by appending: $\texttt{block\_header} = \texttt{block\_header\_fixed} \,||\, \texttt{nonce}$
   Compute $\texttt{h}_{\texttt{new}}$ = SHA256(SHA256((`block_header`))
   if $\texttt{h}_{\texttt{new}} <= \texttt{target}$
     return `nonce`
5. Create and broadcast new candidate block
6. If candidate block is verified, fetch reward

For this purpose we introduce a number of subroutines or protocols that are to be implemented as part of the quantum search algorithm.

The subroutines we have designed describe the processing of the input (blockchain most recent header, transactions, nonce) through a quantum circuit that calculates its hash value (SHA-256) and implements Grover's algorithm as a subroutine in order to find the adequate nonce that will produce the required message digest. Below, we describe said protocols.

3.4. The oracle

To implement the oracle we apply SHA-256 twice on the padded input (as performed in Bitcoin's PoW). Since SHA-256 is realized as a reversible circuit, we need to perform uncomputation operations and we may use the intermediate hash after the second round is applied.

Oracle for PoW quantum search algorithm

Input: $|\texttt{block\_header\_fixed}\rangle$
Input: $|\texttt{nonce}\rangle$
Input: Target value: `target`
Output: SHA256(SHA256$|\psi\rangle$)

Steps:

1. **for** i in range $(0, 1)$:
     Perform the padding protocol
     Perform the creation-of-message-schedule protocol
     Perform the compression protocol
2. Perform the target-comparison protocol

3.4.1. Padding protocol

In this protocol we encode the information for the new candidate block and perform the padding scheme corresponding to the hash function SHA-256:

Padding protocol

Input: $|\texttt{block\_header\_fixed}\rangle \otimes |\texttt{nonce}\rangle$
Output: $|\texttt{padded}\rangle$

Steps:

1. Implement the padding scheme:

a) Initialize one quantum register to $|1\rangle$
b) Pad with enough quantum registers initialized in the zero state $|0\rangle$
c) Create a 64-qubit quantum register to store the big-endian integer representing the length of the message, $|\texttt{length}\rangle$

2. The fiinal state is $|\texttt{padded}\rangle = |\texttt{block\_header\_fixed}\rangle \otimes |\texttt{nonce}\rangle \otimes |1\rangle \otimes |00...00\rangle \otimes |\texttt{length}\rangle$

The first round will produce a 1024-qubit state, while the second will generate a 512-qubit state block.

3.4.2. Creation of message schedule

In this protocol, we create the variables that are going to be needed for the main loop of the SHA-256 function. SHA-256 algorithm is very similar to the SHA-1 algorithm. The main difference is the hash length.

The cryptographic function SHA-256 requires sixty four words W_i and sixty four variables K_i for the compression loop, each word and variable being 32 bits long.

The first sixteen words come directly from a 512-bit message block B produced during the padding protocol. For each block $B \in \{0,1\}^{512}$:

$$B = W_0 \parallel W_1 \parallel ... \parallel W_{15} \tag{3.1}$$

For the remaining words W_i where $16 \leq i \leq 63$, the following operations are needed:

$$W_i = \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} \mod 2^{32} \tag{3.2}$$

Here, the functions σ_0 and σ_1 are defined as:

$$\sigma_0(W) = rightrotate^7(W) \oplus rightrotate^{18}(W) \oplus rightshift^3(W) \tag{3.3}$$
$$\sigma_1(W) = rightrotate^{17}(W) \oplus rightrotate^{19}(W) \oplus rightshift^{10}(W) \tag{3.4}$$

with $\oplus$ being the bitwise XOR operation.

As for the K_i variables, they are the big-endian integers representing the first 32 bits of the fractional part of the cubic root of each of the first sixty four prime numbers, respectively.

Creation of message schedule protocol

Input: $|\texttt{padded}\rangle$
Output: $|W_i\rangle$, $|K_i\rangle$ for $0 \leq i \leq 63$

Steps:

1. Create 32×16 quantum registers and initialize the first sixteen states $|W_i\rangle$, $0 \leq i \leq 15$
2. Create 32×48 quantum registers and compute the states $|W_i\rangle$, $16 \leq i \leq 63$
3. Calculate the values of K_i classically, create 32×64 quantum registers and create the 64 states $|K_i\rangle$, $0 \leq i \leq 63$

Classical operations like $rightrotate^n$ and $rightshift^n$ can be implemented through the use of SWAP gates.

3.4.3. Compression loop

At this stage, the cryptographic function SHA-256 requires also eight 32 bits length variables H_i, $0 \leq i \leq 7$. The initial values of H_i are the big-endian integers representing the first 32 bits of the fractional part of the square root of each of the first eight prime numbers, respectively.

This subroutine describes the main loop of the SHA-256 function which processes sequentially each one of the blocks $B \in \{0,1\}^{512}$ generated in the padding scheme.

For each $B \in \{0,1\}^{512}$ (one at a time) we set:

$$(a, b, c, d, e, f, g, h) = (H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7) \tag{3.5}$$

And perform the main compression loop, ie. we perform iteratively 64 rounds consisting of the following operations:

$$T_1 = h + \Sigma_1(e) + Ch(e, f, g) + K_i + W_i \mod 2^{32} \tag{3.6}$$
$$T_2 = \Sigma_0(a) + Maj(a, b, c) \mod 2^{32} \tag{3.7}$$
$$h = g \tag{3.8}$$
$$g = f \tag{3.9}$$
$$f = e \tag{3.10}$$
$$e = d + T_1 \mod 2^{32} \tag{3.11}$$
$$d = c \tag{3.12}$$
$$c = b \tag{3.13}$$
$$b = a \tag{3.14}$$
$$a = T_1 + T_2 \mod 2^{32} \tag{3.15}$$

Here, the operations Σ_1, Σ_0, Ch and Maj are defined as:

$$\Sigma_0(x) = rightrotate^2(x) \oplus rightrotate^{13}(x) \oplus rightrotate^{22}(x) \tag{3.16}$$
$$\Sigma_1(x) = rightrotate^6(x) \oplus rightrotate^{11}(x) \oplus rightrotate^{25}(x) \tag{3.17}$$
$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) \tag{3.18}$$
$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \tag{3.19}$$

At the end of the main compression loop, we update the variables:

$$H_0 = H_0 + a \tag{3.20}$$
$$H_1 = H_1 + b \mod 2^{32} \tag{3.21}$$
$$H_2 = H_2 + c \mod 2^{32} \tag{3.22}$$
$$H_3 = H_3 + d \mod 2^{32} \tag{3.23}$$
$$H_4 = H_4 + e \mod 2^{32} \tag{3.24}$$
$$H_5 = H_5 + f \mod 2^{32} \tag{3.25}$$
$$H_6 = H_6 + g \mod 2^{32} \tag{3.26}$$
$$H_7 = H_7 + h \mod 2^{32} \tag{3.27}$$
$$\tag{3.28}$$

and proceed to perform the main compression loop on the next message block.

After the last message block has been processed, the hash of the message is:

$$H = H_0 \parallel H_1 \parallel ... \parallel H_7 \tag{3.29}$$

We now proceed to describe the quantum circuit that implements the main compression loop of the function SHA-256.

Compression protocol

Input: $|W_i\rangle$, $|K_i\rangle$ for $0 \leq i \leq 63$
Output: SHA256 $|\psi\rangle$

Steps:

1. Create 32×8 quantum registers and initialize the eight states $|a\rangle, |b\rangle, ..., |h\rangle$
2. For each message block B perform the main compression loop (Fig. 3.1) sequentially:

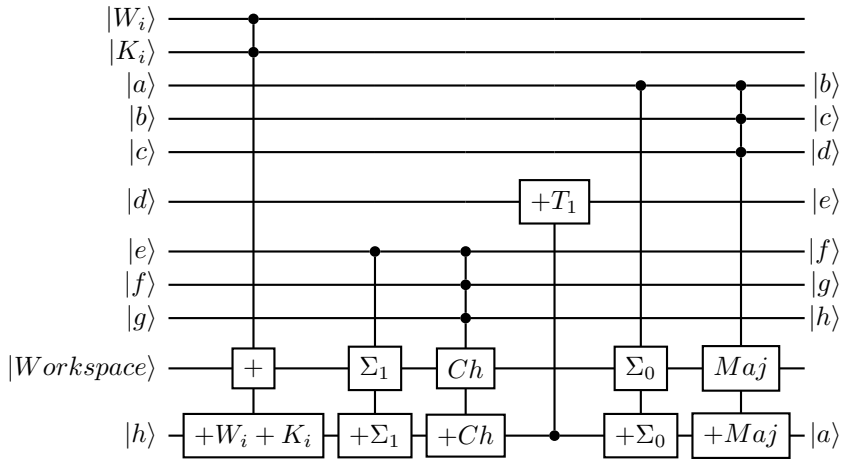

Figure 3.1: Quantum circuit schematic for one of the 64 rounds of the main compression loop. The circuit makes use of a workspace register in order to implement the quantum analogs of the functions Σ_0, Σ_1, Ch and Maj, which are combined through additions mod 32 to compute the intermediate values T_1 and T_2. In this circuit, the control dots shown do not represent CNOT or Multi-CNOT gates; instead, they indicate that the corresponding qubits act as controls for the composite operations targeting the workspace register.

3. The final state is: SHA256$|\psi\rangle = |H_0\rangle \otimes |H_1\rangle \otimes |H_2\rangle ... \otimes |H_6\rangle \otimes |H_7\rangle$

Classical operations like NOT($\neg$), AND($\wedge$) and XOR($\oplus$) can be implemented through the use of basic quantum gates like X, CNOT and Toffoli gates.

3.4.4. Target Comparison Protocol

This protocol is also part of the oracle gate of our algorithm. Its purpose is to determine which states meet the requirements of our solution by comparing them with the target value, which is fixed and depends on the settings of the PoW network. If the computed digest is less than or equal to the target, we apply a phase flip on the states that satisfy the PoW condition.

Target Comparison Protocol

Input: Target value: `target`
Input: SHA256(SHA256$|\psi\rangle$))

Steps:

1. Create a 256-qubit quantum register and initialize the state $|target\rangle$
2. Perform a comparison between SHA256(SHA256$|\psi\rangle$)) and $|target\rangle$ and mark the sought element(s) by making use of the oracle qubit $|-\rangle$

   The comparison can be done with a quantum binary comparator (Fig.3.2) as developed in [21] adapted for the case of two 256-qubit states.

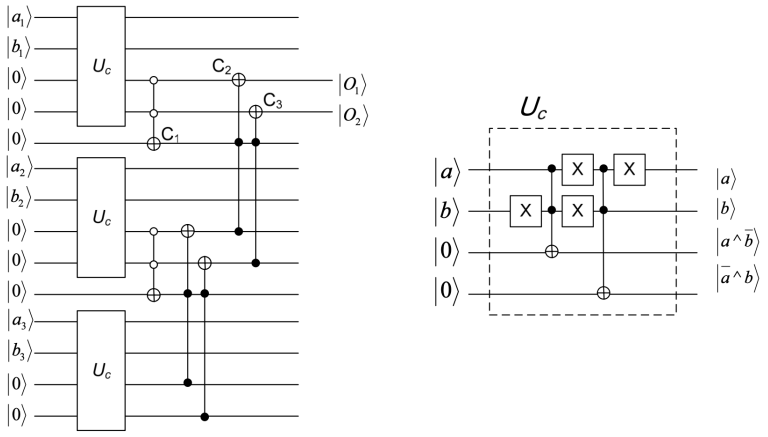

Figure 3.2: Quantum circuit that compares two 3-qubit states: $|a\rangle = |a_1, a_2, a_3\rangle$, $|b\rangle = |b_1, b_2, b_3\rangle$. Each U_c gate performs a reversible comparison a pair of bits outputting results onto two ancilla qubits that encode whether $a_i < b_i$, $a_i > b_i$, $a_i = b_i$. The results are then propagated using Toffoli gates C_i to carry the comparison result from one significant bit position to the next. The final state of the output qubits $|O_1\rangle$ and $|O_2\rangle$ allows to know if $a < b$, $a > b$, $a = b$. Diagram from source [21].

Chapter 4

Conclusions

We estimate that our algorithm prototype (Fig. 4.1)requires at least 5,000 logical qubits. The main contribution to this quantity comes from the qubits involved in the quantum oracle, which is dedicated to the compression protocol.

The target comparison protocol may need fewer qubits since we could reduce the 256 qubits to a lesser number considering that a valid hash under the target starts with a fixed number of zeroes.

The workspace of the oracle however, needs sufficient logical qubits to be able to perform the operations needed in the algorithm. These qubits are often called *ancilla* qubits and in order to re utilize them it is necessary to uncompute the operations applied to them.

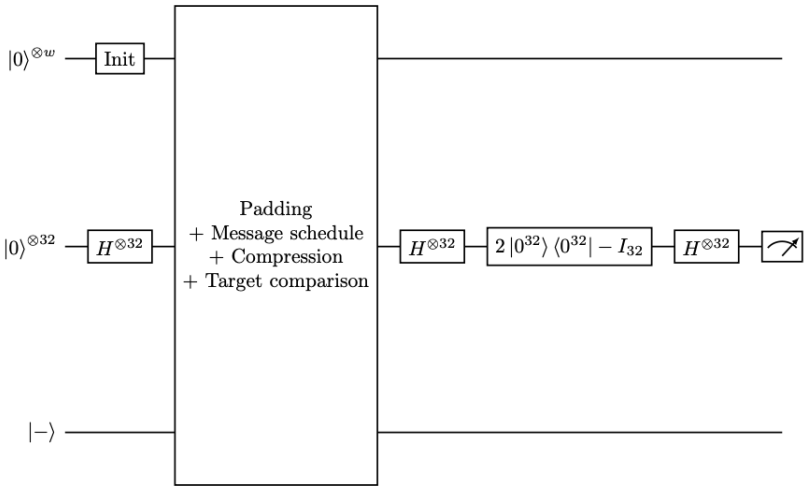

Figure 4.1: Quantum circuit for a Grover-based PoW algorithm. The workspace (first register) is initialized to store classical input data and ancillae used in intermediate computations. The second register represents the nonce, initialized in a uniform superposition using Hadamard gates. The oracle consists of the padding, message schedule, compression rounds, and target comparison protocols, all of which must be implemented reversibly. This oracle, along with the amplification steps (Hadamard layers and reflection about the mean), must be repeated according to the optimal number of Grover iterations. The oracle qubit is initialized in the $|-\rangle$ state and is used to mark valid solutions via phase kickback.

When trying to determine the complexity of a computational algorithm though, we often turn to look at the average number of queries we need to do in order to solve it. As we mentioned before, classically, the solution to the unstructured search problem requires in average N/M queries, with M being the number of elements we look for.

There is a mathematical notation named *Big O notation* $O(f(n))$ that helps us describe the behavior of an algorithm when the input tends to a certain value or infinity. Big O notation takes into account the worst case scenario and this is helpful because it allows us to estimate an upper bound for the running time of an algorithm.

For example, to describe a linear-time algorithm, we say that it is of order N and often write: $O(N)$.

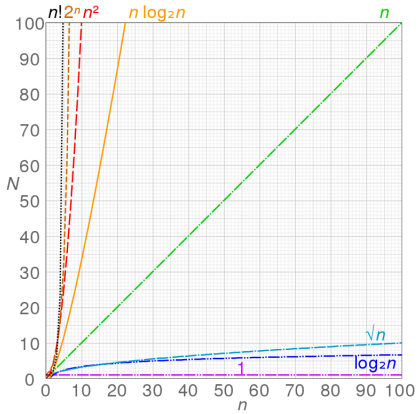

Figure 4.2: Number of operations N vs input size n.

Grover's algorithm uses $O(\sqrt{N/M})$ queries [11]. Also, it uses $O(log_2 N)$ qubits; and only $O(\sqrt{N} log_2 N)$ gates —when there is one element to find [16].

For our algorithm though, we do not know exactly what M is equal to. Luckily, for $M \leq N/2$ unknown, the algorithm complexity is still $O(\sqrt{N})$ [12].

The number of qubits in our algorithm is currently not feasible and therefore cannot compete with classical search algorithms. However our study shows that given a scalable device, there exists an improvement over classical machines.

As of now, a hash function with an output of 256 bits is not threatened by quantum computing. A quantum computer would need to have enough logical qubits, sufficiently long decoherence times and sophisticated quantum error correction techniques.

Future work on this topic includes the gate optimization of the algorithm and an estimation of the gate count. With this information, we will be able to determine the repercussions of the algorithm in a PoW setting and approximate the quantum advantage.

Bibliography

[1] Bravyi S, Gosset D, and König R., *Quantum advantage with shallow circuits.* In: Science 362 (2018). doi: https://doi.org/10.1126/science.aar3106

[2] Rivest RL, Shamir A, and Adleman L., *A method for obtaining digital signatures and public-key cryptosystems.* In: Communications of the ACM 21 (1978). doi: https://doi.org/10.1145/359340.359342

[3] Kearney JJ and Perez-Delgado CA., *Vulnerability of blockchain technologies to quantum attacks.* In: Array 10 (2021). doi: https://doi.org/10.1016/j.array.2021.100065

[4] Santiago-Alarcon D et al., *Quantum aspects of evolution: a contribution towards evolutionary explorations of genotype networks via quantum walks.* In: Journal of the Royal Society Interface 17 (2020). doi: http://dx.doi.org/10.1098/rsif.2020.0567

[5] Cao Y, Romero J, et al., *Quantum chemistry in the age of quantum computing.* In: Chemical Reviews (2019). doi: https://doi.org/10.48550/arXiv.2112.00760

[6] Bard DA, Kearney JJ, and Perez-Delgado CA., *Quantum Advantage on Proof of Work.* In: arXiv (2021). url: https://arxiv.org/pdf/2105.01821

[7] Montanaro Ashley, *Quantum Algorithms: an overview.* In: npj Quantum Information (2016). doi: https://doi.org/10.1038/npjqi.2015.23

[8] Nakamoto Satoshi, *Bitcoin: a peer to peer electronic cash system.* In: Array 322 (2021). doi: https://doi.org/10.1016/j.array.2021.100065

[9] Haber S and Stornetta WS, *How to time-stamp a digital document.* In: J. Cryptology 3, 99-111 (1991). doi: https://doi.org/10.1007/BF00196791

[10] Barenco A, Bennett CH et al., *Elementary gates for quantum computation.* In: Phys. Rev. A 52, 3457. doi: https://link.aps.org/doi/10.1103/PhysRevA.52.3457

[11] Grover Lov K., *A fast quantum mechanical algorithm for database search.* In: Annual ACM Symposium on Theory of Computing (1996). doi: https://doi.org/10.1145/237814.237866

[12] Nielsen MA and Chuang IL, (2011) *Quantum Computation and Quantum Information.* Cambridge University Press.

[13] Wright, John, (2015) *Lecture 4: Grover's Algorithm.* In: https://www.cs.cmu.edu/ odonnell/quantum15/lecture04.pdf

[14] DiVincenzo, David, (1996) *Topics in quantum computers.* In: Mesoscopic Electron Transport.

[15] Devoret, M.H, et al., (2024) *Superconducting Qubits: A Short Review.* doi: https://doi.org/10.48550/arXiv.cond-mat/0411174

[16] Grover Lov K., *Trade-offs in the quantum search algorithm.* In: Phys. Rev. A 66 (2002). doi: https://link.aps.org/doi/10.1103/PhysRevA.66.052314

[17] Holzscheiter, Michael H., (2002) *Ion-trap quantum computation.* In: Los Alamos Science, Number 27

[18] Romero, Jacquiline and Milburn, G., (2024) *Photonic quantum computing.* doi: https://doi.org/10.48550/arXiv.2404.03367

[19] Aggarwal, D., Brennen, G. K., Lee, T., Santha, M., and Tomamichel, M., (2017). *Quantum attacks on Bitcoin, and how to protect against them.* doi: https://doi.org/10.48550/arXiv.1710.10377

[20] National Institute of Standards and Technology, (2015) *Secure Hash Standard (SHS).* FIPS PUB 180-4. In: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[21] Sena Oliveira, David and Viana Ramos R., (2007) *Quantum bit string comparator: circuits and applications.* In: Quantum Computers and Computing, Vol. 7.

[22] Watrous, John, (2024) *Grover's Algorithm.* In: https://learning-api.quantum.ibm.com/assets/73fab28f-4ed4-41e4-8f8a-9a6f858a8bb2

[23] Mellish B. CC BY-SA 4.0 https://creativecommons.org/licenses/by-sa/4.0, via Wikimedia Commons

[24] Mbrzeczek CC BY-SA 4.0 https://creativecommons.org/licenses/by-sa/4.0, via Wikimedia Commons