Security Implications of Quantum Computing on Cryptography

An Independent Study
Submitted to the Faculty of
The Department of Electrical and Computer Engineering
Villanova University

By

Jeffrey A. King

In Partial Fulfillment
Of the Requirements for the Degree of
Master of Science in Computer Engineering

VILLANOVA
UNIVERSITY

May 7, 2020

Abstract

In recent years, there has been significant progress in the development of quantum computers, which perform computations based on the principles of quantum mechanics. Quantum algorithms are much different in nature than classical algorithms, but they are known to provide speedup over classical algorithms for certain problems. Currently, two of the best-known quantum algorithms are Grover's algorithm for unstructured searches, and Shor's algorithm for factoring numbers. These algorithms are known to pose a threat to currently used cryptographic systems, but there are currently no quantum computers powerful enough to execute them. If a large-scale quantum computer is ever built, many public key cryptography schemes would be broken by Shor's algorithm. For this reason, research is being done in the area of post-quantum cryptography. Secret key cryptography schemes will fare better, as the threat posed by Grover's algorithm is easily mitigated by doubling the key length.

Contents

Abstract **iv**

1 Introduction **1**

2 Quantum Computing Principles **2**

3 Attacks on Secret Key Cryptography **10**

4 Attacks on Public Key Cryptography **20**

5 Post-Quantum Cryptography **27**

6 Future Work **28**

7 Conclusion **29**

References **30**

Appendix A – Listing of grover.m **32**

Appendix B – Listing of groveropt.m **33**

Appendix C – Listing of shor.m **34**

List of Tables

3.1	Grover's algorithm, N=8	16
3.2	Grover's algorithm, N=16	16
3.3	Grover's algorithm, N=32	17
4.1	Values of a^x mod 33	23

List of Figures

2.1	The BB84 quantum key distribution protocol	5
2.2	Single-qubit gates	6
2.3	Multi-qubit gates	7
3.1	Grover's algorithm sign flip	12
3.2	Inversion about the average	12
3.3	Grover's algorithm *D* circuit	13
3.4	Summary of Grover's algorithm	14
3.5	Grover's algorithm state for N=65536; initial, after 100 iterations, after 200 iterations	15
3.6	Grover's algorithm state for N=65536 after 250 iterations	15
3.7	Amplitude of desired state vs. number of iterations, N=8	16
3.8	Amplitude of desired state vs. number of iterations, N=16	17
3.9	Amplitude of desired state vs. number of iterations, N=32	18
3.10	Output of grover(4,4)	18
3.11	Output of groveropt(16)	19
4.1	Output of shor(33,5)	25

Chapter 1

Introduction

In the early 1980's, Richard Feynman described the idea of a computer that uses the principles of quantum mechanics to perform computations, after concluding that it was not possible to simulate quantum mechanics on a classical computer [10]. In the time since then, the theory of quantum computing was developed. A well-designed quantum algorithm can provide significant speedup when compared to a corresponding classical algorithm. One well-known example is Shor's algorithm, which can factor numbers in polynomial time [8]. The problem of factoring numbers is considered so difficult that the security of the RSA encryption algorithm is based on that problem. If a large-scale quantum computer powerful enough to run Shor's algorithm is ever built, RSA will be broken.

For a long time, the field of quantum computing was purely theoretical. It was known how the principles of quantum mechanics could be used to perform computations, but it was not known how to build a quantum computer. However, the situation is different today. Currently, there are commercially available quantum computers, such as IBM's Quantum Experience [14]. There has been noticeable progress in the development of quantum computers, and architectures that have potential to scale up to larger systems have been demonstrated [15]. As the capabilities of quantum computers scale up, they will eventually pose a real threat to currently used cryptographic systems.

This report is a study of the ways that quantum computers could be used to attack present-day cryptographic systems. The remainder of the report is organized as follows. Chapter 2 explains some fundamental principles of quantum computing. Next, Chapter 3 describes the use of quantum algorithms to attack secret key cryptography. Then, Chapter 4 explains how quantum algorithms can be used to attack public key cryptography. Next, Chapter 5 provides a survey of post-quantum cryptography, which aims to create cryptographic algorithms that are secure against attacks by quantum computers. Chapter 6 looks at future work that could be done for further study. Finally, Chapter 7 concludes the report.

Chapter 2

Quantum Computing Principles

2.1 Qubits

The goal of quantum mechanics is to find the wave function $\Psi(x,t)$ of a particle, which can be obtained by solving the Schrödinger equation [1]:

$$i\hbar\frac{\partial\Psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\Psi}{\partial x^2} + V\Psi$$

In this equation, $\hbar$ is a constant known as Planck's constant, m is the mass of the particle, and V is the potential energy function. For the study of quantum computing, we will not examine the details of the wave functions, but we will represent a particle's wave function, also known as its state, as a normalized vector, $|\psi\rangle$ [1]. For a given system, there can be multiple solutions to the Schrödinger equation, and the state can be a linear combination of these solutions, since a linear combination of solutions to the Schrödinger equation is also a solution [1].

The simplest system we will consider is a quantum bit, or qubit. A qubit has two states that form an orthonormal basis for a two-dimensional state space [2]. We will label these basis states $|0\rangle$ and $|1\rangle$, and refer to them as the standard basis. Unlike a classical bit, a qubit can be not only 0 or 1, but any combination of 0 and 1. This concept, which is known as superposition, is very important in quantum computing algorithms. The general form of a qubit's state is

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

where α and β are complex numbers. The state is always a unit vector, therefore,

$$|\alpha|^2 + |\beta|^2 = 1.$$

The state can also be represented as follows [5]:

$$|\psi\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix}$$

Obviously, the top element is the $|0\rangle$ component, and the bottom element is the $|1\rangle$ component. The basis vectors can be represented as

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix};\ |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}.$$

The representation $|\psi\rangle$ is known as a "ket," while the conjugate transpose is represented by $\langle\psi|$, and is known as a "bra." This notation is known as "bra-ket" notation. $\langle\psi|$ can be represented as

$$\langle\psi| = [\alpha^* \quad \beta^*].$$

An expression such as $\langle\psi|\phi\rangle$ is called an inner product [5]. If all the components of the two vectors are real numbers, the inner product is the same as the dot product. The inner product of a unit vector and itself is always 1:

$$\langle\psi|\psi\rangle = [\alpha^* \quad \beta^*]\begin{bmatrix}\alpha\\\beta\end{bmatrix} = \alpha^*\alpha + \beta^*\beta = |\alpha|^2 + |\beta|^2 = 1$$

The inner product of two orthogonal unit vectors is always zero. Concerning the basis states, we see that

$$\langle 0|0\rangle = [1 \quad 0]\begin{bmatrix}1\\0\end{bmatrix} = 1;\ \langle 1|1\rangle = [0 \quad 1]\begin{bmatrix}0\\1\end{bmatrix} = 1;\ \langle 1|0\rangle = [0 \quad 1]\begin{bmatrix}1\\0\end{bmatrix} = 0;\ \langle 0|1\rangle = [1 \quad 0]\begin{bmatrix}0\\1\end{bmatrix} = 0.$$

2.2 Quantum Measurement

Because a qubit is represented by two complex numbers, it can represent an infinite number of different states [4]. A classical bit can only represent two states: 0 and 1. However, the laws of quantum mechanics do not allow us to view the entire quantum state. When a qubit is measured, one of two possible outcomes will be observed. Therefore, we can only extract a classical bit's worth of information from a qubit [4]. In most of the examples in this report, qubits will be measured in the standard basis. When a qubit $\alpha|0\rangle + \beta|1\rangle$ is measured in the standard basis, $|0\rangle$ will be observed with probability $|\alpha|^2$, while $|1\rangle$ will be observed with probability $|\beta|^2$. Furthermore, the act of measurement changes the state to the basis state that was observed [4]. So, the state is now either $|0\rangle$ or $|1\rangle$, and the original state is no longer recoverable. However, if the original state was one of the basis states, the state is unchanged by the measurement. Formally, each basis vector has an operator known as a projector [3], which is given by the outer product of the basis vector with itself:

$$P_0 = |0\rangle\langle 0| = \begin{bmatrix}1\\0\end{bmatrix}[1 \quad 0] = \begin{bmatrix}1 & 0\\0 & 0\end{bmatrix};\ P_1 = |1\rangle\langle 1| = \begin{bmatrix}0\\1\end{bmatrix}[0 \quad 1] = \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}$$

A measurement of $|\psi\rangle$ changes the state to

$$\frac{P_i|\psi\rangle}{|P_i|\psi\rangle|}$$

with probability $|P_i|\psi\rangle|^2$ [3].

It is possible to measure a qubit in a basis other than the standard basis. For example, consider the basis vectors

$$|+\rangle = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}; \; |-\rangle = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$$

The projectors for these basis vectors are

$$P_+ = |+\rangle\langle+| = \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix}; \; P_- = |-\rangle\langle-| = \begin{bmatrix} 1/2 & -1/2 \\ -1/2 & 1/2 \end{bmatrix}$$

If $|0\rangle$ is measured in the standard basis, the outcome will always be $|0\rangle$. But, in this basis, the outcome will be either $|+\rangle$ or $|-\rangle$, each with 50% probability.

2.3 Quantum Key Distribution

The principles of the previous example have been applied to create a quantum key distribution protocol. One such protocol, named BB84 after its inventors (Charles Bennett and Giles Brassard) and the year it was invented [3], is explained in [4] as follows: Alice and Bob, who wish to exchange a secret key, are connected by a bidirectional classical channel and a unidirectional quantum channel. Alice can use the quantum channel to send particles to Bob, who measures the state of each particle. Alice encodes a bit of the key in each particle by randomly choosing one of two bases to encode each bit: $0 = |\rightarrow\rangle; 1 = |\uparrow\rangle$, or $0 = |\nwarrow\rangle; 1 = |\nearrow\rangle$. Using the notation from the previous example, the bases could also be represented as $0 = |0\rangle; 1 = |1\rangle$, $0 = |+\rangle; 1 = |-\rangle$. For each particle that Bob receives, he randomly chooses one of the two bases to measure the particle. If Alice and Bob chose the same basis for a given particle, Bob is guaranteed to have measured the state exactly as Alice sent it, and that bit can be used for the key. After Alice has sent all the bits, she and Bob share the bases that they chose over the classical channel. They can use this information to determine which bits were sent correctly and can be used in the key. However, a third party cannot determine the values of the bits from this information. If a third party, Eve, measures a particle sent by Alice and then resends it to Bob, she will measure in the wrong basis, and change the particle's state, on average 50% of the time. As a result, if Bob measures in the correct basis (the basis Alice chose), there is a 25% chance the value he measured is incorrect (50% chance Eve picked the wrong basis x 50% chance of Bob measuring the wrong value). If Alice and Bob are sending parity bits over the classical channel, they will detect a high error rate, and conclude that somebody is eavesdropping on them. The BB84 protocol is illustrated in Figure 2.1:

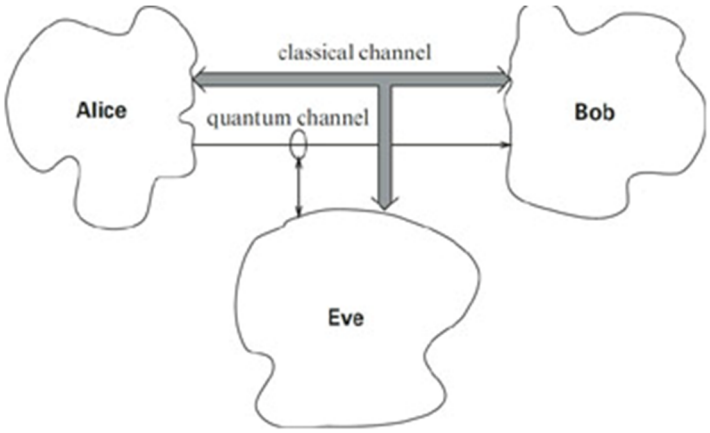

Figure 2.1: The BB84 quantum key distribution protocol [3].

2.4 Multiple Qubits and Entanglement

In a system with multiple qubits, qubits are combined using the tensor product [4]:

$$(a_1|0\rangle + b_1|1\rangle)\otimes(a_2|0\rangle + b_2|1\rangle) = a_1a_2|00\rangle + a_1b_2|01\rangle + b_1a_2|10\rangle + b_1b_2|11\rangle$$

For two qubits, the basis states are $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$. (Depending on the situation, we may choose to label these $|0\rangle$, $|1\rangle$, $|2\rangle$, and $|3\rangle$ instead.) When written in vector form, by convention, the components are listed from top to bottom (or left to right) starting with the lowest-numbered basis state and increasing. For a system of n qubits, there are 2^n basis states. Therefore, as the number of qubits increases, the number of parameters needed to describe the system increases exponentially.

The state shown above is known as a product state [5], since it can be expressed as a tensor product of the individual qubits. However, it is also possible to have a state that cannot be expressed in terms of its individual qubits; an example is $(1/\sqrt{2})(|00\rangle + |11\rangle)$ [4]. Such a state is said to be entangled. The concept of entanglement has no classical counterpart [4].

The previously discussed concepts of measurement extend to multi-qubit systems [3]. For example, if we measure the first qubit of the state

$$a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle$$

in the standard basis, the projectors are

$$P_0 = |00\rangle\langle 00| + |01\rangle\langle 01| = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix};\ P_1 = |10\rangle\langle 10| + |11\rangle\langle 11| = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

After measurement, the state will be either

$$\frac{a|00\rangle + b|01\rangle}{\sqrt{|a|^2 + |b|^2}}$$

or

$$\frac{c|10\rangle + d|11\rangle}{\sqrt{|c|^2 + |d|^2}}$$

with probabilities of $|a|^2 + |b|^2$ and $|c|^2 + |d|^2$, respectively. If we measure both qubits in the standard basis, we will get one of the four basis vectors with probabilities $|a|^2$, $|b|^2$, $|c|^2$, and $|d|^2$.

2.5 Quantum Circuits

We will now look at operations that can be performed on qubits to perform computations. The laws of quantum mechanics dictate that operations performed on a qubit must be unitary [2], and it follows from this stipulation that operations must also be reversible [4]. This means that the transformation performed by a quantum gate can be described as a unitary matrix. In theory, any arbitrary unitary matrix could be a quantum operation. However, in practice, not every unitary operation can be implemented efficiently, as we will see later. But first, we will look at quantum gates that operate on only one qubit. Figure 2.2 shows several common single-qubit gates and their associated matrices:

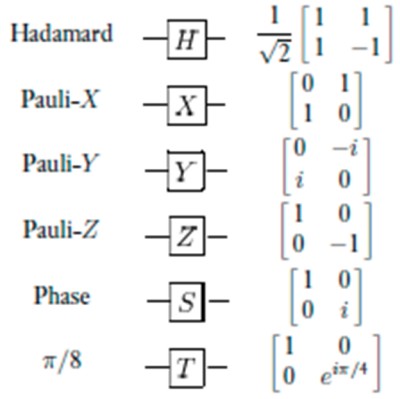

Figure 2.2: Single-qubit gates [2].

The Hadamard gate transforms $|0\rangle$ to $(1/\sqrt{2})(|0\rangle + |1\rangle)$ and $|1\rangle$ to $(1/\sqrt{2})(|0\rangle - |1\rangle)$. The state $(1/\sqrt{2})(|0\rangle + |1\rangle)$ is a superposition of $|0\rangle$ and $|1\rangle$. If we apply a Hadamard gate to every qubit in an *n*qubit register, and all the qubits are $|0\rangle$, the register will be in a superposition of all the basis vectors:

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

This is very powerful, because it enables a quantum circuit to perform computations on all basis vectors at the same time, and the output is a superposition of the results. This is known as quantum parallelism [4]. For that reason, this is the first step in many quantum algorithms [5].

The Pauli-X gate acts like a NOT, flipping $|0\rangle$ to $|1\rangle$ and vice versa [5]. The Pauli-Y gate transforms $|0\rangle$ to $i|1\rangle$ and $|1\rangle$ to $-i|0\rangle$. The Pauli-Z gate transforms $|1\rangle$ to $-|1\rangle$ and does not change $|0\rangle$ [5]. This gate is also known as a "phase flip" gate [5], since it shifts the phase of $|1\rangle$ by π ($e^{i\pi} = -1$). The phase gate transforms $|1\rangle$ to $i|1\rangle$, which is a phase shift of $\pi/2$ ($e^{i\pi/2} = i$). The $\pi/8$ gate has a misleading name (according to [2], this is for historical reasons), as it shifts the phase of $|1\rangle$ by $\pi/4$. To reduce confusion, we will refer to this gate as the T gate.

Next, we will look at gates that operate on more than one qubit. Figure 2.3 shows some common multi-qubit gates:

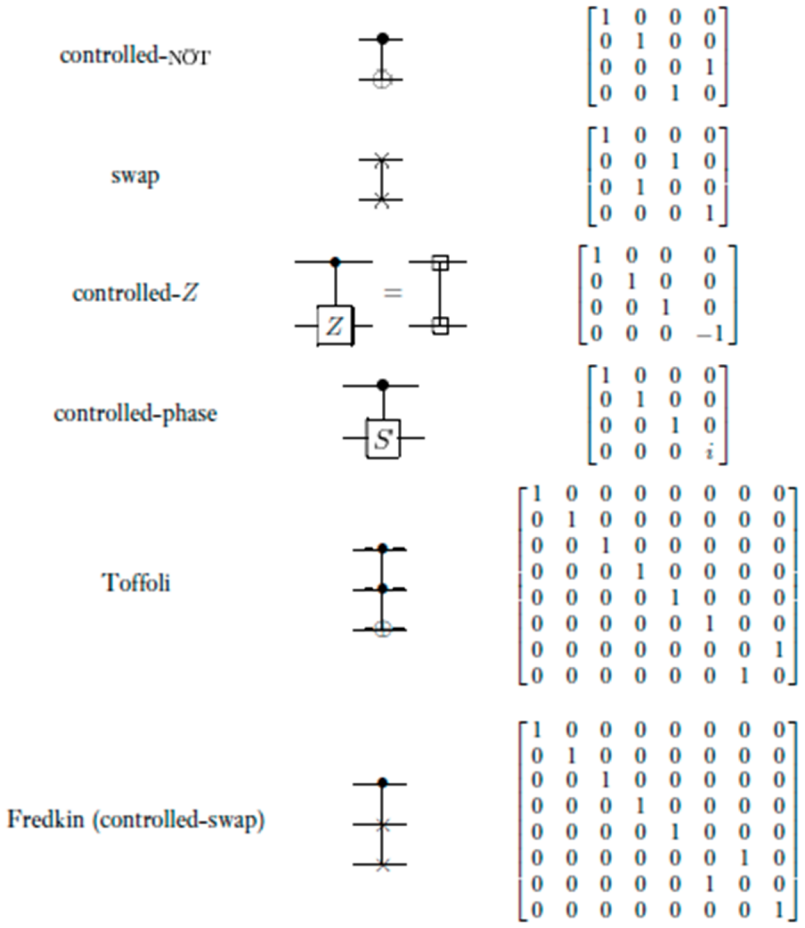

Figure 2.3: Multi-qubit gates [2].

The controlled-NOT, or CNOT, gate flips the second bit if the first bit is 1 [4]. The swap gate, as its name implies, swaps the two bits; $|01\rangle$ is transformed to $|10\rangle$, and vice versa, while $|00\rangle$ and $|11\rangle$ are unchanged. The controlled-Z gate transforms $|11\rangle$ to $-|11\rangle$; it does a

phase flip on the second bit if both bits are 1. The controlled-phase gate shifts the phase of the second bit by $\pi/2$ if both bits are 1. The Toffoli gate is also called the doubly-controlled NOT, or CCNOT, gate [5]; it flips the third bit if the first two bits are 1. The Fredkin, or controlled-swap gate, swaps the last two bits if the first bit is 1.

A Toffoli gate can also be used to implement an AND gate; if the input to the third bit is 0, the output on that bit will be 1 if and only if the first two bits are 1 [4]. It can also be used to implement a NAND gate; if the input to the third bit is 1, the output on that bit will be 0 if and only if the first two bits are 1 [8]. Because the NAND gate is a universal gate for classical computers [8], this shows that any computation that can be done on a classical computer can also be done on a quantum computer. However, it may be more efficient to perform to perform these computations on a classical computer. For this reason, a quantum computer may have a classical component attached to it [2]. In some cases, a quantum computer may be treated like a coprocessor in a classical computer, such as a GPU or an FPGA [17].

There are several sets of universal quantum gates. Nielsen and Chuang [2] show that any arbitrary unitary operation can be implemented using only CNOT and single-qubit gates. However, they also point out that this cannot always be done in a way that is resistant to errors. Hadamard, phase, CNOT, and T gates are a universal set of gates to approximate an arbitrary unitary operation [2]. But this approximation cannot always be done efficiently. So, while, in theory, any arbitrary unitary operation can be implemented on a quantum computer, not all unitary operations can be implemented efficiently [2].

2.6 Quantum Error Correction

Quantum algorithms work very well in theory, but in practice, the interaction of qubits with the external environment leads to decoherence, producing errors in the quantum state [4]. In [4], it is explained that quantum error correction uses an error correcting code that is designed to detect and correct a specific set of errors in the state. The code consists of a mapping that embeds n data bits into $n+k$ code bits. Thus, like classical error correction, quantum error correction uses redundant bits. A syndrome extraction operator maps $n+k$ code bits to a set of indices representing the set of errors that can be corrected. In other words, the syndrome extraction operator determines which error occurred, so that the error can be corrected by applying a transformation that has the inverse effect of the error. Quantum error correction is performed by applying the syndrome extraction operator to a quantum state containing the code bits plus enough bits to hold the index. The code bits may contain not just one error, but a linear combination of all the errors the code is designed to detect. After the syndrome extraction operator is applied, the state is a superposition of the different errors each associated with the corresponding error index. The next step is to measure the bits that hold the index; this yields a random index and changes the state so that the code bits contain the effects of only the error associated with the measured index. The inverse error transformation for that error can then be applied, allowing the original encoded state to be recovered.

2.7 Overview of Quantum Algorithms

As we pointed out earlier, quantum algorithms typically begin with a Hadamard gate being used to put the computer in a superposition of all possible states. It is important to note that quantum computing requires new, non-traditional techniques that use the unique properties of qubits and measurements to their advantage. In [4], two such techniques are highlighted. The first is amplitude amplification, which is the manipulation of the state to maximize the probability that values of interest are measured. Grover's algorithm, which is discussed in Chapter 3, is an example of amplitude amplification. The second technique is finding common properties in the value of a function. An example is Shor's algorithm, discussed in Chapter 4, which uses a quantum Fourier transform to find the period of a function.

Chapter 3

Attacks on Secret Key Cryptography

3.1 Overview

In secret key cryptography, two parties wish to communicate securely with each other. They share a key that is known only to them, and this key is used both for encryption and decryption of data. Two well-known examples are the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES). If the encryption algorithm is well-designed, the best possible attack is a brute-force attack. If the attacker has one or more plaintext-ciphertext pairs, a brute-force attack can be done by trying every possible key to decrypt the ciphertext, until a match is found with the known plaintext [6]. If the number of possible keys is $N = 2^n$, where n is the length of the key in bits, the average number of attempts needed is $N/2 = 2^{n-1}$ [6]. The number of attempts needed to determine the key increases exponentially as the key length increases. As we will see, Grover's algorithm can provide some speedup to this brute-force attack.

3.2 Grover's Algorithm

Grover's algorithm is designed to perform an unstructured search [4]. An unstructured search involves finding an item that satisfies a given condition in a list that is in random order. In this case, the average number of items that would need to be examined is $N/2$ [9]. (If the items were ordered in some way, such as alphabetically or numerically, a binary search could find the desired item faster). A brute-force attack on DES or AES is an unstructured search. The basic idea of the algorithm is to use superposition to examine every item in the list simultaneously [9]. Now, if we were able to examine the entire quantum state, only one iteration would be needed; but, as we have already seen, the nature of quantum measurement does not allow us access to the entire state. So, the approach of Grover's algorithm is to maximize the probability that the state corresponding to the desired item is measured; this is an example of amplitude amplification [3].

To begin, assume there are $N = 2^n$ items to be searched, and the search can be represented by a function *f(x)*, where *x* is between 0 and *N*-1 inclusive. *f(x)* is equal to 1 if item x satisfies a condition, and 0 otherwise [2]. For now, we need not be concerned with its inner workings, so we will treat *f(x)* as a black box. To execute the algorithm, we will need *n* qubits so that each item has its own quantum state, and one auxiliary qubit that is used to handle the output of *f(x)* [5].

The algorithm uses a quantum circuit known as an *oracle*, which performs the following transformation [2], which will be referred to as U_f [5]:

$$|x\rangle|q\rangle \rightarrow |x\rangle|q \oplus f(x)\rangle$$

The auxiliary, or oracle, qubit $|q\rangle$, is XOR'd with the output of *f(x)*. If *f(x)*=0, then $|q\rangle$ is unchanged. But, if *f(x)*=1, $|0\rangle \rightarrow |1\rangle$ and $|1\rangle \rightarrow |0\rangle$. An important case occurs when $|q\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$. In that case, if *f(x)*=1,

$$|x\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2} \rightarrow |x\rangle| \otimes (|1\rangle - |0\rangle)/\sqrt{2} = -|x\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2}$$

This effectively flips the sign of the state, while leaving the oracle qubit unchanged. For this case, the transformation can be summarized as follows [2]:

$$|x\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2} \rightarrow (-1)^{f(x)}|x\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2},$$

or,

$$|x\rangle \rightarrow (-1)^{f(x)}|x\rangle.$$

The algorithm begins, as all quantum algorithms do, with all qubits being set to $|0\rangle$. The oracle qubit is then set to $|1\rangle$ using the *X* gate. To complete initialization, the Hadamard gate is applied to all qubits, resulting in the following initial state [5]:

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

In the initial state, the first *n* qubits are in a superposition of all possible *x*. In general, the state can be represented as

$$|\psi\rangle = \sum_{x=0}^{N-1} \alpha_x |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

where $\alpha_x = 1/\sqrt{N}$ at initialization.

An iteration of Grover's algorithm consists of two steps: a sign flip on the state corresponding to the desired item and an inversion about the average for all items [5]. Because the oracle qubit is in the state $(|0\rangle - |1\rangle)/\sqrt{2}$, the sign flip can be done simply by applying the oracle. The resulting state is

$$U_f|\psi\rangle = \sum_{x=0}^{N-1} (-1)^{f(x)} \alpha_x |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Figure 3.1 illustrates the effect of the sign flip:

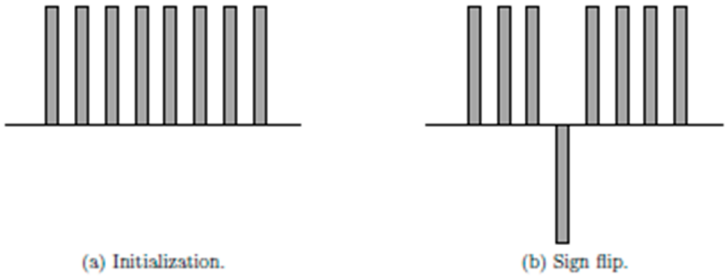

Figure 3.1: Grover's algorithm sign flip [5].

Again, we notice that the oracle qubit is unchanged. Since the oracle qubit does not change throughout the algorithm, we will ignore it from this point on.

The second step of the Grover iteration is inversion about the average. This is represented by the following transformation on the first n qubits [5]:

$$\sum_{j=0}^{N-1} \alpha_j |j\rangle \rightarrow \sum_{j=0}^{N-1} \left[2 \left(\sum_{k=0}^{N-1} \frac{\alpha_k}{N} \right) - \alpha_j \right]$$

Alternately, this can be represented as

$$\sum_{j=0}^{N-1} \alpha_j |j\rangle \rightarrow \sum_{j=0}^{N-1} (2A - \alpha_j)$$

where A is the average of all values of α. The term $2A - \alpha_j$ can be rewritten as $A + (A - \alpha_j)$ [9], which illustrates how the inversion works. Since the state that had its sign flipped is further away from the average, its magnitude ends up larger than those for the other states. This is illustrated by Figure 3.2:

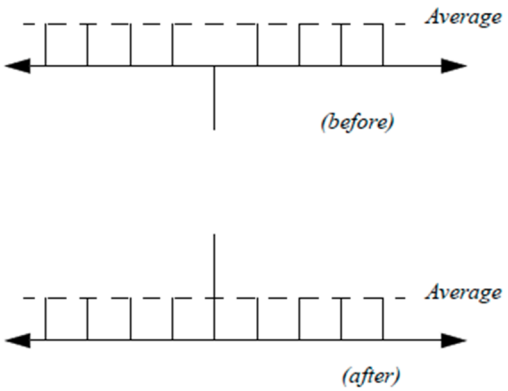

Figure 3.2: Inversion about the average [9].

Notice that the only state that is treated differently from the rest is the state for the desired item; only its sign is flipped, but all states are inverted about the average. Therefore, all undesired states will have the same magnitude, while the desired state will have a different magnitude.

The inversion about the average can be implemented by the following unitary matrix [5]:

$$W = \begin{bmatrix} 2/N - 1 & 2/N & \cdots & 2/N \\ 2/N & 2/N - 1 & \cdots & 2/N \\ \vdots & \vdots & \ddots & \vdots \\ 2/N & 2/N & \cdots & 2/N - 1 \end{bmatrix} = \begin{bmatrix} 2/N & 2/N & \cdots & 2/N \\ 2/N & 2/N & \cdots & 2/N \\ \vdots & \vdots & \ddots & \vdots \\ 2/N & 2/N & \cdots & 2/N \end{bmatrix} - I^{\otimes n}$$

This can be re-written as

$$-H^{\otimes n} D H^{\otimes n}, D = \begin{bmatrix} -1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

Note that the negative sign does not affect the probabilities of measuring any of the basis states, so it can safely be ignored. (This is a global phase factor [2] of $e^{i\pi}$.) Figure 3.3 shows a circuit that will implement *D*:

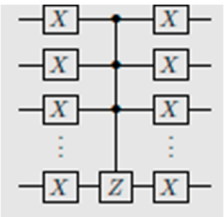

Figure 3.3: Grover's algorithm *D* circuit [5].

So, the full inversion about the average can be performed by applying Hadamard gates to all qubits before and after the D operation.

After an optimal number of iterations is performed, the first *n* qubits are measured. The desired state will have the highest probability of being measured, and that probability will be close to one. Figure 3.4 summarizes the algorithm:

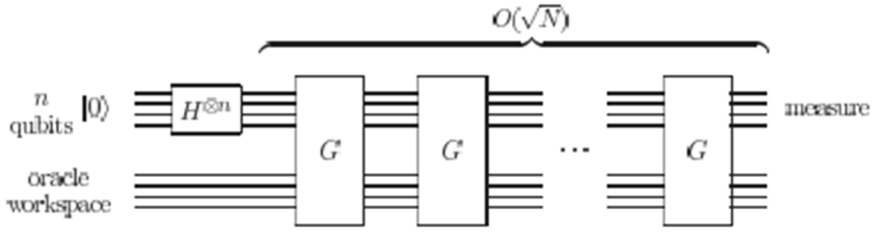

Figure 3.4: Summary of Grover's algorithm [2].

In Figure 3.4, *G* represents a single Grover iteration. In the next section, we will show how the optimal number of iterations is determined.

3.3 Optimal Number of Grover Iterations

As we just mentioned, the optimal number of iterations is such that the probability of measuring the desired state is as close to one as possible. Nannicini [5] shows how this can be illustrated geometrically. The state after the *k*-th iteration can be split into two components; one for the desired state, and the other for the undesired states:

$$|\psi_k\rangle = d_k|\psi_D\rangle + u_k|\psi_U\rangle$$

At initialization, $d_0 = 1/\sqrt{N}$ and $u_0 = \sqrt{(N-1)/N}$. By applying a Grover iteration to the state vector in this form, it can be shown that

$$d_{k+1} = \left(1 - \frac{1}{N-1}\right) d_k + \frac{2\sqrt{N-1}}{N} u_k$$

$$u_{k+1} = -\frac{2\sqrt{N-1}}{N} d_k + \left(1 - \frac{1}{N-1}\right) u_k$$

Inspection of the above equations shows that they have the form of a clockwise rotation matrix:

$$\begin{bmatrix} d_{k+1} \\ u_{k+1} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} d_k \\ u_k \end{bmatrix}$$

where

$$\sin\theta = \frac{2\sqrt{N-1}}{N};\ \cos\theta = 1 - \frac{1}{N-1}$$

and this satisfies the relationship $\sin^2\theta + \cos^2\theta = 1$. In other words, each Grover iteration is a rotation of a two-dimensional vector by an angle θ. We need to find the optimal number of iterations, *k*, such that d_k is as close to 1 and u_k is as close to 0 as possible. When *N* is large, we can use the approximation $N - 1 \approx N$; as a result, d_0 is almost 0, and u_0 is almost 1. Therefore, we want to rotate the vector $\pi/2$ radians to obtain the optimal solution. We can also use the small-angle approximation $\sin\theta \approx \theta$. The optimal number of iterations is then

$$k = \frac{\pi}{2\theta} \approx \frac{\pi N}{4\sqrt{N-1}} \approx \frac{\pi N}{4\sqrt{N}} = \frac{\pi}{4}\sqrt{N}$$

Figure 3.5 shows the rotation of the vector for *N*=65536. The first graph shows the initial state, represented by the black arrow, and the optimal state, represented by the green arrow. The second graph shows the state after 100 iterations, and the third graph shows the state after the optimal number of 201 iterations.

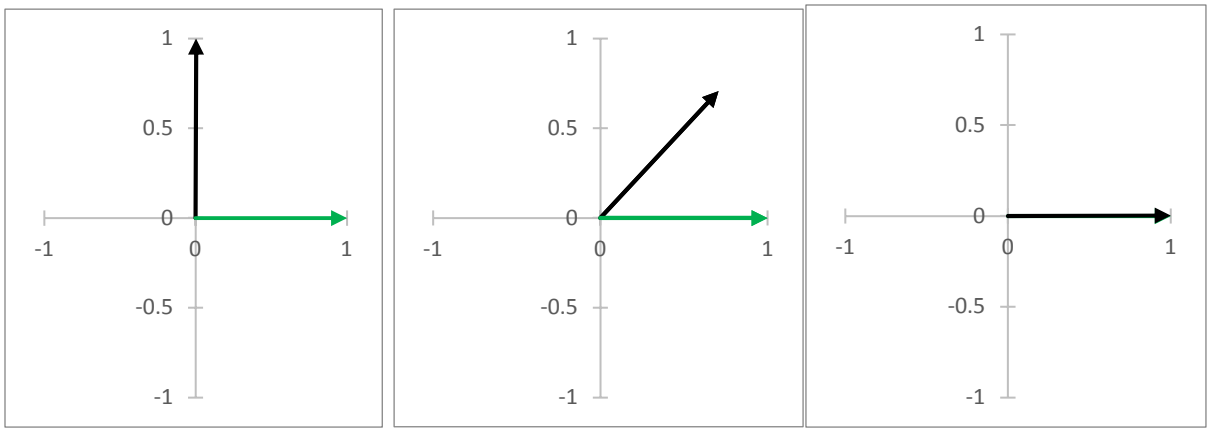

Figure 3.5: Grover's algorithm state for N=65536; initial, after 100 iterations, after 200 iterations.

One thing that is important to note about Grover's algorithm is that performing additional iterations does not converge further on the optimal solution; rather, it rotates the vector past the optimal solution. Figure 3.6 shows the state after 250 iterations.

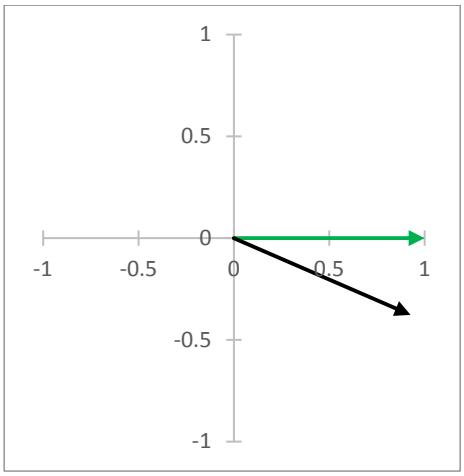

Figure 3.6: Grover's algorithm state for N=65536 after 250 iterations.

So, it is important to do exactly the optimal number of iterations.

3.4 Demonstration

The following tables and figures show the calculations of Grover's algorithm for N=8, N=16, and N=32, respectively. The tables have two columns for each iteration; the first column shows the result of the sign flip, while the second column shows the result of the inversion about the average. The desired item and the state after the optimal number of iterations are both highlighted. The figures show the amplitude of the desired state after each iteration.

Init	Iter 1		Iter 2		Iter 3	
0.353553	0.353553	0.176777	0.176777	-0.08839	-0.08839	-0.30936
0.353553	0.353553	0.176777	0.176777	-0.08839	-0.08839	-0.30936
0.353553	0.353553	0.176777	0.176777	-0.08839	-0.08839	-0.30936
0.353553	-0.35355	0.883883	-0.88388	0.972272	-0.97227	0.574524
0.353553	0.353553	0.176777	0.176777	-0.08839	-0.08839	-0.30936
0.353553	0.353553	0.176777	0.176777	-0.08839	-0.08839	-0.30936
0.353553	0.353553	0.176777	0.176777	-0.08839	-0.08839	-0.30936
0.353553	0.353553	0.176777	0.176777	-0.08839	-0.08839	-0.30936

Table 3.1: Grover's algorithm, N=8.

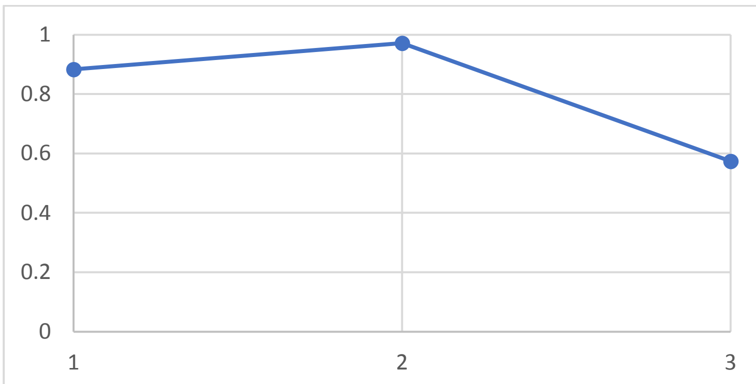

Figure 3.7: Amplitude of desired state vs. number of iterations, N=8.

Init	Iter 1		Iter 2		Iter 3		Iter 4		Iter 5	
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146
0.25	-0.25	0.6875	-0.6875	0.953125	-0.95313	0.980469	-0.98047	0.762695	-0.7627	0.354248
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146
0.25	0.25	0.1875	0.1875	0.078125	0.078125	-0.05078	-0.05078	-0.16699	-0.16699	-0.24146

Table 3.2: Grover's algorithm, N=16.

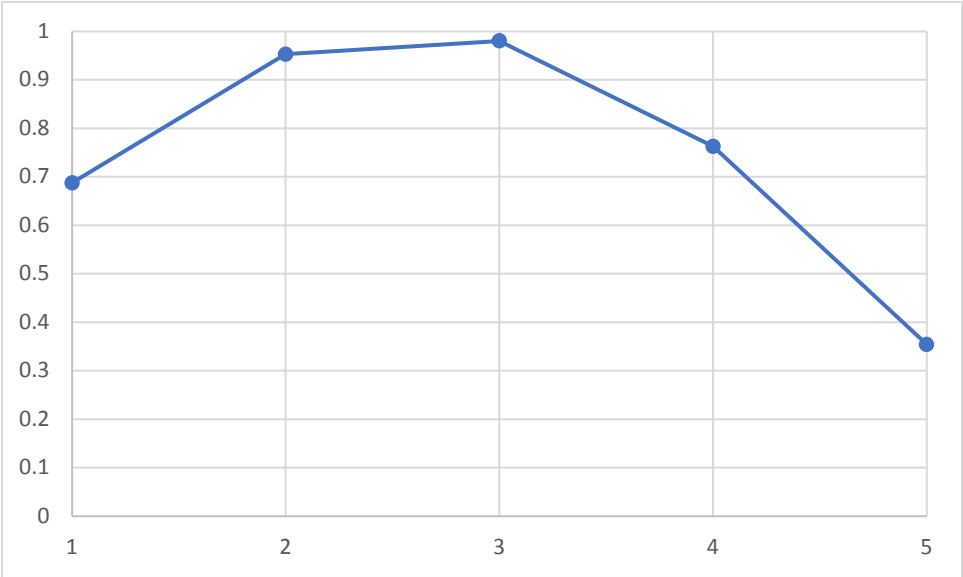

Figure 3.8: Amplitude of desired state vs. number of iterations, N=16.

Init	Iter 1		Iter 2		Iter 3		Iter 4		Iter 5		Iter 6		Iter 7		Iter 8	
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	-0.17678	0.508233	-0.50823	0.77616	-0.77616	0.947067	-0.94707	0.999591	-0.99959	0.927166	-0.92717	0.738845	-0.73885	0.458169	-0.45817	0.120221
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783
0.176777	0.176777	0.15468	0.15468	0.113248	0.113248	0.05766	0.05766	-0.00514	-0.00514	-0.06729	-0.06729	-0.12103	-0.12103	-0.15964	-0.15964	-0.1783

Table 3.3: Grover's algorithm, N=32.

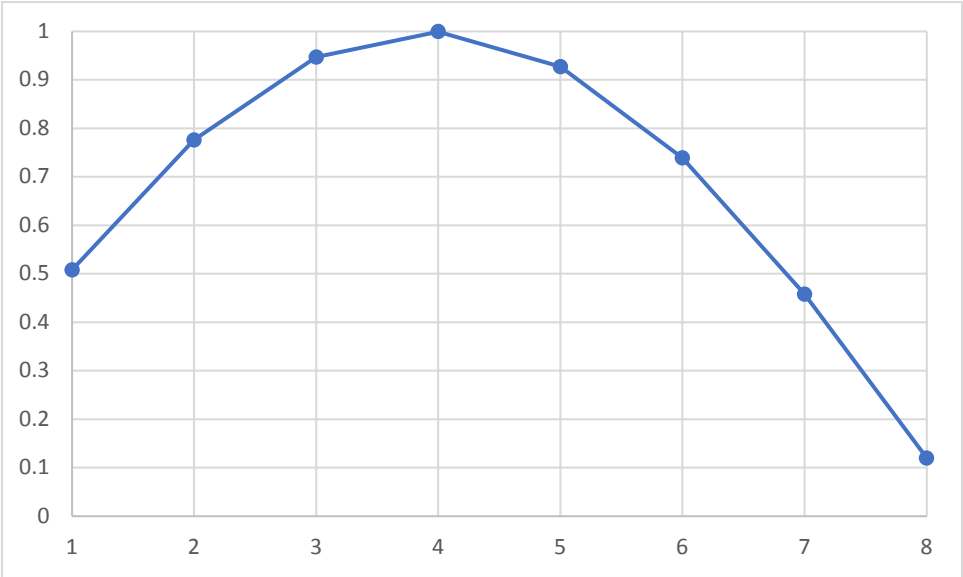

Figure 3.9: Amplitude of desired state vs. number of iterations, N=32.

The MATLAB script grover.m (listed in Appendix A) performs an emulation of Grover's algorithm by using matrix multiplications to simulate the various gates that are used. The optimal number of iterations are performed, and a bar graph is displayed showing the amplitude of each state. Figure 3.10 shows the output of grover(4,4); the first argument specifies 2^4=16 items, while the second argument specifies that the fourth item is the desired item.

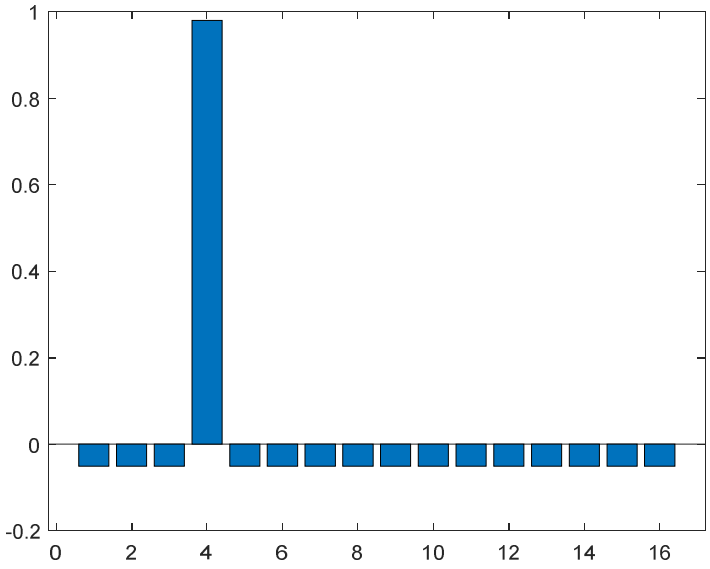

Figure 3.10: Output of grover(4,4).

The MATLAB script groveropt.m (listed in Appendix B) plots the amplitude of the desired state versus the number of iterations. Figure 3.11 shows the output of groveropt(16), where the argument specifies 2^{16}=65,536 items.

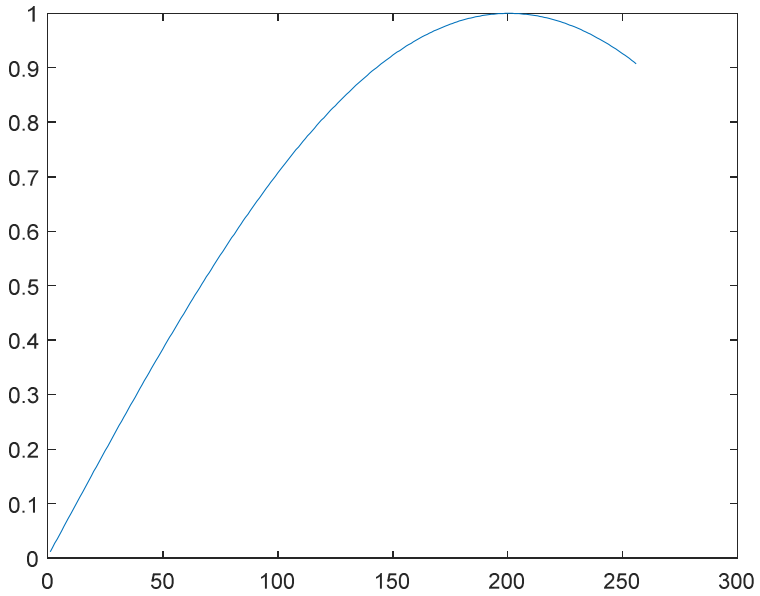

Figure 3.11: Output of groveropt(16).

3.5 Complexity

As we have shown, the number of iterations required for Grover's algorithm is $O(\sqrt{N})$, or $O(2^{n/2})$, while a classical unstructured search is $O(N)$, or $O(2^n)$. Thus, Grover's algorithm provides a quadratic speedup over the classical search [5]. While this reduces the amount of time needed to do a brute-force attack on DES or AES, this problem can easily be solved by doubling the length of the key. If the new key length is $2n$, the number of iterations required for Grover's algorithm becomes $O(2^n)$, which is the same as a classical search over the original key length. In addition, the speedup also depends on the efficiency with which the oracle can be implemented [3]. Research has been done on the implementation of Grover oracles for AES [11] [12]. The findings show that it can take up to nearly 7,000 qubits to implement an oracle for AES, making the oracle by far the most complicated part of the algorithm for this application. For these reasons, quantum computing does not pose a serious threat to secret key cryptography.

Chapter 4

Attacks on Public Key Cryptography

4.1 Overview

Public key cryptography is described in [6]. Each user generates a pair of keys; one key is made public, while the other is kept private. Anyone can send a user a confidential message by encrypting it with the recipient's public key, but only the recipient can decrypt the message with their private key. Similarly, a digital signature can be produced by encrypting with the private key; anyone can use the public key to decrypt and check the signature. Public key cryptography is also used to exchange secret keys for algorithms such as DES and AES. In general, public-key cryptographic algorithms derive their security from the difficulty of solving certain problems. For example, RSA, which we will examine in-depth in this chapter, bases its security on the difficulty of factoring large numbers. Another example is the Diffie-Hellman key exchange algorithm, which gets its security from the difficulty of computing discrete logarithms. Because of its difficulty, finding an efficient algorithm for factoring numbers is a topic of interest in computer science. The best classical factoring algorithms run in exponential time with respect to the number of bits [4]. But, on a quantum computer, Shor's algorithm can factor a number in polynomial time with respect to the number of bits [8].

4.2 The RSA Algorithm

The RSA algorithm is explained as follows in [6]. A user's public key is given by $\{e, N\}$, and their private key is given by $\{d, N\}$, where N is the product of two prime numbers p and q. Then, encryption is performed using the public key, by

$$C = M^e \bmod N$$

where M is the plaintext message and C is the corresponding ciphertext. Decryption is performed using the private key, by

$$M = C^d \bmod N = M^{ed} \bmod N$$

For this to work, e and d must be multiplicative inverses modulo $\phi(N)$, where $\phi(N)$ is the *Euler totient function* of N. This relationship is represented by the following equation:

$$ed \equiv 1 \bmod \phi(N)$$

If $N=pq$, and p and q are prime, then

$$\phi(N) = (p-1)(q-1)$$

Once $\phi(N)$ is determined, suitable values for e and d can be chosen.

The only values that are publicly known are e and N. But, if an attacker can factor N into p and q, the private key d can be determined. Unlike the brute-force attack on secret key algorithms, this attack does not require any known plaintext or ciphertext. The only information needed by the attacker is the user's public key, $\{e, N\}$.

4.3 The Quantum Fourier Transform

In Shor's algorithm, the problem of factoring a number is reduced to the problem of finding the period of a function [4], specifically, $a^x \bmod N$, where N is the number to be factored. As such, the Fourier transform is an important part of the algorithm.

The definition of the discrete Fourier transform (DFT) is as follows [7]:

$$Y_{k+1} = \sum_{j=0}^{N-1} y_{j+1} e^{-2\pi ijk/N}$$

If we define $f_{kj} = e^{-2\pi i(j-1)(k-1)/N}$, the above equation becomes

$$Y_k = \sum_{j=1}^{N} f_{kj} y_j$$

The DFT can now be written in matrix form: $Y = Fy$. To illustrate, we will use the case where N=4 as an example:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix} = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \\ f_{41} & f_{42} & f_{43} & f_{44} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} e^{-2\pi i(0)(0)/4} & e^{-2\pi i(1)(0)/4} & e^{-2\pi i(2)(0)/4} & e^{-2\pi i(3)(0)/4} \\ e^{-2\pi i(0)(1)/4} & e^{-2\pi i(1)(1)/4} & e^{-2\pi i(2)(1)/4} & e^{-2\pi i(3)(1)/4} \\ e^{-2\pi i(0)(2)/4} & e^{-2\pi i(1)(2)/4} & e^{-2\pi i(2)(2)/4} & e^{-2\pi i(3)(2)/4} \\ e^{-2\pi i(0)(3)/4} & e^{-2\pi i(1)(3)/4} & e^{-2\pi i(2)(3)/4} & e^{-2\pi i(3)(3)/4} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

The inverse transform is defined as follows [7]:

$$y_{j+1} = \frac{1}{N} \sum_{k=0}^{N-1} Y_{k+1} e^{2\pi ijk/N}$$

If we define $g_{jk} = \frac{1}{N} e^{2\pi i(j-1)(k-1)/N}$, this becomes

$$y_j = \sum_{k=1}^{N} g_{jk} Y_k$$

In matrix form, this become $y = GY$. Since G is the inverse transform, it follows that $GF = I$. By factoring $1/N$ out of G, and absorbing $1/\sqrt{N}$ back into F and G, we can redefine f_{kj} and g_{jk} as

$$f_{kj} = \frac{1}{\sqrt{N}} e^{-2\pi i(j-1)(k-1)/N};\ g_{jk} = \frac{1}{\sqrt{N}} e^{2\pi i(j-1)(k-1)/N}$$

By doing this, we have made the transform unitary, since it can be seen that $G = F^{\dagger}$. Because the transform is unitary, it can be implemented as a quantum circuit. The quantum Fourier transform (QFT) can be used when N is a power of 2 [4]. In the quantum computing literature, the QFT is defined with $e^{2\pi ijk/N}$, even though that is actually the inverse transform. We will follow the convention of the quantum computing literature from this point on. With that in mind, the QFT performs the following transformation on each basis state [2]:

$$|j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk/N} |k\rangle$$

Aside from the change in convention, this is equivalent to the matrix form we discussed earlier.

4.4 Factoring Using Shor's Algorithm

As we mentioned earlier, Shor's algorithm reduces the problem of factoring N to finding the period of $a^x \bmod N$. Table 4.1 lists the values of $a^x \bmod N$ for various a and x when N=33:

N	33																												
	a																												
x	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
2	4	9	16	25	3	16	31	15	1	22	12	4	31	27	25	25	27	31	4	12	22	1	15	31	16	3	25	16	9	4	1
3	8	27	31	26	18	13	17	3	10	11	12	19	5	9	4	29	24	28	14	21	22	23	30	16	20	15	7	2	6	25	32
4	16	15	25	31	9	25	4	27	1	22	12	16	4	3	31	31	3	4	16	12	22	1	27	4	25	9	31	25	15	16	1
5	32	12	1	23	21	10	32	12	10	11	12	10	23	12	1	32	21	10	23	21	22	23	21	1	23	12	10	32	21	1	32
6	31	3	4	16	27	4	25	9	1	22	12	31	25	15	16	16	15	25	31	12	22	1	9	25	4	27	16	4	3	31	1
7	29	9	16	14	30	28	2	15	10	11	12	7	20	27	25	8	6	13	26	21	22	23	18	31	5	3	19	17	24	4	32
8	25	27	31	4	15	31	16	3	1	22	12	25	16	9	4	4	9	16	25	12	22	1	3	16	31	15	4	31	27	25	1
9	17	15	25	20	24	19	29	27	10	11	12	28	26	3	31	2	30	7	5	21	22	23	6	4	14	9	13	8	18	16	32
10	1	12	1	1	12	1	1	12	1	22	12	1	1	12	1	1	12	1	1	12	22	1	12	1	1	12	1	1	12	1	1
11	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
12	4	9	16	25	3	16	31	15	1	22	12	4	31	27	25	25	27	31	4	12	22	1	15	31	16	3	25	16	9	4	1
13	8	27	31	26	18	13	17	3	10	11	12	19	5	9	4	29	24	28	14	21	22	23	30	16	20	15	7	2	6	25	32
14	16	15	25	31	9	25	4	27	1	22	12	16	4	3	31	31	3	4	16	12	22	1	27	4	25	9	31	25	15	16	1
15	32	12	1	23	21	10	32	12	10	11	12	10	23	12	1	32	21	10	23	21	22	23	21	1	23	12	10	32	21	1	32
p	10	5	5	10	10	10	10	5	2	2	1	10	10	5	10	10	10	10	10	2	1	2	10	5	10	5	10	10	10	10	2
a^(p/2) + 1 mod N	0			24	22	11	0		11	12		11	24		2	0	22	11	24	22		24	22		24		11	0	22	2	0
GCD(N, a^(p/2) + 1 mod N)	33			3	11	11	33		11	3		11	3		1	33	11	11	3	11		3	11		3		11	33	11	1	33
a^(p/2) - 1 mod N	31			22	20	9	31		9	10		9	22		0	31	20	9	22	20		22	20		22		9	31	20	0	31
GCD(N, a^(p/2) - 1 mod N)	1			11	1	3	1		3	1		3	11		33	1	1	3	11	1		11	1		11		3	1	1	33	1

Table 4.1: Values of a^x mod 33.

The table shows the periodic nature of $a^x \bmod N$. For example, if we focus on a=5, we see a pattern in 5^x mod 33 as x increases: 5, 25, 26, 31, 23, 16, 14, 4, 20, 1, 5, 25, 26, and so forth. We can see that 5^1=5 mod 33 and 5^{11}=5 mod 33; the pattern repeats once we reach x=11. Therefore, the period of 5^x mod 33 is 10. This is shown in the row labeled "p." Similar patterns can also be observed for the other values of a. If a is relatively prime to N, and $a^x = a^{x+p} \bmod N$, then $a^p = 1 \bmod N$ [3]. If p is even, then

$$a^p - 1 = 0 \bmod N$$
$$\left(a^{p/2} + 1\right)\left(a^{p/2} - 1\right) = 0 \bmod N$$

This means $a^{p/2} + 1$ and $a^{p/2} - 1$ have nontrivial factors with N, unless one of them is a multiple of N [3]. Finding the greatest common divisor (GCD) of $a^{p/2} + 1$ and N will yield one of the factors of N [3]. Since N is the product of two prime numbers, the other factor can be obtained at this point. From the table, we see that $a^{p/2} + 1 = a^5 + 1 = 24 \bmod 33$. Since the GCD of 24 and 33 is 3, we now have the factors of N: 3 and 11.

The first step in Shor's algorithm is to choose a value of a. If that value is not relatively prime to N, it can be used to find a factor of N, and we can stop [4]. Otherwise, we continue. Next, a value n is chosen such that $N^2 \leq 2^n < 2N^2$. We will compute $a^x \bmod N$ for $0 \leq x < 2^n$. The algorithm uses two quantum registers. The first register is n qubits long, and holds values of x. The second register is $\lceil \log_2 N \rceil$ qubits long and holds values of $a^x \bmod N$. At the beginning of the algorithm, all qubits are set to $|0\rangle$. Hadamard gates are then applied to the first register to put it in a superposition. The resulting state is [8]

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

Next, $a^x \bmod N$ is computed on the above superposition, using a quantum circuit that stores the values in the second register. After this step, the state is as follows [4]:

$$|\psi\rangle = \sum_{x=0}^{2^n-1} \frac{1}{\sqrt{2^n}} |x\rangle |a^x \bmod N\rangle$$

For example, if a=5 and N=33, the state is

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} (|0\rangle|1\rangle + |1\rangle|5\rangle + |2\rangle|25\rangle + |3\rangle|26\rangle + \cdots |10\rangle|1\rangle + |11\rangle|5\rangle + \cdots)$$

A quantum Fourier transform is then performed on the first register. Because this is an entangled state, we need to consider both registers. To illustrate how this works, we will look at the simplest case possible, where each register has one qubit. Such a state can be represented as

$$\alpha_0|0\rangle|0\rangle + \beta_0|0\rangle|1\rangle + \alpha_1|1\rangle|0\rangle + \beta_1|1\rangle|1\rangle$$

The matrix representing a QFT on the first register is

$$\begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix} \otimes I = \begin{bmatrix} f_{11} & 0 & f_{12} & 0 \\ 0 & f_{11} & 0 & f_{12} \\ f_{21} & 0 & f_{22} & 0 \\ 0 & f_{21} & 0 & f_{22} \end{bmatrix}$$

Applying the above matrix gives us

$$\begin{bmatrix} f_{11} & 0 & f_{12} & 0 \\ 0 & f_{11} & 0 & f_{12} \\ f_{21} & 0 & f_{22} & 0 \\ 0 & f_{21} & 0 & f_{22} \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \beta_0 \\ \alpha_1 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} f_{11}\alpha_0 + f_{12}\alpha_1 \\ f_{11}\beta_0 + f_{12}\beta_1 \\ f_{21}\alpha_0 + f_{22}\alpha_1 \\ f_{21}\beta_0 + f_{22}\beta_1 \end{bmatrix}$$

This shows that we have done two Fourier transforms; one on all states where the second register is $|0\rangle$, and one on all states where the second register is $|1\rangle$. Therefore, the QFT performs a separate Fourier transform for every group of states that have the same value in the second register.

After the QFT, the next step is to measure the state. Because the results of Fourier transform will have spikes where the first register is a multiple of $2^n/p$, we are likely to measure a value v for the first register that is either a multiple j of $2^n/p$, or very close. If p is a power of 2, we are guaranteed to measure $v = j(2^n/p)$ [4].

Once the measurement is obtained, the remainder of the algorithm can be performed classically [8]. If p is a power of 2, and j and p are relatively prime, reducing $v/2^n$ to its lowest terms will yield a fraction with p in its denominator [4]. Otherwise, the continued fraction expansion of $v/2^n$ can be used to obtain p [4]. If the period is even, we can find the factors of N as described at the beginning of this section.

In [4], several cases are pointed out where Shor's algorithm will not give a factor of N. These include the period being odd, the measured value not being close enough to a multiple of $2^n/p$, the period and the multiplier j having a common factor, which makes us unable to determine the period by reducing $v/2^n$, and the algorithm returning N itself as a factor of N. It is also possible that the measured value will be zero, in which case we cannot obtain the period. Shor [8] explains that repeating the algorithm $O(\log\log p)$ times will have a high probability of returning a factor of N.

4.5 Demonstration

The MATLAB script shor.m (listed in Appendix C) simulates Shor's algorithm and plots the probabilities of measuring each possible value of the first register. Figure 4.1 shows the output of shor(33,5), where the arguments indicate that N=33 and a=5.

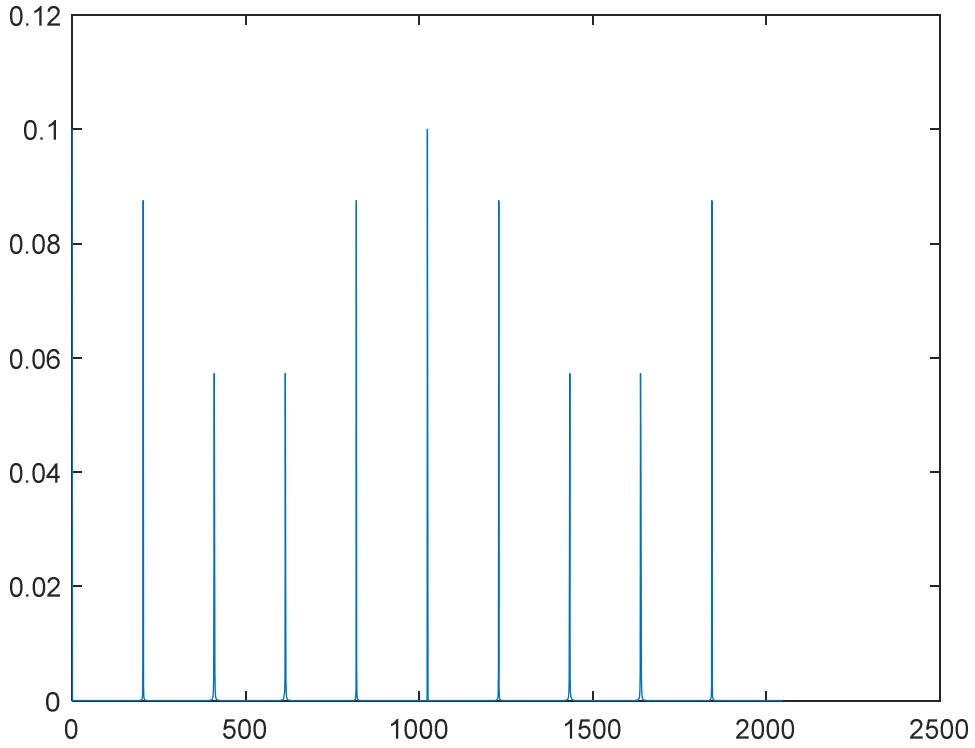

Figure 4.1: Output of shor(33,5).

There are 2^{11}=2,048 states in the first register (33^2=1,089; $1{,}089 \leq 2{,}048 < 2{,}178$). The spikes occur at the values of 205, 410, 614, 819, 1,024, 1,229, 1,434, 1,638, and 1,843. Because $205/2{,}048 \approx 1/10$, it follows that the period is 10.

4.6 Complexity

The circuit that calculates the modular exponents combines repeated squaring with a quantum implementation of classical circuits for multiplication [8] (recall that any classical computation can be implemented on a quantum computer). The only input to the circuit is *x*; values involving *a* and *N* are precomputed classically and built into the circuit [8]. The number of gates in this circuit is $O(n^3)$ [8]. The QFT circuit is implemented with a combination of Hadamard gates that operate on a single qubit, and the following gate that operates on qubits *j* and *k*, where $j < k$ [8]:

$$S_{j,k} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta_{k-j}} \end{bmatrix}, \theta_{k-j} = \frac{\pi}{2^{k-j}}$$

This gate is a conditional phase shift. If bits *j* and *k* are both 1, a phase shift of $\pi/2^{k-j}$ is applied. Otherwise, the bits are left unchanged. If $k - j = 1$, this is identical to an *S* gate, since $\theta_{k-j} = \pi/2$. If $k - j = 2$, this is identical to a *T* gate, since $\theta_{k-j} = \pi/4$. To obtain the QFT, the transformations are performed in the following order, followed by a bit reversal [8]:

$$H_{n-1}, S_{n-2,n-1}, H_{n-2}, S_{n-3,n-2}, S_{n-3,n-2}, H_{n-3}, \cdots, H_1, S_{0,n-1}, S_{0,n-2}, \cdots, S_{0,2}, S_{0,1}, H_0$$

The number of gates in the QFT circuit is $n(n-1)/2$, which is $O(n^2)$ [8]. Therefore, the complexity of Shor's algorithm is polynomial with respect to the number of bits in *N*. However, the best classical algorithm for factoring numbers is exponential with respect to the number of bits. As a result, Shor's algorithm provides a much more dramatic speedup over its classical counterpart than Grover's algorithm.

The number of qubits needed for the first register is $\lceil 2 \log_2 N \rceil$, while the number of qubits needed for the second register is $\lceil \log_2 N \rceil$. This means thousands of qubits would be needed to factor numbers of the size that are typically used for RSA.

Because of the speedup provided by Shor's algorithm (from exponential to polynomial), increasing the number of bits in *N* will not provide protection against an attack. However, it may provide some protection if it makes the number of qubits needed for Shor's algorithm larger than what is currently possible with quantum computers. But this also degrades the performance of RSA, which is already slow in comparison to secret key algorithms. When quantum computers scale up to the number of qubits required to run Shor's algorithm, it will pose a serious threat to RSA and other currently used public key algorithms.

Chapter 5

Post-Quantum Cryptography

Because of the growth in the development of quantum computers, there has been research devoted to finding cryptographic algorithms that are safe from attacks carried out on quantum computers. This is known as post-quantum cryptography. It is typically focused on public-key cryptography, since, as we have shown, it faces the biggest threat from quantum computers. Much research centers around several classes of cryptographic systems for which no attacks using Shor's algorithm are known to exist. These include hash-based, code-based, lattice-based, and multivariate-quadratic-equations cryptography [13] [15].

The National Institute of Standards and Technology (NIST), noticing the recent progress in the development of quantum computers, has begun the process of standardizing post-quantum cryptography [16]. In 2019, this process entered its second round, and, at the time this report was written, NIST was in the process of evaluating the algorithms that made it to the second round. This process could take two to three years, as time needs to be taken to analyze the candidate algorithms and perform cryptanalysis on them.

One challenge in the area of post-quantum cryptography is that, while Shor's algorithm is well-known, and a good area to focus on, it is not possible to know what kinds of quantum algorithms will be developed in the future. It is always possible that new quantum algorithms could break cryptographic algorithms previously believed to be secure.

Chapter 6

Future Work

An area for future study is the implementation of the examples throughout this report on Qiskit and/or Q#. Qiskit [18] is a Python-based library for developing quantum circuits, while Q# [17] is Microsoft's language for developing quantum circuits.

Chapter 7

Conclusion

Quantum computers have the potential to solve certain problems faster than classical computers currently can. One notable example is an unstructured search, for which Grover's algorithm provides a quadratic speedup. A much more noticeable example is number factoring, which Shor's algorithm can do in polynomial time, while the best that classical algorithms can do is exponential time. However, a side effect of the speedup provided by these algorithms is a compromise to the security of cryptographic systems currently in use. If large-scale quantum computers become a reality, Grover's algorithm could reduce the amount of time needed to do a brute-force attack on secret-key algorithms such as AES. But this is easily mitigated by doubling the key length. Shor's algorithm is the more serious threat by far. If a quantum computer can execute this algorithm, it can render many public-key algorithms, such as RSA, insecure. For this reason, research is currently being done on post-quantum cryptography, and NIST is currently in the process of developing standards for it. There is much work that needs to be done in order to develop public-key cryptography that is secure against attacks from quantum computers.

References

[1] D. Griffiths, *Introduction to Quantum Mechanics*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[2] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information, 10th Anniversary Edition*. Cambridge, UK: Cambridge University Press, 2010.

[3] E. Rieffel and W. Polak, *Quantum Computing-A Gentle Introduction*. Cambridge, MA: MIT Press, 2011.

[4] E. Rieffel and W. Polak, "An introduction to quantum computing for non-physicists", ACM Comput. Surv., vol. 32, no. 3, pp. 300-335, Sep. 2000.

[5] G. Nannicini, "An Introduction to Quantum Computing Without the Physics", Feb. 2018., https://arxiv.org/abs/1708.03684

[6] W. Stallings, *Cryptography and Network Security: Principles and Practice, Sixth Edition.* Boston, MA: Pearson, 2014.

[7] A. Garcia, *Numerical Methods for Physics*. Englewood Cliffs, NJ: Prentice-Hall, 1994.

[8] P. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer", SIAM J. Computing 26, pp. 1484-1509, 1997.

[9] L. Grover, "A fast quantum mechanical algorithm for database search", Proc. 28th Ann. ACM Symp. on Theory of Comput., 212219, 1996.

[10] R. Feynman, “Simulating physics with computers,” *International Journal of Theoretical Physics*, Vol. 21, Nos. 6/7, pp. 467-488, 1982.

[11] M. Grassl, B. Langenberg, M. Roetteler and R. Steinwandt, "Applying Grover's Algorithm to AES: Quantum Resource Estimates", *Proc. PQCrypto'16*, pp. 29-43, 2016.

[12] S. Jaques, M. Naehrig, M. Roetteler and F. Virdia, "Implementing Grover oracles for quantum key search on AES and LowMC", 2019, [online] Available: https://eprint.iacr.org/2019/1146.

[13] D. Bernstein, J. Buchmann, E. Dahmen (editors). *Post-quantum cryptography*. Berlin, Springer, 2009.

[14] IBM | Quantum Computing, https://www.ibm.com/quantum-computing/

[15] Call for Proposals - Post-Quantum Cryptography | CSRC, https://csrc.nist.gov/Projects/post-quantum-cryptography/Post-Quantum-Cryptography-Standardization/Call-for-Proposals

[16] Post-Quantum Cryptography | CSRC, https://csrc.nist.gov/Projects/post-quantum-cryptography

[17] The Q# Programming Language - Microsoft Quantum | Microsoft Docs, https://docs.microsoft.com/en-us/quantum/language/

[18] Qiskit, https://qiskit.org/

Appendix A

Listing of grover.m

```
function x=grover(n,p)
N=2^n; % The total number of items to search
assert(N>1);
assert(N<=1024);
assert(p>0);
assert(p<=N);
% Set up oracle
Uf=eye(N*2);
Uf(2*p-1,2*p-1)=0;
Uf(2*p,2*p)=0;
Uf(2*p,2*p-1)=1;
Uf(2*p-1,2*p)=1;
X=[0 1;1 0];
InX=kron(eye(N),X);
H=1;
for j=1:log2(N)+1
    H=kron(H,[1 1;1 -1]/sqrt(2));
end
W=repmat(2/N,N)-eye(N);
WxI=kron(W,eye(2));
% Set up a superposition of all states tensored with (|0> -
|1>)/sqrt(2)
y=zeros(N*2,1);
y(1)=1;
y=InX*y;
y=H*y;
% Calculate optimal number of iterations
k=(pi/4)*sqrt(N)
for j=1:k
    % Apply to oracle to sign flip the desired item
    y=Uf*y;
    % Invert all states about the average
    y=WxI*y;
end
% Extract first n qubits (the oracle qubit never changes)
x=zeros(N,1);
for j=1:N
    x(j)=y(2*j-1)*sqrt(2);
end
bar(x);
```

Appendix B

Listing of groveropt.m

```
function x=groveropt(n)
N=2^n
s=2*sqrt(N-1)/N;
c=1-2/N;
R=[c s;-s c];
d=1/sqrt(N);
u=sqrt((N-1)/N);
k=(pi/4)*sqrt(N)
for j=1:ceil(sqrt(N))
    du=[d u]';
    du=R*du;
    d=du(1);
    u=du(2);
    x(j)=d;
end
plot(x);
```

Appendix C

Listing of shor.m

```
function ym=shor(M,a)
% This function performs the quantum Fourier transform of Shor's
algorithm
%  by simulating the changes that occur in the quantum state as
the
%  algorithm is executed
% ym=shor(M,a)
% Inputs
%   M = The number we are factoring
%   a = Arbitrary number such that 0 < a < M
% Output
%   ym = The probabilities of measuring each outcome of the QFT
m=ceil(log2(M*M)); % M^2 <= 2^m < 2*M^2
L=ceil(log2(M));
em=2^m;
eL=2^L;
emL=em*eL; % Total number of quantum states
% State space ranges from |0,0> to |2^m-1,2^L-1>
y=zeros(1,emL);
yt=zeros(1,emL);
% Create superposition of states |x,a^x mod M> for all x where 0
<= x < 2^m
% Since a^0 = 1, the first state in the superposition is |0,1>
axM = 1;
y(1+axM)=1/sqrt(em); % |0,1> is 2nd element; |0,0> is 1st
(MATLAB uses 1-based indices)
for x=1:em-1
    axM=mod(axM*a, M);
    y(1+axM+x*eL)=1/sqrt(em);
end
% Perform quantum Fourier transform on the first m qubits
% The last L qubits are unchanged
% Therefore, the actual state transition matrix is the tensor
product
%  of the QFT matrix and a (2^L)x(2^L) identity matrix
% Because the full matrix is very large, we use a for loop to do
the
%  matrix multiplication instead of building the full matrix
for u=1:eL
    yt(u:eL:emL)=fft(y(u:eL:emL))/sqrt(em);
```

```
end
% Plot and return the probabilities of measuring each possible
state of the
% first m qubits; the peaks are used to determine the period of
a^x mod M
ym=zeros(1,em);
for x=0:em-1
    ym(1+x)=norm(yt(1+eL*x:eL*(x+1)))^2;
end
plot(0:em-1,ym);
return;
```