From Classical Complexity to Quantum Efficiency: The Mathematical Basis of Shor's Algorithm

Tyler Rose

July 15, 2024

Abstract

As RSA encryption secures over half of all HTTPS servers and quantum computing advances rapidly, with volumes increasing tenfold annually since 2020, Shor's algorithm emerges as a potential cryptographic game-changer. This paper provides a comprehensive exploration of Shor's algorithm, from its mathematical foundations to its far-reaching implications. We analyze its structure, demonstrating its ability to factor large integers in polynomial time—a task believed intractable for classical computers. Through complexity analysis and examples, we illustrate the exponential speedup offered by Shor's algorithm over classical methods. We examine its profound implications for current cryptographic systems and discuss the emerging field of post-quantum cryptography. As quantum computing evolves, understanding Shor's algorithm becomes crucial for the future of secure digital communication.

1 Introduction

As of 2021, approximately 52% of all HTTPS servers and 75% of digital certificates use RSA encryption, safeguarding trillions of online transactions and communications daily [3]. Meanwhile, the quantum computing landscape is evolving rapidly, with quantum volumes increasing by approximately 10 fold every year since 2020 [1]. What does this rapidly advancing field of quantum computing have to do with encryption? The answer lies in a groundbreaking quantum algorithm called Shor's algorithm, which has the potential to completely shift the landscape of cryptography. As quantum computers continue to grow in power and capability, the implications of this algorithm become increasingly significant.

The security of RSA and many other encryption systems relies on the computational difficulty of integer factorization. This mathematical problem, which involves breaking down a large number into its prime factors, forms the basis for most modern encryption algorithms. The following subsections delve deeper into the intricacies of this problem and its far-reaching implications in the world of cryptography and quantum computing.

1.1 The Problem of Integer Factorization

Integer factorization has long been a central problem in number theory and computer science. While factoring small numbers is relatively straightforward, the difficulty increases exponentially with the size of the number. For instance, factoring a 2048-bit number using the most efficient classical algorithms would take billions of years, even with the most powerful supercomputers available today.

This inherent difficulty of factoring large numbers is not merely an academic curiosity. It forms the foundation of many cryptographic systems, most notably RSA (Rivest-Shamir-Adleman) encryption. RSA relies on the presumed difficulty of factoring the product of two large prime numbers. The security of this system, and by extension, much of our digital infrastructure, hinges on the belief that no efficient algorithm exists for factoring large numbers on classical computers.

1.2 Cryptographic Significance

The significance of integer factorization in cryptography cannot be overstated. RSA encryption, which depends on the difficulty of this problem, is widely used in secure data transmission, digital signatures, and key exchange protocols. It protects sensitive information in various sectors, including finance, healthcare, and government communications.

However, the landscape of cryptography is on the brink of a major shift due to the advent of quantum computing. Quantum computers, leveraging the principles of quantum mechanics, can perform certain algorithms exponentially faster than classical computers. This capability poses a significant threat to current cryptographic systems, as quantum computers could potentially break encryption methods that are considered secure today.

1.3 Peter Shor

At the heart of this cryptographic revolution is Peter Shor, an American mathematician and professor at MIT. In 1994, Shor developed his eponymous algorithm while working at AT&T Bell Laboratories. His breakthrough came at a time when quantum computing was still a largely theoretical field, with no practical quantum computers in existence.

Shor's work was inspired by earlier quantum algorithms, particularly those developed by David Deutsch and Richard Jozsa. However, Shor's algorithm was the first to demonstrate a practical application where quantum computers could exponentially outperform classical computers on a problem of wide interest.

The development of Shor's algorithm was not just a mathematical curiosity; it represented a paradigm shift in our understanding of computational complexity and the potential power of quantum computers. It sparked intense interest in quantum computing and quantum-resistant cryptography, areas that continue to be at the forefront of research and development today.

1.4 The Breakthrough: Polynomial-Time Factorization

Shor's seminal work introduced an algorithm that can factor integers in polynomial time on a quantum computer. This stands in stark contrast to the subexponential time required by the best known classical algorithms. Specifically, Shor's algorithm can factor an n-bit number in $O(n^3)$ time and $O(n)$ space, a dramatic improvement over classical methods.

This breakthrough has far-reaching implications. It suggests that once large-scale quantum computers become a reality, many current cryptographic systems will be vulnerable to attack. The algorithm's efficiency stems from its clever use of quantum parallelism and the quantum Fourier transform to find the period of a carefully chosen function, which can then be used to factor the target number.

In the following sections, we will delve deeper into the mechanics of Shor's algorithm. We'll explore the necessary background in quantum computing, including concepts like quantum superposition and entanglement. We'll then break down the algorithm step by step, from state preparation through quantum Fourier transform to measurement and classical post-processing. We'll analyze its complexity, compare it with classical factoring methods, and discuss its implementation challenges. Finally, we'll examine the profound implications of this algorithm for the future of cryptography and the ongoing efforts to develop quantum-resistant encryption methods. Through this exploration, we aim to provide a comprehensive understanding of one of the most significant algorithms in the field of quantum computing.

2 Background

This section covers the necessary prerequisites for understanding Shor's algorithm, including relevant concepts from classical computing, quantum computing basics, and the quantum Fourier transform. [2]

2.1 Classical Computing Concepts

To appreciate the significance of Shor's algorithm, it's crucial to understand certain fundamental concepts from classical computing, particularly in the areas of computational complexity and factoring algorithms.

Computational complexity measures the resources required to solve a problem as a function of the input size. In algorithm analysis, we're primarily concerned with time complexity, which quantifies how an algorithm's running time scales with its input size. Two key complexity classes are relevant to our discussion: polynomial time and exponential time.

An algorithm runs in polynomial time if its running time is upper bounded by a polynomial expression in the input size, formally $O(n^k)$ for some constant k, where n is the input size. Problems solvable in polynomial time are generally considered tractable. In contrast, an algorithm runs in exponential time if its running time is $O(2^{n^k})$ for some constant k. Exponential-time problems are generally considered intractable for large inputs.

Integer factorization, the problem of decomposing a composite number into its prime factors, is of particular interest due to its importance in cryptography. The best known classical algorithms for integer factorization are subexponential but superpolynomial in complexity. Some notable examples include:

- **Trial division:** The simplest factoring algorithm, with a time complexity of $O(\sqrt{n})$. While polynomial in $\sqrt{n}$, it's exponential in the number of bits of n.
- **Pollard's rho algorithm:** A probabilistic algorithm with an expected running time of $O(\sqrt[4]{n})$.
- **General number field sieve (GNFS):** Currently the most efficient known classical algorithm for factoring large integers, with a subexponential time complexity of $O(e^{((64/9)^{1/3}+o(1))(\ln n)^{1/3}(\ln \ln n)^{2/3}})$.

The superpolynomial nature of these algorithms makes factoring large numbers computationally infeasible with classical computers. For instance, factoring a 2048-bit number using the GNFS would take billions of years even with the most powerful classical supercomputers available today. This computational difficulty forms the basis for the security of many cryptographic systems, including RSA.

The apparent intractability of integer factorization on classical computers has led to its widespread use in cryptography. However, as we will see, quantum computing offers a radically different approach to this problem. Shor's algorithm achieves polynomial-time factoring by leveraging the unique properties of quantum systems, potentially undermining the security of systems that rely on the difficulty of this problem.

2.2 Quantum Computing Fundamentals

Quantum computing harnesses the principles of quantum mechanics to process information in ways that are fundamentally different from classical computing. This section introduces the key concepts of quantum computing that are essential for understanding Shor's algorithm.

2.2.1 Qubits and Quantum States

The fundamental unit of quantum information is the quantum bit, or qubit. Unlike classical bits, which can only be in one of two states (0 or 1), qubits can exist in a superposition of states, allowing for much more complex information processing.

The state of a qubit is represented mathematically by a state vector in a two-dimensional complex vector space, known as a Hilbert space. This vector space is spanned by two orthonormal basis vectors, conventionally denoted as $|0\rangle$ and $|1\rangle$, which correspond to the classical bit states 0 and 1, respectively. These basis states are represented in vector form as:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{1}$$

In quantum mechanics and quantum computing, we use the Dirac or bra-ket notation to represent state vectors. A ket $|\psi\rangle$ represents a column vector and denotes a quantum state, while a bra $\langle\psi|$ represents the conjugate transpose of the ket, i.e., a row vector.

The general state of a qubit can be written as a linear combination of the basis states:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2}$$

where α and β are complex numbers called probability amplitudes, satisfying the normalization condition $|\alpha|^2 + |\beta|^2 = 1$. In matrix form, this state is represented as:

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{3}$$

The corresponding bra vector is:

$$\langle\psi| = \begin{pmatrix} \alpha^* & \beta^* \end{pmatrix} \tag{4}$$

where α^* and β^* are the complex conjugates of α and β, respectively.

The inner product of two state vectors $|\psi\rangle$ and $|\phi\rangle$ is denoted as $\langle\phi|\psi\rangle$ and is calculated as:

$$\langle\phi|\psi\rangle = \alpha_\phi^*\alpha_\psi + \beta_\phi^*\beta_\psi \tag{5}$$

This inner product is essential for calculating probabilities and expectation values in quantum mechanics.

Superposition, a fundamental principle of quantum mechanics, allows a quantum system to exist in a combination of states simultaneously. For a qubit, this means it can be in a state that is neither definitely $|0\rangle$ nor definitely $|1\rangle$, but rather a combination of both. The probabilities of measuring the qubit in each basis state are given by the squared magnitudes of the probability amplitudes: $P(0) = |\alpha|^2$ and $P(1) = |\beta|^2$.

Measurement plays a crucial role in quantum computing. When a qubit in superposition is measured, it collapses to one of the basis states with probabilities determined by the probability amplitudes. This collapse is a fundamental feature of quantum mechanics and has important implications for quantum algorithms.

Transformations on quantum states are represented by unitary operators. A unitary operator U satisfies $U^\dagger U = UU^\dagger = I$, where $U^\dagger$ is the conjugate transpose of U and I is the identity operator. These transformations preserve the normalization of the state vector and the total probability.

Single-qubit gates are unitary operations that act on individual qubits. Some important single-qubit gates include:

- **Pauli gates:** X, Y, and Z gates, represented by the Pauli matrices:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{6}$$

- **Hadamard gate (H):** Creates superposition:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{7}$$

The Hadamard gate is particularly important as it creates an equal superposition of $|0\rangle$ and $|1\rangle$ states when applied to $|0\rangle$:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{8}$$

2.2.2 Multi-Qubit Systems and Entanglement

While single qubits are the fundamental units of quantum information, most practical quantum algorithms, including Shor's algorithm, require multiple qubits working together. The state space of a multi-qubit system is the tensor product of the individual qubit spaces, resulting in a 2^n-dimensional complex vector space for n qubits.

For a system of n qubits, the computational basis states are typically denoted as $|x_1 x_2 ... x_n\rangle$, where each x_i is either 0 or 1. The general state of an n-qubit system can be written as:

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} c_x |x\rangle \tag{9}$$

where c_x are complex amplitudes satisfying the normalization condition $\sum_{x \in \{0,1\}^n} |c_x|^2 = 1$.

One of the most important features of multi-qubit systems is entanglement. Entangled states are quantum states that cannot be factored as tensor products of individual qubit states. For example, the Bell state:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{10}$$

is an entangled two-qubit state. Entanglement allows for correlations between qubits that are stronger than any classical correlation and is a key resource in many quantum algorithms, including Shor's algorithm.

Multi-qubit gates operate on two or more qubits simultaneously. The Controlled-NOT (CNOT) gate is a fundamental two-qubit gate that flips the second qubit (target) if the first qubit (control) is $|1\rangle$:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{11}$$

Quantum circuits are diagrams representing sequences of quantum gates applied to a set of qubits. In these diagrams, horizontal lines represent qubits, boxes on these lines represent single-qubit gates, and vertical lines connecting boxes represent multi-qubit gates. Time flows from left to right in these diagrams.

Measurement in multi-qubit systems can be partial, where only a subset of qubits is measured. This operation collapses the measured qubits but leaves the others in a superposition, conditioned on the measurement outcome.

A set of quantum gates is considered universal if any unitary operation can be approximated to arbitrary accuracy using only gates from this set. Some universal gate sets include H, T, CNOT and H, S, T, CNOT, where S and T are phase gates introducing $\pi/2$ and $\pi/4$ phase shifts, respectively.

In physical quantum computers, these abstract gates and operations are implemented using controlled electromagnetic pulses or laser beams, depending on the specific qubit technology (e.g., superconducting qubits, trapped ions, photonic qubits).

Understanding these quantum computing fundamentals is crucial for grasping the power and potential of quantum algorithms like Shor's algorithm, which leverage quantum superposition, entanglement, and interference to achieve computational speedups over classical algorithms.

3 Shor's Algorithm: An Overview

Shor's algorithm, developed by Peter Shor in 1994, is a quantum algorithm designed to solve the integer factorization problem efficiently. This problem, which involves finding the prime factors of a large composite number, is believed to be computationally difficult for classical computers. The best known classical algorithms for this task have subexponential time complexity, which forms the basis for the security of many cryptographic systems, including RSA.

The significance of Shor's algorithm lies in its ability to factor large integers in polynomial time on a quantum computer, potentially undermining the security of widely used public-key cryptosystems. The algorithm combines principles from quantum mechanics with number theory to achieve this remarkable speedup.

3.1 The Core Idea: Period Finding

At its heart, Shor's algorithm is a period-finding algorithm. The key insight is that the problem of factoring can be reduced to finding the period of a certain function. Specifically, for a number N to be factored, we define the function:

$$f(x) = a^x \mod N \tag{12}$$

where a is a randomly chosen integer coprime to N. This function is periodic, meaning there exists a smallest positive integer r such that:

$$f(x + r) = f(x) \quad \text{for all } x \tag{13}$$

This r is called the period of the function. If we can find this period, we can use it to factor N with high probability using classical post-processing steps.

The period-finding problem is difficult for classical computers but can be solved efficiently using quantum computation. This is where the power of Shor's algorithm lies.

3.2 Quantum Parallelism in Shor's Algorithm

Shor's algorithm leverages quantum parallelism to evaluate the function $f(x)$ for many values of x simultaneously. This is achieved through the following steps:

1. **State Preparation:** The algorithm begins by preparing a quantum state that is a superposition of all possible inputs x. This is typically done using Hadamard gates on a register of n qubits, where n is chosen to be approximately $2\log_2 N$. The resulting state is:

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \tag{14}$$

2. **Function Evaluation:** The function $f(x) = a^x \mod N$ is then applied to this superposition. This is done using a quantum circuit for modular exponentiation, resulting in the state:

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle|f(x)\rangle \tag{15}$$

This step effectively computes $f(x)$ for all 2^n values of x simultaneously, showcasing the power of quantum parallelism.

3. **Quantum Fourier Transform:** The Quantum Fourier Transform (QFT) is then applied to the first register. The QFT is crucial for extracting the periodicity information from the superposition state.

4. **Measurement:** The state is measured, collapsing the superposition. The measurement result, when processed classically, yields information about the period r.

3.3 Key Components of the Algorithm

Shor's algorithm consists of both quantum and classical parts:

1. **Quantum Part:** - State preparation - Modular exponentiation - Quantum Fourier Transform - Measurement

2. **Classical Part:** - Pre-processing (choosing a, checking for trivial factors) - Post-processing of measurement results (continued fraction expansion, checking candidate factors)

The quantum part of the algorithm is where the exponential speedup occurs. It allows us to extract information about the period r in a time that scales polynomially with $\log N$, rather than exponentially as in classical algorithms.

3.4 Complexity and Efficiency

The time complexity of Shor's algorithm is $O((\log N)^3)$, which is polynomial in the number of digits of N. This is exponentially faster than the best known classical factoring algorithms, such as the General Number Field Sieve, which has a subexponential time complexity of $O(e^{((64/9)^{1/3}+o(1))(\ln N)^{1/3}(\ln\ln N)^{2/3}})$.

The space complexity of Shor's algorithm is $O(\log N)$ qubits, which is polynomial in the input size. However, this still represents a significant challenge for current quantum hardware, as factoring even modest-sized numbers requires more coherent qubits than are currently available.

3.5 Probabilistic Nature

It's important to note that Shor's algorithm is probabilistic. It may need to be run multiple times to factor N successfully. However, the probability of success in each run is sufficiently high that the expected number of repetitions is constant, preserving the overall polynomial time complexity.

4 Detailed Explanation of Shor's Algorithm

This section will break down the algorithm into its constituent steps.

4.1 Step 1: Quantum State Preparation

The first step in Shor's algorithm is the preparation of the initial quantum state, which sets up the quantum superposition necessary for parallel evaluation of the modular exponentiation function. This step involves two quantum registers and a series of quantum operations.

Let N be the number we want to factor. We use two quantum registers:

1. An input register with $n = 2\lceil\log_2 N\rceil$ qubits.

2. An output register with $\lceil\log_2 N\rceil$ qubits.

The size of the input register ensures that the period r of the modular exponentiation function (which is at most N) can be represented, and provides sufficient precision for the Quantum Fourier Transform in a later step. The output register can represent numbers from 0 to $N-1$.

Initially, both registers are set to the $|0\rangle$ state. The starting state of the system is thus:

$$|\psi_0\rangle = |0\rangle^{\otimes n} \otimes |0\rangle^{\otimes \lceil \log_2 N \rceil} \tag{16}$$

where $\otimes$ denotes the tensor product.

To create a superposition of all possible input states, we apply a Hadamard gate (H) to each qubit in the input register. The Hadamard gate is defined as:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{17}$$

When applied to the $|0\rangle$ state, it produces:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{18}$$

Applying this operation to all n qubits in the input register results in:

$$(H^{\otimes n} \otimes I)|\psi_0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \otimes |0\rangle \tag{19}$$

where I is the identity operator applied to the output register.

The final state after preparation is:

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \otimes |0\rangle \tag{20}$$

This state represents a uniform superposition of all possible inputs x from 0 to $2^n - 1$ in the input register, while the output register remains in the $|0\rangle$ state.

The creation of this superposition state is crucial for quantum parallelism, allowing simultaneous evaluation of the modular exponentiation function for all possible inputs in the subsequent step. This parallel processing capability is a fundamental advantage over classical computing, where function evaluation would need to be performed sequentially for each input.

Mathematically, this state preparation can be represented as a unitary transformation U_{prep} acting on the initial state:

$$U_{prep} = (H^{\otimes n} \otimes I) \tag{21}$$

such that:

$$|\psi_1\rangle = U_{prep}|\psi_0\rangle \tag{22}$$

This unitary nature ensures that the quantum information is preserved throughout the operation.

It's worth noting that while this state preparation is mathematically straightforward, maintaining such a superposition state in physical quantum systems presents significant challenges due to decoherence effects. The coherence time of the system effectively sets a countdown for the remaining computational steps, emphasizing the need for efficient subsequent operations and robust error correction techniques in practical implementations of Shor's algorithm.

4.2 Step 2: Modular Exponentiation

Building upon the superposition state created in Step 1, the second step of Shor's algorithm involves applying the modular exponentiation function to this state. This step is the computational core of the algorithm and the source of its quantum speedup.

The modular exponentiation function we need to implement is:

$$f(x) = a^x \mod N \tag{23}$$

where N is the number we want to factor, a is a randomly chosen integer coprime to N (1 ¡ a ¡ N), and x is the input represented in the superposition state of the input register.

Our goal is to transform the state $|\psi_1\rangle$ from Step 1 into:

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \otimes |f(x)\rangle \tag{24}$$

This transformation must be unitary to preserve the quantum nature of the computation. We can represent it as a unitary operator U_f such that:

$$U_f(|x\rangle \otimes |y\rangle) = |x\rangle \otimes |y \oplus f(x)\rangle \tag{25}$$

where $\oplus$ denotes bitwise addition modulo 2.

The implementation of U_f leverages the binary decomposition of x:

$$a^x = a^{x_0 \cdot 2^0} \cdot a^{x_1 \cdot 2^1} \cdot a^{x_2 \cdot 2^2} \cdot \ldots \cdot a^{x_{n-1} \cdot 2^{n-1}} \tag{26}$$

where x_i are the binary digits of x. This decomposition allows us to construct U_f as a series of controlled modular multiplications:

$$U_f = \prod_{i=0}^{n-1} (|0\rangle\langle 0|_i \otimes I + |1\rangle\langle 1|_i \otimes M_{a^{2^i} \mod N}) \tag{27}$$

Here, $M_{a^{2^i} \mod N}$ is the unitary operation that performs modular multiplication by $a^{2^i} \mod N$.

Each $M_{a^{2^i} \mod N}$ can be further decomposed into a sequence of modular additions and bit shifts. A common approach is to use the quantum version of the Montgomery multiplication algorithm, which allows for efficient modular multiplication without division.

Mathematically, the Montgomery multiplication for computing $ab \mod N$ can be expressed as:

$$MontMult(a, b, N) = abR^{-1} \mod N \tag{28}$$

where R is a power of 2 greater than N, and R^{-1} is its modular multiplicative inverse modulo N.

The complexity of this step is $O((\log N)^3)$ quantum gates, which dominates the overall complexity of Shor's algorithm. This cubic complexity arises from the need to perform $O(\log N)$ modular multiplications, each requiring $O((\log N)^2)$ operations.

It's crucial to note that while we've described this operation in terms of its action on computational basis states, the power of quantum computation allows it to be applied to the superposition of all possible inputs simultaneously. This parallel processing is what gives Shor's algorithm its advantage over classical factoring algorithms.

The final state after this step encodes the period of the modular exponentiation function in the amplitudes of the superposition:

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \otimes |a^x \mod N\rangle \tag{29}$$

This state preparation sets the stage for the next step of the algorithm, where the Quantum Fourier Transform will be used to extract the period information encoded in this superposition.

4.3 Step 3: Quantum Fourier Transform

Following the modular exponentiation step, Shor's algorithm employs the Quantum Fourier Transform (QFT) to transform the periodicity encoded in the function values into a measurable phase difference. This step is crucial for efficiently extracting the period information.

The QFT is the quantum analogue of the classical discrete Fourier transform. For a quantum state $|x\rangle$ in an n-qubit register, the QFT is defined as:

$$QFT_n|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} e^{2\pi i xy/2^n} |y\rangle \tag{30}$$

The QFT acts linearly on superpositions of states:

$$QFT_n\left(\sum_{x=0}^{2^n-1} c_x|x\rangle\right) = \sum_{y=0}^{2^n-1} \tilde{c}_y|y\rangle \tag{31}$$

where $\tilde{c}_y = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} c_x e^{2\pi i xy/2^n}$.

To understand how and why the QFT works, let's consider its action on a periodic state. Suppose we have a state with period r:

$$|\psi\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |x_0 + kr\rangle \tag{32}$$

where x_0 is some starting point. Applying the QFT to this state gives:

$$QFT_n|\psi\rangle = \frac{1}{\sqrt{r2^n}} \sum_{y=0}^{2^n-1} \sum_{k=0}^{r-1} e^{2\pi i(x_0+kr)y/2^n} |y\rangle \tag{33}$$

The key insight is that this sum will have large amplitudes for values of y that are close to multiples of $2^n/r$. To see why, let's factor out the x_0 term:

$$QFT_n|\psi\rangle = \frac{1}{\sqrt{r2^n}} \sum_{y=0}^{2^n-1} e^{2\pi i x_0 y/2^n} \left(\sum_{k=0}^{r-1} e^{2\pi i k r y/2^n} \right) |y\rangle \tag{34}$$

The sum in parentheses is a geometric series. It will be large when $ry/2^n$ is close to an integer, i.e., when y is close to a multiple of $2^n/r$. Specifically, this sum equals r when y is an exact multiple of $2^n/r$, and is much smaller for other values of y.

This property of the QFT is what allows us to extract the period r. After applying the QFT, measuring the state will yield a value of y that is likely to be close to a multiple of $2^n/r$. From this measurement, we can deduce r with high probability.

In Shor's algorithm, we apply the QFT to the input register of our state after modular exponentiation:

$$|\psi_3\rangle = (QFT_n \otimes I)|\psi_2\rangle \tag{35}$$

Expanding this out, we get:

$$|\psi_3\rangle = \frac{1}{2^n} \sum_{y=0}^{2^n-1} \sum_{x=0}^{2^n-1} e^{2\pi i x y/2^n} |y\rangle \otimes |a^x \mod N\rangle \tag{36}$$

The QFT can be implemented efficiently using a quantum circuit consisting of Hadamard gates and controlled rotation gates. For an n-qubit register, the circuit requires n Hadamard gates and $n(n-1)/2$ controlled rotation gates. The controlled rotation gates are of the form:

$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix} \tag{37}$$

The unitary matrix representing the QFT on n qubits can be expressed as:

$$U_{QFT} = \frac{1}{\sqrt{2^n}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{2^n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(2^n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{2^n-1} & \omega^{2(2^n-1)} & \cdots & \omega^{(2^n-1)(2^n-1)} \end{pmatrix} \tag{38}$$

where $\omega = e^{2\pi i/2^n}$ is the 2^n-th root of unity.

The QFT can be implemented using $O(n^2)$ gates, where n is the number of qubits. This is exponentially faster than the classical Fast Fourier Transform, which requires $O(n2^n)$ gates. In practice, an approximate version of the QFT is often used, where rotations by very small angles are omitted. This reduces the number of gates required while maintaining sufficient accuracy for Shor's algorithm.

The QFT is crucial in Shor's algorithm as it transforms the periodic structure in the function values into a form that can be efficiently measured. It allows us to extract the period r with high probability using only a single measurement, rather than requiring many repeated function evaluations as would be necessary classically. This transformation of the periodic structure into a measurable phase difference is what enables Shor's algorithm to achieve its exponential speedup over classical period-finding methods.

4.4 Step 4: Measurement and Classical Post-processing

The final step of Shor's algorithm involves measuring the quantum state and performing classical post-processing on the measurement results to determine the factors of N. This step translates the quantum information encoded in the superposition state into classical information that can be used to factor N.

After applying the QFT, the state of our quantum system is:

$$|\psi_3\rangle = \frac{1}{2^n} \sum_{y=0}^{2^n-1} \sum_{x=0}^{2^n-1} e^{2\pi i xy/2^n} |y\rangle \otimes |a^x \mod N\rangle \tag{39}$$

When we measure the input register, we obtain a value y with probability:

$$P(y) = \left| \frac{1}{2^n} \sum_{x=0}^{2^n-1} e^{2\pi i xy/2^n} \right|^2 \tag{40}$$

Due to the periodicity of the function $f(x) = a^x \mod N$, this probability distribution has peaks at values of y that are close to multiples of $2^n/r$, where r is the period we're trying to find.

The classical post-processing steps following the measurement are crucial for extracting the period r and using it to factor N. These steps involve number theory techniques and can be summarized as follows:

1. Continued Fraction Expansion: We use the continued fraction expansion of $y/2^n$ to find a rational approximation d/r' where r' is a candidate for the

period r. We look for a fraction where $r' < N$. The continued fraction expansion can be computed using the following recurrence relations:

$$a_0 = \lfloor y/2^n \rfloor \tag{41}$$
$$p_0 = a_0, \quad q_0 = 1 \tag{42}$$
$$p_1 = a_0 a_1 + 1, \quad q_1 = a_1 \tag{43}$$
$$p_i = a_i p_{i-1} + p_{i-2} \tag{44}$$
$$q_i = a_i q_{i-1} + q_{i-2} \tag{45}$$

where a_i are the quotients in the continued fraction expansion.

2. Check Candidate Period: For each candidate period r', we check if:

$$a^{r'} \equiv 1 \pmod{N} \tag{46}$$

If this condition is met, r' is likely to be the actual period r or a multiple of it.

3. Factor Finding: If we find a suitable r', we compute:

$$\gcd(a^{r'/2} \pm 1, N) \tag{47}$$

If r' is even and this gcd is nontrivial (not 1 or N), it gives us a factor of N. This step is based on the fact that if r is the period of $a^x \mod N$, then:

$$a^r - 1 = (a^{r/2} - 1)(a^{r/2} + 1) \equiv 0 \pmod{N} \tag{48}$$

Therefore, if r is even, N must divide $(a^{r/2} - 1)(a^{r/2} + 1)$, but it's unlikely to divide both factors, so one of the gcd computations is likely to yield a non-trivial factor.

The probability of measuring a y that leads to the correct period r is at least:

$$P(\text{success}) \geq \frac{4}{\pi^2} \approx 0.405 \tag{49}$$

This means that, on average, we need to repeat the quantum part of Shor's algorithm only a constant number of times to find the period with high probability.

If the post-processing fails to find a factor (which can happen due to the probabilistic nature of the algorithm), we repeat the entire process with a new random a. The classical post-processing steps have polynomial time complexity:

- Continued fraction expansion: $O((\log N)^2)$ - Modular exponentiation check: $O((\log N)^3)$ - GCD computation: $O((\log N)^3)$

These are dominated by the $O((\log N)^3)$ complexity of the quantum part of the algorithm.

To illustrate this process, consider factoring $N = 15$. Suppose we measure $y = 2^{13}$ in a 16-qubit input register. Then:

$$\frac{y}{2^n} = \frac{2^{13}}{2^{16}} = \frac{1}{8} \tag{50}$$

This suggests a period of 8. We check $a^8 \equiv 1 \pmod{15}$. If this holds, we compute $\gcd(a^4 \pm 1, 15)$, which may give us a factor of 15.

The efficiency of this measurement and post-processing step, combined with the quantum speedup in evaluating the modular exponentiation function, is what gives Shor's algorithm its power to factor large numbers efficiently. The algorithm transforms the problem of factoring, which is believed to be computationally hard for classical computers, into the problem of period-finding, which can be solved efficiently on a quantum computer.

5 Complexity Analysis

The efficiency of Shor's algorithm is one of its most remarkable features, offering a significant speedup over classical factoring methods. This section provides a detailed analysis of the algorithm's time and space complexity, comparing it with classical factoring algorithms and exploring its probabilistic nature.

5.1 Time Complexity

The time complexity of Shor's algorithm is dominated by three main components: the Quantum Fourier Transform (QFT), modular exponentiation, and classical post-processing. Let's examine each of these components in detail.

The QFT on an n-qubit register can be implemented using $O(n^2)$ quantum gates. In Shor's algorithm, we use $n = 2 \log N$ qubits, resulting in a complexity of $O((\log N)^2)$ for the QFT step.

The modular exponentiation step, which computes $a^x \mod N$ for superpositions of x, is the most time-consuming part of the algorithm. Using fast multiplication techniques, this step can be implemented with $O((\log N)^3)$ quantum gates. The complexity of this step dominates the overall time complexity of the algorithm.

The classical part of the algorithm, including continued fraction expansion and greatest common divisor computations, has a time complexity of $O((\log N)^3)$. This includes the post-processing of measurement results to extract the period and compute the factors.

Combining these components, the overall time complexity of Shor's algorithm is:

$$T(N) = O((\log N)^2 + (\log N)^3 + (\log N)^3) = O((\log N)^3) \tag{51}$$

This polynomial time complexity represents an exponential speedup over the best known classical factoring algorithms. To put this into perspective, let's compare it with the General Number Field Sieve (GNFS), the most efficient classical factoring algorithm for large numbers.

The GNFS has a subexponential time complexity of:

$$T_{GNFS}(N) = O(e^{((64/9)^{1/3}+o(1))(\ln N)^{1/3}(\ln \ln N)^{2/3}}) \tag{52}$$

To illustrate the dramatic difference, consider factoring a 2048-bit number, which is common in RSA cryptography. Shor's algorithm would require approximately $(\log_2 2048)^3 \approx 1.7 \times 10^6$ operations, while the GNFS would require approximately 2^{112} operations. This latter figure is infeasible with current and foreseeable classical computing power.

5.2 Space Complexity and Probabilistic Nature

The space complexity of Shor's algorithm is determined by the number of qubits required. The algorithm needs $2\log N$ qubits for the input register, $\log N$ qubits for the output register, and a small number of additional qubits for the modular exponentiation circuit. Therefore, the total number of qubits required is $O(\log N)$.

While this space complexity is polynomial in the input size, it still presents a significant challenge for current quantum hardware. For instance, factoring a 2048-bit RSA key would require approximately 4096 qubits for the input register alone, which exceeds the capabilities of current quantum computers.

It's crucial to note that Shor's algorithm is probabilistic in nature. Each run of the algorithm has a probability of at least $4/\pi^2 \approx 0.405$ of succeeding. This probability arises from the quantum measurement process and the subsequent classical post-processing.

If we denote the probability of failure in a single run as p, then the probability of failure after k runs is p^k. To achieve a success probability of at least $1 - \epsilon$, we need to run the algorithm k times, where:

$$k \geq \frac{\log \epsilon}{\log p} \tag{53}$$

Given that $p \leq 1 - 4/\pi^2$, we can ensure a high probability of success with a constant number of repetitions, independent of the size of N. This means that the probabilistic nature of the algorithm does not affect its overall polynomial time complexity.

The algorithm's efficiency stems from its clever use of quantum superposition and the quantum Fourier transform to find the period of the modular exponentiation function. This period-finding is at the heart of the algorithm's ability to factor large numbers efficiently.

While the asymptotic complexity of Shor's algorithm is impressive, practical implementation faces several challenges. These include the need for quantum error correction, limited coherence times of qubits, and the high fidelity requirements for quantum gates. These factors make the implementation of Shor's algorithm for large numbers a significant engineering challenge, despite its theoretical efficiency.

In conclusion, the complexity analysis of Shor's algorithm reveals its potential to revolutionize factoring and, by extension, cryptography. Its polynomial time complexity offers an exponential speedup over classical methods, while its space complexity, though challenging for current technology, remains polynomial in the input size. The probabilistic nature of the algorithm, far from being a drawback, is managed efficiently to ensure a high probability of success with minimal impact on the overall complexity.

6 Example: Factoring a Small Number

To illustrate the workings of Shor's algorithm, let's walk through a concrete example of factoring a small number. We'll factor N = 15, which, while trivial for classical methods, serves as a good demonstration of the quantum algorithm's steps.

6.1 Setup and Initial Steps

Our goal is to factor N = 15. We'll use a = 7 as our randomly chosen number coprime to N.

6.1.1 Quantum Register Sizes

We need:

- Input register: $2\lceil \log_2 15 \rceil = 8$ qubits
- Output register: $\lceil \log_2 15 \rceil = 4$ qubits

6.1.2 Initial State Preparation

We start with all qubits in the $|0\rangle$ state:

$$|\psi_0\rangle = |0\rangle^{\otimes 8} \otimes |0\rangle^{\otimes 4} \tag{54}$$

Apply Hadamard gates to the input register:

$$|\psi_1\rangle = \frac{1}{\sqrt{2^8}} \sum_{x=0}^{255} |x\rangle \otimes |0\rangle \tag{55}$$

6.2 Quantum Computation Steps

6.2.1 Modular Exponentiation

We apply the function $f(x) = 7^x \mod 15$ to our state:

$$|\psi_2\rangle = \frac{1}{\sqrt{2^8}} \sum_{x=0}^{255} |x\rangle \otimes |7^x \mod 15\rangle \tag{56}$$

The period of this function is actually 4, as we can see from the sequence:

$$\begin{gathered}7^0 \equiv 1 \pmod{15}\\ 7^1 \equiv 7 \pmod{15}\\ 7^2 \equiv 4 \pmod{15}\\ 7^3 \equiv 13 \pmod{15}\\ 7^4 \equiv 1 \pmod{15} \text{ (cycle repeats)}\end{gathered}$$

6.2.2 Quantum Fourier Transform

Apply QFT to the input register:

$$|\psi_3\rangle = \frac{1}{2^8}\sum_{y=0}^{255}\sum_{x=0}^{255} e^{2\pi i x y/2^8}|y\rangle \otimes |7^x \mod 15\rangle \tag{57}$$

6.3 Measurement and Classical Post-processing

6.3.1 Measurement

We measure the input register. Let's say we obtain y = 64.

6.3.2 Continued Fraction Expansion

We compute the continued fraction expansion of $\frac{64}{256} = \frac{1}{4}$.

This immediately gives us the candidate period r' = 4.

6.3.3 Verify the Period

We check if $7^4 \equiv 1 \pmod{15}$, which is true.

6.3.4 Find Factors

We compute:

$$\gcd(7^{4/2} - 1, 15) = \gcd(48, 15) = 3 \tag{58}$$

$$\gcd(7^{4/2} + 1, 15) = \gcd(50, 15) = 5 \tag{59}$$

6.4 Result

We have successfully factored 15 into 3 and 5 using Shor's algorithm.

6.5 Observations

- In this simple example, we were fortunate to measure a y that immediately gave us the correct period. In practice, we might need to repeat the quantum part of the algorithm several times.

- For such a small number, the quantum algorithm is overkill. Classical methods would be much faster. The power of Shor's algorithm becomes apparent only for very large numbers.
- In a real implementation, we would need to account for quantum errors and the limitations of actual quantum hardware, which we've ignored in this idealized example.

This example demonstrates the basic flow of Shor's algorithm, from quantum state preparation through measurement and classical post-processing, resulting in the successful factorization of our target number.

7 Implications for Cryptography

Shor's algorithm represents a paradigm shift in the field of cryptography, with its potential to efficiently factor large numbers posing a significant threat to many widely used cryptographic systems. This section explores the mathematical foundations of these systems, how Shor's algorithm undermines them, and the emerging field of post-quantum cryptography.

The security of many public-key cryptosystems, including RSA, ElGamal, and elliptic curve cryptography, relies on the presumed difficulty of certain mathematical problems. RSA, for instance, bases its security on the difficulty of factoring the product of two large prime numbers. The time complexity for factoring an n-bit RSA modulus using the best known classical algorithms, such as the General Number Field Sieve (GNFS), is:

$$T_{classical} = O(e^{((64/9)^{1/3}+o(1))(n\ln 2)^{1/3}(\ln(n\ln 2))^{2/3}}) \tag{60}$$

This subexponential complexity has long been considered sufficient for practical security. However, Shor's algorithm can factor the same number in polynomial time:

$$T_{Shor} = O(n^3) \tag{61}$$

This exponential speedup is not limited to integer factorization. Shor's algorithm can also efficiently solve the discrete logarithm problem, which underpins the security of systems like ElGamal and DSA. For a cyclic group of order n, classical algorithms like the baby-step giant-step algorithm have a time complexity of $O(\sqrt{n})$, while Shor's algorithm achieves $O((\log n)^3)$.

Elliptic curve cryptography (ECC), which relies on the difficulty of the elliptic curve discrete logarithm problem (ECDLP), is similarly vulnerable. While the best classical algorithms for ECDLP have a complexity of $O(\sqrt{n})$, where n is the order of the elliptic curve group, Shor's algorithm solves ECDLP in $O((\log n)^3)$ time.

It's worth noting that symmetric key cryptography, such as AES, is less affected by quantum computing. Grover's algorithm provides a quadratic speedup for brute-force attacks on symmetric ciphers, reducing the effective key strength by half. However, this can be mitigated by doubling the key size. The time complexity for a brute-force attack on a symmetric cipher with key size n becomes:

$$T_{\text{Grover}} = O(2^{n/2}) \tag{62}$$

compared to the classical $O(2^n)$.

The threat posed by quantum computers has spurred the development of post-quantum cryptography (PQC), a field dedicated to creating cryptographic systems secure against both quantum and classical computers. Several mathematical approaches are being explored:

1. Lattice-based cryptography relies on the difficulty of certain problems in lattice theory, such as the shortest vector problem (SVP) and the closest vector problem (CVP). These problems are believed to be hard even for quantum computers. The security of lattice-based systems often relies on the hardness of the Learning With Errors (LWE) problem or its ring-based variant (Ring-LWE).

2. Hash-based signatures, such as SPHINCS+, offer another quantum-resistant alternative. These schemes rely on the security of their underlying hash functions and the difficulty of finding collisions or preimages. The security of these systems can be expressed in terms of the collision resistance of the hash function:

$$P_{collision} \approx \frac{q^2}{2^n} \tag{63}$$

where q is the number of queries an attacker can make, and n is the output size of the hash function.

3. Code-based cryptography, like the McEliece system, bases its security on the difficulty of decoding a general linear code. The best known algorithms for decoding random linear codes have a time complexity of:

$$T_{decode} = O(2^{0.0885n}) \tag{64}$$

where n is the code length.

4. Multivariate cryptography relies on the difficulty of solving systems of multivariate polynomial equations over finite fields. The general problem of solving such systems is NP-hard, with the best algorithms having exponential complexity in the number of variables.

The transition to post-quantum cryptography presents several challenges, including performance issues (many PQC algorithms have larger key sizes or slower performance), implementation difficulties, and the need for extensive scrutiny to build confidence in their security.

While the threat from quantum computers is not immediate due to current hardware limitations, the cryptographic community is actively working to prepare for a post-quantum future. The goal is to develop and deploy quantum-resistant cryptographic systems before large-scale quantum computers become a reality, ensuring the continued security of our digital communications and transactions in the quantum era.

The U.S. National Institute of Standards and Technology (NIST) is currently in the process of standardizing post-quantum cryptographic algorithms, aiming to identify and standardize one or more quantum-resistant public-key cryptographic algorithms. This process involves rigorous mathematical analysis and testing to ensure the selected algorithms provide the necessary security guarantees in a post-quantum world.

In conclusion, Shor's algorithm has profound implications for cryptography, necessitating a fundamental shift in our approach to secure communication. The mathematical challenges posed by this quantum algorithm have sparked innovative research in cryptography, leading to the development of new mathematical techniques and problems that we believe will remain hard even in the face of quantum computation. As we transition into this new era of cryptography, the interplay between quantum computing, number theory, and computational complexity will continue to drive advancements in the field.

8 Conclusion

Shor's algorithm stands as a landmark achievement in the field of quantum computing, demonstrating the potential for quantum algorithms to solve problems that are intractable for classical computers. Through this paper, we have explored the algorithm's structure, its mathematical foundations, and its far-reaching implications.

8.1 Summary of Key Points

- Shor's algorithm provides a quantum method for factoring large integers in polynomial time, a task believed to be exponentially difficult for classical computers.
- The algorithm's power stems from its clever use of quantum parallelism and the quantum Fourier transform to extract the period of a modular exponentiation function.
- With a time complexity of $O((\log N)^3)$, Shor's algorithm offers an exponential speedup over the best known classical factoring algorithms.
- The algorithm's potential to break widely used public-key cryptosystems, particularly RSA, has profound implications for cybersecurity and has spurred the development of post-quantum cryptography.

8.2 Significance and Impact

The significance of Shor's algorithm extends beyond its immediate application to factoring. It serves as a proof of concept, demonstrating that quantum computers can indeed solve certain problems exponentially faster than classical computers. This has energized the field of quantum computing, driving both theoretical research and practical efforts to build large-scale quantum computers.

In the realm of cryptography, Shor's algorithm has initiated a paradigm shift. It has highlighted the vulnerability of many current cryptographic systems to quantum attacks and has catalyzed the development of quantum-resistant cryptographic methods. This ongoing work in post-quantum cryptography is crucial for ensuring the security of digital communications in a future where powerful quantum computers exist.

8.3 Future Outlook

While the full realization of Shor's algorithm on a scale that threatens current cryptographic systems remains a future prospect, the algorithm continues to play a central role in quantum computing research and development. As quantum hardware advances, we can expect to see implementations of Shor's algorithm on increasingly larger numbers, serving as important milestones in the progress of quantum computing.

The journey from Shor's theoretical breakthrough to practical implementation highlights the interdisciplinary nature of quantum computing, involving physics, computer science, mathematics, and engineering. Overcoming challenges such as quantum error correction and maintaining quantum coherence will be crucial in bringing the full power of Shor's algorithm to bear.

8.4 Broader Implications

Beyond its specific application to factoring and cryptography, Shor's algorithm exemplifies the potential of quantum computing to revolutionize computational approaches across various fields. It encourages us to reconsider problems currently deemed intractable and to explore new algorithmic possibilities offered by quantum systems.

As we stand on the brink of the quantum computing era, Shor's algorithm remains a shining example of the transformative potential of this technology. It continues to inspire researchers, drive technological development, and shape our preparation for a future where quantum computers are a reality. The story of Shor's algorithm is far from over; indeed, it may be just the beginning of a new chapter in the history of computation.

References

[1] Ilyas Khan and Jenni Strabley. Quantinuum extends its significant lead in quantum computing, achieving historic milestones for hardware fidelity and quantum volume, 4 2024. Accessed on July 14, 2024.

[2] Eleanor Rieffel and Wolfgang Polak. *Quantum Computing: A Gentle Introduction.* MIT Press, Cambridge, MA, 2011.

[3] David Warburton and Sander Vinberg. The 2021 tls telemetry report, October 2021. Accessed on July 14, 2024.